

Content-aware Load Balancing for Distributed Backup

Fred Dougliis
EMC
Fred.Dougliis@emc.com

Deepti Bhardwaj
EMC
Deepti.Bhardwaj@emc.com

Hangwei Qian*
Case Western Reserve Univ.
Hangwei.Qian@case.edu

Philip Shilane
EMC
Philip.Shilane@emc.com

Abstract

When backing up a large number of computer systems to many different storage devices, an administrator has to balance the workload to ensure the successful completion of all backups within a particular period of time. When these devices were magnetic tapes, this assignment was trivial: find an idle tape drive, write what fits on a tape, and replace tapes as needed. Backing up data onto deduplicating disk storage adds both complexity and opportunity. Since one cannot swap out a filled disk-based file system the way one switches tapes, each separate backup appliance needs an appropriate workload that fits into both the available storage capacity and the throughput available during the backup window. Repeating a given client's backups on the same appliance not only reduces capacity requirements but it can improve performance by eliminating duplicates from network traffic. Conversely, any reconfiguration of the mappings of backup clients to appliances suffers the overhead of repopulating the new appliance with a full copy of a client's data. Reassigning clients to new servers should only be done when the need for load balancing exceeds the overhead of the move.

In addition, deduplication offers the opportunity for *content-aware* load balancing that groups clients together for improved deduplication that can further improve both capacity and performance; we have seen a system with as much as 75% of its data overlapping other systems, though overlap around 10% is more common. We describe an approach for clustering backup clients based on content, assigning them to backup appliances, and adapting future configurations based on changing requirements while minimizing client migration. We define a cost function and compare several algorithms for minimizing this cost. This assignment tool resides in a tier between backup software such as EMC NetWorker and deduplicating storage systems such as EMC Data Domain.

*Work done during an internship.

Tags: backups, configuration management, infrastructure, deduplication

1 Introduction

Deduplication has become a standard component of many disk-based backup storage environments: to keep down capacity requirements, repeated backups of the same pieces of data are replaced by references to a single instance. Deduplication can be applied at the granularity of whole files, fixed-sized blocks, or variable-sized “chunks” that are formed by examining content [12].

When a backup environment consists of a handful of systems (or “clients”) being backed up onto a single backup appliance (or “server”), provisioning and configuring the backup server is straightforward. An organization buys a backup appliance that is large enough to support the capacity requirements of the clients for the foreseeable future, as well as capable of supporting the I/O demands of the clients. That is, the backup appliance needs to have adequate *capacity* and *performance* for the systems being backed up.

As the number of clients increases, however, optimizing the backup configuration is less straightforward. A single backup administration domain might manage thousands of systems, backing them up onto numerous appliances. An initial deployment of these backup appliances would require a determination of which clients to back up on which servers. Similar to the single-server environment, this assignment needs to ensure that no server is overloaded in either capacity or performance requirements. But the existence of many available servers adds a new dimension of complexity in a deduplicating environment, because some clients may have more content in common than others. Assigning similar clients to the same server can gain significant benefits in capacity requirements due to the improved deduplication; in a constrained environment, assigning clients in a content-aware fashion can make the difference between meeting

one's capacity constraints and overflowing the system.

The same considerations apply in other environments. For example, the "clients" being backed up might actually be virtual machine images. VMs that have been cloned from the same "golden master" are likely to have large pieces in common, while VMs with different histories will overlap less. As another example, the systems being copied to the backup appliance might be backup appliances themselves: some enterprises have small backup systems in field offices, which replicate onto larger, more centralized, backup systems for disaster recovery.

Sending duplicate content to a single location can not only decrease capacity requirements but also improve performance, since content that already exists on the server need not be transferred again. Eliminating duplicates from being transmitted is useful in LAN environments [5] and is even more useful in WAN environments.

Thus, in a deduplicating storage system, *content-aware* load balancing is desirable to maximize the benefits of deduplication. There are several considerations relating to how best to achieve such balance:

Balancing capacity and throughput Above all, the system needs to assign the clients in a fashion that minimizes hot spots for storage utilization or throughput. Improvements due to overlap can further reduce capacity requirements.

Identifying overlap How does the system identify how much different clients have in common?

Efficiency of assignment What are the overheads associated with assignment?

Coping with overload If a server becomes overloaded, what is the best way to adapt, and what are the costs of moving a client from that server?

Our paper has three main contributions:

1. We define a *cost function* for evaluating potential assignments of clients to backup servers. This function permits different configurations to be compared via a single metric.
2. We present several techniques for performing these assignments, including an iterative refinement heuristic for optimizing the cost function in a content-aware fashion.
3. We compare multiple methods for assessing content overlap, both for collecting content and for clustering that content to determine the extent of any overlap.

Our assignment algorithm serves as a middleware layer that sits between the backup software and the underlying backup storage appliances. Our ultimate goal is

a fully automated system that will dynamically reconfigure the backup software as needed. As an initial prototype, we have developed a suite of tools that assess overlap, perform initial assignments by issuing recommendations for client-server assignments, and compute updated assignments when requirements change. Client assignments can be converted into a sequence of commands to direct the backup software to (re)map clients to specific storage appliances.

The rest of this paper is as follows. The next section provides more information about deduplication for backups and other related work. §3 provides use cases for the tool. §4 describes load balancing in more detail, including the "cost" function used to compare configurations and various algorithms for assignment of clients to servers. §5 discusses the question of content overlap and approaches to computing it. §6 presents results of simulations on several workloads. §7 examines some alternative metrics and approaches. Finally, §8 discusses our conclusions and open issues.

2 Background and Related Work

2.1 Evolving Media

In the past decade, many backup environments have evolved from tape-centric to disk-centric. Backup software systems, such as EMC NetWorker [6], IBM Tivoli Storage Manager [9], or Symantec NetBackup [19], date to the tape-based era. With tapes, a backup server could identify a pool of completely equivalent tape drives on which to write a given backup. When data were ready to be written, the next available tape drive would be used. Capacity for backup was not a critical issue, since it would usually be simple to buy more magnetic tape. The main constraint in sizing the backup environment would be to ensure enough throughput across the backup devices to meet the "backup window," i.e., the time in which all backups must complete. Some early work in this area includes the Amanda Network Backup Manager [16, 17], which parallelized workstation backups and created schedules based on anticipated backup sizes. Interleaving backup streams is necessary to keep the tapes busy and avoid "shoe-shining" from underfull buffers, but this affects restore performance [20]. The equivalence of the various tape drives, however, made parallelization and interleaving relatively straightforward.

Disk-based backup grew out of the desire to have backup data online and immediately accessible, rather than spread across numerous tapes that had to be located, mounted, and sequentially accessed in case of data loss. Deduplication was used to reduce the capacity requirements of the backup system, in order to permit disk-

based backup to compete financially with tape. The most common type of deduplication breaks a data stream into “chunks,” using features of the data to ensure that most small changes to the data do not affect the chunk boundaries. This way, inserting a few bytes early in a file might change the chunk where the insertion occurs, but the rest of the file will deduplicate. Deduplicating systems use a strong hash (known as a “fingerprint”) of the content to identify when a chunk already exists in the system [15, 21].

2.2 Deduplication: Challenges and Opportunities

With deduplicated disk backups replacing tape, the equivalence of appliances is partly lost. Writing to the same storage system gains efficiencies by suppressing duplicate data; these efficiencies can be further reflected back to the backup server or even the client being backed up, if the duplicates are identified before data cross the network [5]. The effort of dividing the content into chunks and computing fingerprints over the chunks can be distributed across the backup infrastructure, allowing the storage appliance to scale to more clients and reducing network traffic when the deduplication rate is high.

Thus, the “stickiness” of the assignment of a client to a storage appliance changes the role of the backup administrator. Instead of simply pooling many clients across many tape drives, the mapping of clients to storage appliances needs to be done *a priori*. Once a client has been paired with a particular storage appliance, it gets great benefits from returning to that appliance and omitting duplicates. Should it move to a different appliance, it must start over, writing all of its data anew. But if its target appliance is overloaded, the client queues up and waits longer than desired, possibly causing the backup not to complete within its “backup window.” Capacity is similarly problematic, since a client that is being backed up onto a full storage appliance either is not protected or must move to another less loaded system and pay a cost for copying data that would otherwise have been suppressed through deduplication.

In summary, once a client is backed up onto a particular storage appliance, there is a tension between the benefits of continuing to use it and the disadvantages that may ensue from overload; at some tipping point, the client may move elsewhere. It then pays a short-term overhead (lack of deduplication) but gets long-term benefits.

Another interesting challenge relating to deduplicating storage is anticipating when it will fill up. One needs to consider not only how much is written but also how well that data will deduplicate. Predictions of future capacity requirements on a device-by-device basis, based on mining past load patterns [2], would feed into our load balancing framework.

2.3 Load Balancing

Finally, the idea of mapping a set of objects to a set of appropriate containers is well-known in the systems community. Load balancing of processor-intensive applications has been around for decades [3, 8], including the possibility of dynamically reassigning tasks when circumstances change or estimates prove to be inaccurate [13]. More recently, allocating resources within grid or cloud environments is the challenge. Allocating virtual resources within the constraints of physical datacenters is particularly problematic, as one must deal with all types of resources: processor, memory, storage, and network [18]. There are many examples of provisioning systems that perform admission control, load balancing, and reconfiguration as requirements change (e.g., [7]), but we are unaware of any work that does this in the context of deduplication.

3 Use Cases

In this section we describe the motivation behind this system in greater detail. Figure 1 demonstrates the basic problem of assigning backups from clients to deduplicated storage systems, and there are a number of ways in which automated content-aware assignment can be useful.

Sizing and deployment Starting with a “clean slate,” an administrator may have a large number of client machines to be backed up on a number of deduplicating storage appliances. The assignment tool can use information about the size of each client’s backups, the throughput required to perform the backups, the rate of deduplication within each client’s backups, the rate at which the backup size is expected to change over time, and other information. With this data it can estimate which storage appliances will be sufficient for this set of clients. Such “sizing tools” are commonplace in the backup industry, used by vendors to aid their customers in determining requirements. Using information about overlapping content across clients allows the tool to refine its recommendations, potentially lowering the total required storage due to improved deduplication.

First assignment Whether the set of storage appliances is determined via this tool or in another fashion, once the capacity and performance characteristics of the storage appliances are known, the tool can recommend which clients should be assigned to which storage system. For the first assignment, we assume that no clients are already backed up on any storage appliance, so there is no benefit (with respect to deduplication) to preferring one appliance over another for

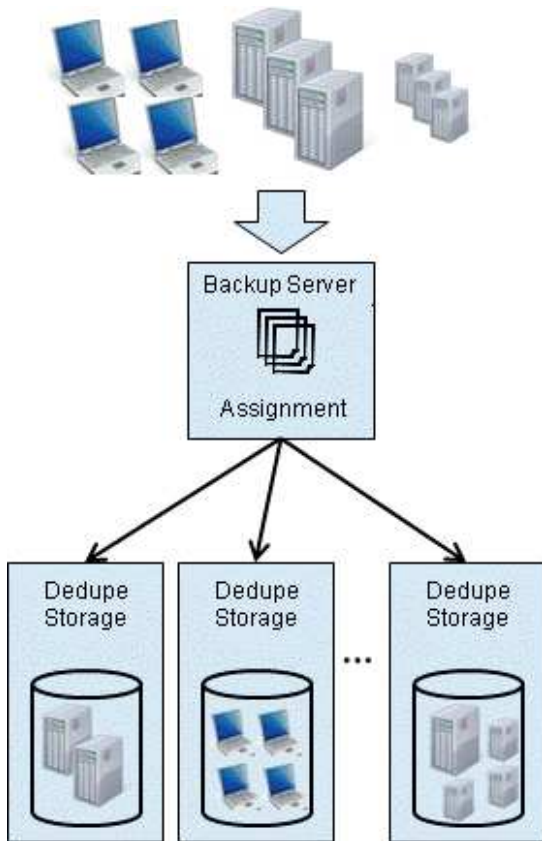


Figure 1: Backups from numerous clients are handled by the backup manager, which assigns clients to deduplicated storage while accounting for content overlap between similar clients. In this case, there was insufficient room on a single storage node for all three large servers, so one was placed elsewhere.

individual clients, though overlapping content is still a potential factor.

Reconfigurations Once a system is in steady state, there are a number of possible changes that could result in reconfiguration of the mappings. Clients may be added or removed, and backup storage appliances may be added. Storage may even be removed, especially if backups are being consolidated onto a smaller number of larger-capacity servers. Temporary failures may also require reconfiguration. Adding new clients and backup storage simultaneously may be the simplest case, in which the new clients are backed up to the new server(s). More commonly, extra backup capacity will be required to support the growth over time of the existing client population, so existing clients will be spread over a larger number of servers.

Disaster recovery As mentioned in the introduction, the “clients” might be backup storage appliances them-

selves, which are being replicated to provide disaster recovery (DR). In terms of load balancing, there is little distinction between backing up generic computers (file servers, databases, etc.) and replicating deduplicating backup servers. However, identifying content overlap is easier in the latter case because the content is already distilled to a set of fingerprints. Also, DR replication may be performed over relatively low-bandwidth networks, increasing the impact of any re-configuration that results in a full replication to a new server.

4 Load Balancing and Cost Metrics

In order to assign clients to storage appliances, we need a method for assessing the relative desirability of different configurations, which is done with a *cost* function described in §4.1. Given this metric, there are different ways to perform the assignment and evaluate the results. In §4.2, we describe and compare different approaches, both simple single-pass techniques that do not explicitly optimize the cost function and a heuristic for iteratively minimizing the cost.

4.1 Cost Function

The primary goal of our system is to assign clients to backup servers without overloading any individual server, either with too much data being stored or too much data being written during a backup window. We define a *cost metric* to provide a single utility value for a given configuration. The cost has several components, representing skew, overload, and movement, as shown in Table 1.

The basic cost represents the skew across storage and throughput utilizations of the various servers, and when the system is not heavily loaded it is the dominant component of the total cost metric. Under load, the cost goes up dramatically. Exceeding capacity is considered fatal, in that it is not a transient condition and cannot be recovered from without allocating new hardware or deleting data. Exceeding throughput is not as bad as exceeding capacity, as long as there is no “hard deadline” by which the backups must complete — in that event, data will not be backed up. Even if not exceeded, the closer capacity or throughput is to the maximum allowable, the higher the “cost” of that configuration. In contrast, having a significantly lower capacity utilization than is allowable may be good, but being 50% full is not “twice as good” as being 100% full. As a result, the cost is nonlinear, with dramatic increases close to the maximum allowed and jumps to extremely high costs when exceeding the maximum allowed. Finally, there are costs to reassigning clients to new servers. We cover each in turn.

Variable	Description	Scope	Typical values
C_{basic}	weighted sum of skews	system-wide	0-2
C_{fit}	fit penalty	overloaded server	1000's
C_{util}	sum of storage and throughput utilization metrics	server	0-1000's
$C_{movement}$	sum of movement penalties	server	0-10's
D_S, D_T	standard deviation (skew) of {storage, throughput} utilizations	system-wide	0-1
$U_{i,s}$	storage utilization of node i	server	0-1
$U_{i,t}$	throughput utilization of node i	server	0-1
α	weight for storage skew relative to throughput		0.8
m	number of servers		

Table 1: Components of the cost function and related variables.

Skew

The basic cost starts with a weighted sum of the standard deviations of the capacity and throughput utilizations of the storage appliances:

$$C_{basic} = \alpha D_S + (1 - \alpha) D_T,$$

where α is a configurable weight (defaulting to 0.8), D_S is the standard deviation of storage utilizations $U_{i,s}$ (the storage utilization of node i is between 0 and 1, or above 1 if node i is overloaded), and D_T is the standard deviation of throughput utilizations $U_{i,t}$ (similar definition and range). The notion is that if predicted utilization is completely equal, there is no benefit to adjusting assignments and increasing that skew; however, one might redefine this metric to exclude one or more systems explicitly targeted to have excess capacity for future growth. Since throughput is more dynamic than capacity, the default

weights emphasize capacity skew (80%) over throughput skew (20%).

Overflowing Clients

There are then some add-ons to the cost to account for penalties. First and foremost, if a server would be unable to fit all the clients assigned to it, there is a penalty for each client that does not fit:

$$C_{fit} = \text{fit_penalty_factor} \sum_{i=1}^m F_i,$$

F_i is the number of clients not fitting on node i , and the penalty factor used in our experiments is a value of 1000 per excess host. We use 1000 as a means of ensuring a step function: even if one out of many servers is just minimally beyond its storage capacity (i.e., a utilization of 1.00...01) the cost will jump to 1000+. In addition, when several clients do not fit, the contribution to total cost from the fit penalty is in the same range as the contribution from the utilization (see below).

To count excess clients, we choose to remove from smallest to largest until capacity is not exceeded: this ensures that the greatest amount of storage is still allocated, but it does have the property that we could penalize many small clients rather than removing a single large one. We consider the alternate approach, removing the largest client first, in §7.2.

Utilization

There are also level-based costs. There are two thresholds, an *upper* threshold above (100%), a clearly unacceptable state, and a *lower* threshold (80% of the maximum capacity or throughput) that indicates a warning zone. The costs are marginal, similar to the U.S. tax system, with a very low cost for values below the lower threshold, a moderate cost for values between the lower and upper thresholds, and a high cost for values above the upper threshold, the overload region. Since the costs are marginal, the penalty for a value just above a threshold is only somewhat higher than a value just below that threshold, but then the increase in the penalty grows more quickly with higher values.

The equation for computing the utilization cost of a configuration is as follows. Constant scaling factors ranging from 10–10000 are used to separate the regions of bad configurations: all are bad, but some are worse than others and get a correspondingly higher penalty. The weight of 0.1 for the more lightly loaded utilization makes adjustments in the range of the other penalties such as utilization skew. Each range of values inherits from the lower ranges; for example, if $U_{i,s} > 1$ then its penalty is 10,000 for everything above the threshold of

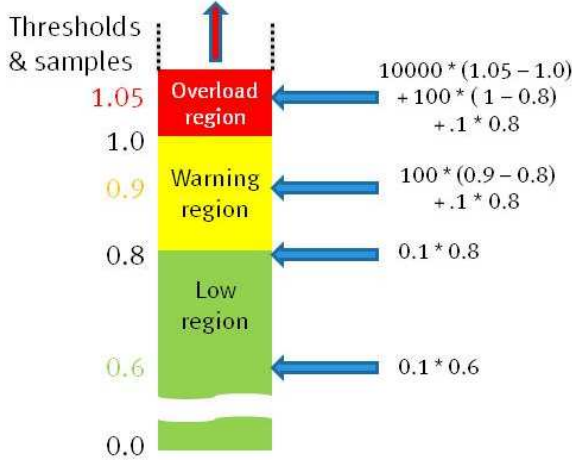


Figure 2: The cost associated with storage on a node, S_i , depends on whether the utilization falls into the *low* region, the *warning* region, or the **overload** region. The costs are cumulative from one region to a higher one.

1, added to the penalty for values between 0.8 and 1 ($100 * .2$, the size of that region) and the penalty for values between 0 and 0.8 ($.1 * .8$, the size of *that* region). Figure 2 provides an example with several possible utilization values within and between the regions of interest.

$$C_{util} = \sum_{i=1}^m S_i + T_i$$

$$S_i = \begin{cases} .1 * U_{i,s}, & U_{i,s} < .8 \\ .1 * .8 + 100 * (U_{i,s} - .8), & .8 < U_{i,s} \leq 1 \\ .1 * .8 + 100 * .2 + 10000 * (U_{i,s} - 1), & U_{i,s} > 1 \end{cases}$$

$$T_i = \begin{cases} 0, & U_{i,t} < .8 \\ 10 * (U_{i,t} - .8), & .8 < U_{i,t} \leq 1 \\ 10 * .2 + 1000 * (U_{i,t} - 1), & U_{i,t} > 1 \end{cases}$$

The highest penalty is for being $> 100\%$ storage capacity, followed by being $> 100\%$ throughput. If an appliance is above the lower threshold for capacity or throughput, a lesser penalty is assessed. If it is below the lower threshold, no penalty is assessed for throughput, and a small cost is applied for capacity to reflect the benefit of additional free space. (Generally, a decrease on one appliance is accompanied by an increase on another and these costs balance out across configurations, but content overlap can cause unequal changes.) These penalties are weights that vary by one or more orders of magnitude, with the effect that any time one or more storage appliances is overloaded, the penalty for that overload dominates the less important factors. Only if no appliance has capacity or throughput utilization significantly over the lower threshold do the other penalties

such as skew, data movement, and small differences in utilization, come into play.

Within a given cost region, variations in load still provide an ordering: for instance, if a server is at 110% of its capacity and a change in assignments brings it to 105%, it is still severely loaded but the cost metric is reduced. As a result, that change to the configuration might be accepted and further improved upon to bring utilization below 100% and, hopefully, below 80%. Dropping capacity below 100% and avoiding the per-client penalties for the clients that cannot be satisfied is a big win; this could result in shifting a single large client to an overloaded server in order to fit many smaller ones. Conversely, the reason for the high penalty for each client that does not fit is to ensure that the cost encompasses not only the magnitude of the capacity gap but also the number of clients affected, but there is a strong correlation between C_{fit} and C_{util} in cases of high overload.

Movement

The final cost is for data movement: if a client was previously assigned to one system and moves to another, a penalty is assessed in proportion to that client's share of the original system's capacity. This penalty is weighted by a configurable "movement penalty factor." Thus, a client with 1TB of post-dedupe storage, moving from a 30-TB server, would add $movement_penalty_factor * \frac{1}{30}$ to the configuration cost.

$$M_i = \sum_{clients_i} movement_penalty_factor * \frac{size_{client}}{size_i}$$

$$C_{movement} = \sum_{i=1}^m M_i$$

Movement_penalty_factor defaults to 1, which also results in the adjustment to the cost being in the same range as skew, though the *movement_penalty_factor* could be higher in a WAN situation. We discuss other values below.

In total, the cost C for a given configuration is:

$$C = C_{basic} + C_{fit} + C_{util} + C_{movement}$$

The most important consideration in evaluating a cost is whether it indicates overload or not; among those with low enough load, any configuration is probably acceptable. In particular, penalties for movement are inherently lower than penalties for overload conditions, and then among the non-overloaded configurations, any with movement is probably worse than any that avoids such movement. Thus the weight for the movement penalty, if at least 1 and not orders of magnitude higher, has little effect on the configuration selected.

4.2 Algorithmic Approaches

We considered four methods of assigning clients; three are fairly simple and efficient but tend to work badly in overloaded environments, while the fourth is much more computationally expensive but can have significant benefits. In all cases, if a configuration includes predetermined assignments, those assignments are made first, and then these methods are used to assign the remaining clients. Existing assignments can constrain the possible assignments in a way that makes imbalance and overload likely, if not unavoidable.

The three “simple” algorithms are as follows. None of them takes content overlap into account in *selecting* a server for a particular client. However, they do a limited form of accounting for content overlap once the server is assigned. The next section discusses extensive analysis to compute *pair-wise* overlaps between specific hosts, but computing the effects of the pair-wise overlaps as each client is assigned is expensive. The simple algorithms instead consider only *class-wise* overlaps, in which the presence of another client in the same “class” as the client most recently added is assumed to provide a fixed reduction to the newer client’s space requirements. That reduction is applied to that client before continuing, so servers with many overlapping clients can be seen to have additional capacity. The final, more precise, cost calculation is performed after all assignments are completed.

Random (RAND) Randomly assign clients to backup servers. RAND picks from all servers that have available capacity. If a client does not fit on the selected server, it then checks each server sequentially. If it wraps around to the original selection and the client does not fit, the client is assigned to the first choice, whose utilization will now be > 1 , and the cost metric will reflect both the high utilization and the overflowing client. By default, we run RAND 10 times and take the best result, which dramatically improves the outcome compared to a single run [14].

Round-robin (RR) Assign clients to servers in order, regardless of the size of the client. Again, if a server does not have sufficient capacity, the next server in order will be tried; if no server is sufficient, the first one will be used and an overflowing client will be recorded.

Bin-packing (BP) Assign based on capacity, in decreasing order of required capacity, to the server with the most available space. If no server has sufficient capacity, the one with the most remaining capacity (or the least overflow) will be selected and the overflowing client will be recorded.

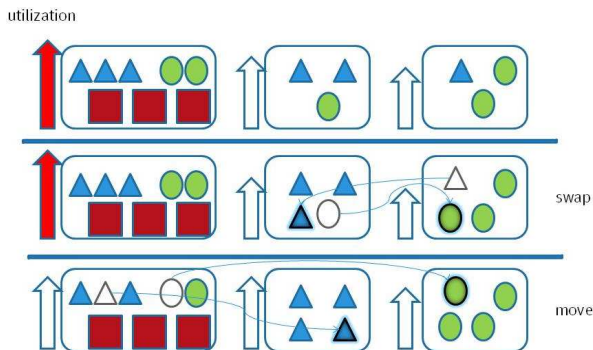


Figure 3: With simulated annealing, the system tries swapping or moving individual clients to improve the overall system cost. Here, the different shapes are assumed to deduplicate well against each other, so swapping a circle with a triangle reduces the load of both machines. Then moving a circle and a triangle from the overloaded server on the left onto the other systems increases their loads but decreases the leftmost server’s load. The arrows represent storage utilization, with the red ones highlighting overload. The dark borders and unshaded shapes represent new or removed assignments, respectively.

The fourth algorithm bears additional detail. It is the only one that dynamically reassigns previously assigned clients, trading a movement penalty for the possible benefit of lowered costs in other respects. It does a full cost calculation for each possible assignment, and does many possible assignments, so it is computationally expensive by comparison to the three previous approaches.

Simulated annealing (SA) [11] Starting with the result from BP, perturb the assignments attempting to lower the cost. At each step, a small number of servers are selected, and clients are either *swapped* between two servers or *moved* from one to another (see Figure 3). The probability of movement is higher initially, and over time it becomes more likely to swap clients as a way of reducing the impact. The cost of the new configuration is computed and compared with the cost of the existing configuration; the system moves to the new configuration if it lowers the cost or, with some smaller probability, if the cost does not increase dramatically. The configuration with the lowest cost is always remembered, even if the cost is temporarily increased, and used at the end of the process.

We use a modified version of the Perl MachineLearning::IntegerAnnealing library,¹

¹This library appears to have been superseded by the AI::SimulatedAnnealing library, <http://search.cpan.org/~bfitch/AI-SimulatedAnnealing-1.02/>.

which allows some control over the way in which the assignments are perturbed:

- The algorithm accepts a set of initial assignments, rather than starting with random assignment.
- It accepts a specification of the percent of assignments to change in a given “trial,” when it tries to see if a change results in a better outcome. This percentage, which defaults to 10%, decreases over time.
- The probability of moving a client from one storage appliance to another or swapping it with a client currently assigned to the other appliance is configurable. It starts at $\frac{2}{3}$ and declines over time.
- The choice of the target systems for which to modify assignments can be provided externally. This allows it to focus on targets that are overloaded rather than moving assignments among equally underloaded systems.

By default, SA is the only algorithm that reassigns a client that has already been mapped to a specific storage appliance (we consider a simple alternative to this for the other algorithms in §7.1).

We evaluate the effectiveness of these algorithms in §6.3. In general, RAND and RR work “well enough” if the storage appliances are well provisioned relative to the client workload and the assignments are made on an empty system. However, if we target having each system around 80–90% storage utilization or adjust a system that was overloaded prior to adding capacity, these approaches may result in high skew and potential overload. BP works well in many of the cases, and SA further improves upon BP to a limited extent in a number of cases and to a great extent in a few extreme examples. SA has the greatest benefit when the system is overloaded, especially if the benefits of content overlap are significant, but in some cases it is putting lipstick on a pig: it lowers the cost metric, but the cost is still so high that the difference is not meaningful. Naturally, the solution in such cases is to add capacity.

5 Computing Overlap

There are a number of ways by which one can determine the overlap of content on individual systems. In each case we start with a set of “fingerprints” representing individual elements of deduplication, such as chunks. These fingerprints need not be as large as one would use for actual deduplication. (For instance, a 12-byte fingerprint with a collective false positive rate of $\frac{1}{2^{32}}$ is fine for

estimating overlap even if it would be terrible for actually matching chunks – for that one might use 20 bytes or more, with a false positive rate of $\frac{1}{2^{96}}$.) The fingerprints can be collected by reading and chunking the file system, or by looking at existing backups that have already been chunked.

Given fingerprints for each system, we considered two basic approaches to computing overlap: **sort-merge** and **Bloom filters** [1].

With sort-merge, the fingerprints for each system are sorted, then the minimal fingerprint across all systems is determined. That fingerprint is compared to the minimal fingerprint of all the systems, and a counter is incremented for any systems that share that fingerprint, such that the pair-wise overlap of all pairs of systems is calculated. After that fingerprint is removed from the ordered sets containing it, the process repeats.

With Bloom filters, the systems are processed sequentially. Fingerprints for the first system are inserted into its Bloom Filter. Then for each subsequent system, fingerprints are added to a new Bloom filter, one per system. When these fingerprints are new to that system, they are checked against each of the previous systems, but not added to them.

The sort-merge process can be precise, if all fingerprints are compared. Bloom filters have an inherent error rate, due to false positives when different insertions have collectively set all the bits checked by a later data element. However, that false positive rate can be fairly low (say 0.001%), depending on the size of the Bloom filter and the number of functions used to hash the data.

If the Bloom filters are all sufficiently sparse after all insertions have taken place, another way to estimate overlap is to count the number of intersecting bits that have been set in the bit-vector; however, for “standard-size” Bloom filters setting multiple bits per element inserted, we found it is easy to have a 1% overlap of fingerprints result in 20–30% overlap in bits. Each filter would need to be scaled to be significantly larger than would normally be required for a given number of elements, which would in turn put more demands on system memory, or the number of bits set for each entry would have to be reduced, increasing the rate of false positives. (Consistent with this result, Jain, et al. [10], reported a detailed analysis of the false positive rate of intersecting Bloom filters, finding that it is very accurate when there is high overlap but remarkably misleading in cases of little or no overlap. Since we expect many systems to overlap by 0–20% rather than 75–100%, Bloom filter intersection would not be helpful here.)

Regardless of which approach is used, there is an additional concern with respect to clustering more than two clients together. Our goal is to identify what fraction of a new client A already exists on a system containing data

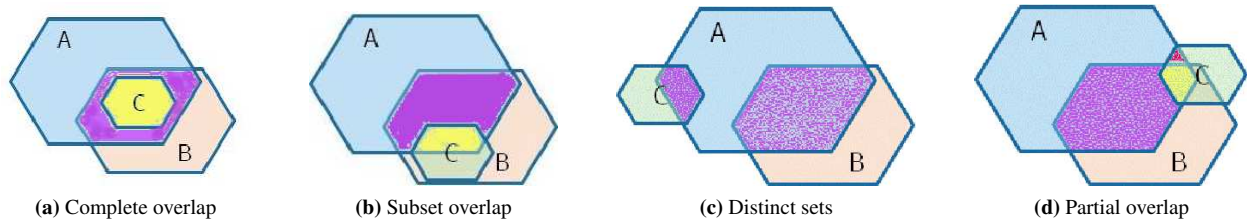


Figure 4: Four views of possible overlap among A , B , and C . The red or magenta areas indicate overlap that can be attributed to a single pair. The yellow area indicates overlap that must be attributed to multiple intersecting datasets.

from clients $B, C, \dots Z$. This is equivalent to taking the intersection of A 's content with the *union* of the content of the clients already present:

$$Dup(A) = A \cap (B \cup C \cup \dots \cup Z)$$

However, we cannot store the contents of every client and recompute the union and intersection on the fly. To get an accurate estimate of the intersection, we ideally want to precompute and store enough information to estimate this value for all combinations of clients. If we only compute the number of chunks in common between A and B , A and C , and B and C , then we don't know how many are shared by all of A , B , and C . For example, if $A \cap B = 100$, $A \cap C = 100$, and $B \cap C = 100$, $A \cap B \cap C$ may be 100 as well, or it may be 0. If A and B are already assigned to a server and then C is added to it, C may have as little as 100 in common with the existing server or it may have as many as 200 overlapping. The value of $A \cap B \cap C$ provides that quantity.

Figure 4 depicts some simple scenarios in a three-client example. In the first two cases, $C \subset B$, so even though C overlaps with A the entire overlap can be computed by looking at A and B . In the third case, B and C are completely distinct, and so if A joined a storage appliance with B and C the content in $A \cap B$ and $A \cap C$ would all be duplicates and the new data would consist of the size of A minus the sizes of $A \cap B$ and $A \cap C$. The last case shows the more complicated scenario in which B and C partially intersect, and each intersects A . Here, the yellow region highlights an area where A intersects both B and C , so subtracting $A \cap B$ and $A \cap C$ from A 's size would overestimate the benefits of deduplication. The size of the region $A \cap B \cap C$ must be counted only once.

Therefore, the initial counts are stored for the largest group of clients. By counting the number of chunks in common among a set S of clients, we can enumerate the $2^{|S|}$ subsets and add the same number of matches to each subset. Then, for each client C , we can compute the fraction of its chunks that are shared with any set of one or more other clients; this similarity metric then guides the assignment of clients to servers.

To keep the overhead of the subset enumeration from being unreasonable, we cap the maximum value of $|S|$. Fingerprints that belong to $> S_{max}$ clients are shared widely enough not to be interesting from the perspective of content-aware assignment, for a couple of reasons: first, if more clients share content than would be placed on a single storage appliance, the cluster will be broken up regardless of overlap; and second, the more clients sharing content, the greater the odds that the content will exist on many storage appliances regardless of content-aware assignment. Empirically, a good value of S_{max} is in the range $[\frac{S}{3}, \frac{S}{2}]$.

In summary, for each client, we can compute the following information:

- What fraction of its chunks are completely unique to that client, and will not deduplicate against any other client? This value places an upper bound on possible deduplication.
- What fraction of its chunks are shared with at least $S_{max} - 1$ clients? We assume these chunks will deduplicate on any appliance that already stores other clients, providing an approximate lower bound on deduplication, but there is an inherent error from such an assumption: if the $S_{max} - 1$ clients are all on a single appliance, the S^{th} client will only get the additional deduplication if it is co-resident with these others.
- How much does the client deduplicate against each other client, excluding the common chunks?

Combining the per-pair overlaps with per-triple data, we can identify the best-case client with which to pair a given client for maximum deduplication, then the best-case second client that provides the most *additional* deduplication beyond the first matching client. §6.2 describes the results of this analysis on a set of 21 Linux systems. Since even the 3rd client is usually a marginal improvement beyond the 2nd, we do not use overlap beyond pairwise intersections in our experiments.

5.1 Approximation Techniques

Dealing with millions of fingerprints, or more, is unwieldy. In practice, as long as the fingerprints are uniformly distributed, it is possible to estimate overlap by sampling a subset of fingerprints. This sampling is similar to the approach taken by Dong, et al., when routing groups of chunks based on overlapping content [4], except that the number of chunks in a group was limited to a relatively small number (200 or so). Thus in that work, the quality of the match degraded when sampling fewer than $\frac{1}{8}$ fingerprints, but when characterizing entire multi-GB or multi-TB datasets, we have many more fingerprints to choose from. Empirically, sampling 1 in 1024 fingerprints has proven to be about as effective as using all of them; we discuss this further in §6.2.1.

In addition, it is possible to approximate the effect of larger clusters by pruning the counts of matches whenever the number is small enough. For instance, if $A \cap B$ is 10% of A and 5% of B , $A \cap C$ is 15% of A and 5% of C , and $A \cap B \cap C$ is 0.5% of A , then we estimate from $A \cap B$ and $A \cap C$ that adding A to B and C will duplicate 25% of A 's content. This overestimates the duplication by 0.5% of A since it counts that amount twice, but the adjustment is small enough not to affect the outcome. Similarly, in Figure 4d, the yellow region of overlap $A \cap B \cap C$ is much greater than the intersection only between A and C that does not include B : adding A to B and C is approximately the same as adding A to B alone, and C can be ignored if it is co-resident with B .

This approximation does not alleviate the need to compute the overlap in the first place, since it is necessary to do the comparisons in order to determine when overlap is negligible. But the state to track each individual combination of hosts adds up; therefore, it is helpful to compute the full set, then winnow it down to the significant overlaps before evaluating the best way to cluster the hosts. This filter can be applied all the way at the level of pairs of clients, ignoring pairs that have less than some threshold (such as 5%) of the content of at least one client in common.

6 Evaluation

In this section we describe the use of the client assignment tool in real-world and simulated environments. §6.1 discusses the datasets used, §6.2 reports some examples of overlapping content and the impact of sampling the dataset fingerprints, and §6.3 compares the various algorithms introduced in §4.2.

6.1 Datasets

To evaluate our approach, we draw from three datasets:

1. *Linux workstations, full content.* We have a set of 21 collections of fingerprints of content-defined chunks on individual Linux workstations and file servers. Most of these are drawn from a canonical internal test dataset² from 2005-6 containing full and incremental backups of workstations over a period of months; since duplicate fingerprints are ignored, this is the union of all content on these systems over that period (excluding any data that never makes it to a backup). About $\frac{1}{4}$ are from a set of workstations and file servers currently in the Princeton EMC office, collected in 2011 through a single pass over each local file system.
2. *Artificial dataset, no content.* In order to show the effect of repeatedly adding clients to a system over time, we generated an artificial dataset with a mix of three client sizes. Each iteration, the system adds 20 clients: 10 small clients with full backups of 20GB, 7 medium 100GB clients, and 3 big 2TB clients. This adds up to 6.9TB of total full backups, which scales to about 8TB of unique data to be retained over a period of several months. We simulate writing the datasets onto a number of DD690 backup systems with 35.3TB storage each; after deduplication, about 5 sets of clients (100 clients in total) can fit on one such appliance. We start with 2 servers and then periodically add capacity: the goal is to go from comfortable capacity to overload, then repeatedly add a server and add more clients until overloaded once again. This can be viewed as an outer loop, in which DD690 appliances are added, and an inner loop, in which 20 clients are assigned per iteration. Once assigned to a server, a client starts with a preference for that server, except for when a new backup server is added: to give the non-migrating algorithms a chance to rebalance, the previous assignments are forgotten with $\frac{1}{3}$ probability.

We consider two types of overlap, one in which there is a small set of clients with high overlap, and one in which all clients of a “class” have small overlap. In the former case, each client added during an iteration of the outer loop deduplicates 30% of its content with the corresponding clients from previous iterations of the outer loop: the i^{th} client added when there were 6 DD690s dedupes well with the i^{th} client added when there were [2..5] DD690s present. It deduplicates 10% with all other clients of the same type (big, medium, or small). In the latter case, only the 10% per-class overlap applies.

²This is the “workstations” dataset in a previous paper [4].

ID	Best Possible (%)	Best2 (%) =	Widely Shared (%) +	Saved1 (%) +	Add'l Saved (%)	Chunks	Unique Chunks	Match1 host	Match2 host	Pct Saved2 (in isolation)
host1	73.87	73.75	0.77	48.9	24.08	823,256	215,083	host21	host16	30.9
host2	32.06	31.53	0.53	27.8	3.19	9,065,414	6,158,755	host16	host20	3.6
host3	18.68	17.21	0.80	14.9	1.51	3,843,577	3,125,766	host4	host20	12.1
host4	15.03	14.53	0.84	13.2	0.50	4,852,119	4,122,931	host5	host20	10.2
host5	14.34	13.04	0.74	10.5	1.80	6,645,378	5,692,506	host9	host20	7.8
host6	13.00	12.43	2.82	8.9	0.71	7,853,942	6,832,555	host9	host11	1.1
host7	11.67	10.19	5.94	3.3	0.95	3,460,930	3,057,042	host11	host16	2.4
host8	10.88	10.7	8.14	2.5	0.06	2,458,516	2,191,010	host19	host13	1.2
host9	10.3	8.05	0.44	5.7	1.91	31,410,032	28,176,318	host16	host5	2.2
host10	9.41	8.91	5.32	2.7	0.89	4,195,226	3,800,335	host13	host18	2.2
host11	8.73	6.16	2.81	1.9	1.46	8,066,949	7,362,355	host9	host13	1.6
host12	8.36	7.38	4.67	1.6	1.11	4,512,393	4,135,327	host6	host13	1.4
host13	7.89	6.47	3.61	2.1	0.76	6,231,280	5,739,526	host11	host18	1.3
host14	7.88	7.28	5.07	1.9	0.31	4,361,658	4,018,166	host19	host11	1.6
host15	7.70	7.08	4.34	2.4	0.34	5,141,613	4,745,660	host11	host13	1.0
host16	7.36	6.28	0.10	3.9	2.28	64,735,211	59,973,773	host2	host9	2.7
host17	7.17	6.52	2.27	3.3	0.96	3,035,582	2,817,910	host16	host5	1.2
host18	6.27	5.55	2.44	2.2	0.91	9,220,185	8,641,937	host16	host11	1.6
host19	4.73	3.99	2.39	1.2	0.40	9,359,512	8,917,158	host11	host5	0.4
host20	3.07	2.33	0.15	1.8	0.38	28,381,188	27,508,835	host5	host2	1.1
host21	1.77	1.70	0.03	0.9	0.77	43,045,905	42,284,086	host1	host16	0.9
Average	8.13					260,699,866	239,517,034			

Table 2: Inter-host deduplication of 21 workstations

3. *Customer backup metadata, no content.* We have a collection of 480 logs of customer backup metadata, including such data as the size of each full or incremental backup, the duration of each backup, the retention period, and so on. These logs do not include actual content, though they include the “class” that each machine being backed up is in: one can infer better overlap between machines in the same class than machines in different classes, but not quantify the extent of the overlap. (We assume a 10% overlap for clients in the same class.) We preprocess these logs to estimate the requirements for each client within a given customer environment, compute the size and number of backup storage appliances necessary to support these clients, then assign the clients to this set of storage appliances. By adjusting the desired threshold of excess capacity, we can vary the contention for storage capacity and I/O. In this paper we consider only the largest of these customer logs, with nearly 3,000 clients.

6.2 Content Overlap

Table 2 and Figure 5 describe the intersection of the 21 Linux datasets. Hosts are anonymized via the names “host1” to “host21,” shown in the first column of the table. The next column shows the idealized deduplica-

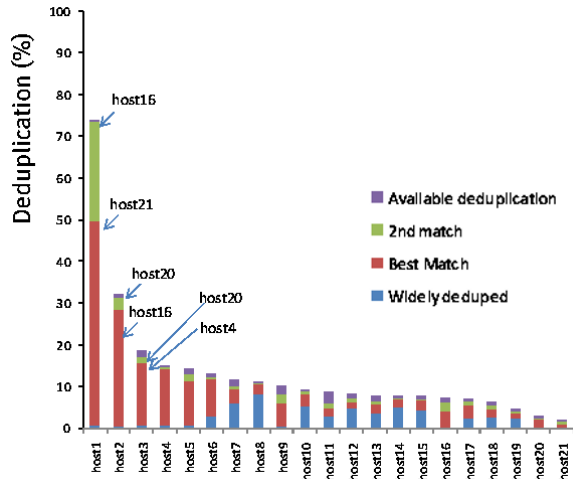


Figure 5: This is a visual depiction of the data in Table 2, showing the components contributions to the deduplication of each host.

tion, computed by dividing the number of unique chunks by the number of chunks and subtracting the result from 100%. (We assume that all chunks are the same size, although in practice they are statistically an *average* of

8 Kbytes.) Hosts are sorted by the best possible deduplication rate. The average across all hosts is about 8%.

The next column, **Best2**, reports the deduplication obtained by matching a given host against the best two hosts, as described in §5. It is the sum of the widely shared data appearing on that host (typically around 1–2% but as high as 8%), the additional deduplication specifically against the **Match1 host**, and the additional deduplication against **Match2 host**. Since the second match excludes both common chunks and anything on **Match1**, the added benefit from the second host is usually under 2%, but in the case of `host1`, the second matching host provides about half as much deduplication as the first host, over 24%. **Pct Saved2** indicates how much deduplication could have been achieved by the second host without the first.

The columns listing which hosts provided the best and second-best deduplication indicate that a handful of hosts provide most of the matches. Also, the relationships are not always symmetric, in part because of varying dataset sizes. `Host2` is the best match for `Host16` and vice-versa, but in other cases it is more of a directed graph.

Figure 5 shows this data visually. The height of each bar corresponds to the best possible deduplication, The blue bar at the bottom is the percent of chunks on that host that appear on many other hosts, the red bar shows the additional benefit from the best single match, the green bar shows the additional benefit from a second host, and the purple bar shows extra deduplication that might be obtained through three or more co-resident hosts. Not all bars are visible for each host. For the first three hosts, arrows identify the matching hosts shown in the table. A host with relatively little data may deduplicate well against a larger host, while the larger host gets relatively little benefit from deduplicating in turn against the smaller one; in this case the host with the best overall deduplication matches the host with the poorest deduplication, as a fraction of its total data.

Lest there be a concern that there is a small number of examples reflecting good deduplication, while the average is relatively low, there are other meaningful datasets with substantial overlap. For example, two VMDK files representing different Windows VMware virtual machine images used by an EMC release engineering group overlapped by 49% of the chunks in each.

6.2.1 Sampling

Our goal for sampling is to ensure that even with approximation, the system will find the same “best match” as with perfect data (a.k.a. the “ground truth”), or at least a close approximation to it. We use the following criteria:

- If a host H had a significant match with at least one other host H_1 of 5% of its data, above and beyond

the “widely shared” fingerprints, we want the approximated best match to be close to the ground truth. We define “close” as a window β around the correct value, which is within either 5%, 10%, or 20% of the value, with a minimum of 1%. For example, if the ground truth is 50%, acceptable $\beta = 5\%$ approximations would be 47.5–52.5%, but if the ground truth is 5%, values from 4–6% would be acceptable. Note that if the estimated match were outside that range but H_1 was believed to be the best match, we might cluster the two together but misestimate the benefit of the overlap.

- If the best match found via approximation is with another host H_2 , rather than the ground truth best match, it may still be acceptable. The approximate overlap needs to be close to the actual overlap of H_2 , or we would misestimate the benefits, but we only would find the alternate host H_2 acceptable if it was within β of the value of H_1 . Thus the approximate match $H_{2,approx}$ must be $> (1 - \beta)H_1$ and $< (1 + \beta H_2)$.
- If the host had *no* significant match ($> 5\%$) with another single host, we want the approximation to reflect that. But again, a small change is acceptable. For example, if the best match were 4.5% and we would have ignored it, but the approximation reports that the best match is 5.5%, that is a reasonable variance. If the best match was 1% and is now reported as 5.5%, that would be a significant error.

The ranges of overlap are important because in practice a high relative error is inconsequential if the extent of the match is limited to begin with. If we believe two clients match in only 0.5% of their data, we are unlikely to do much differently if we estimate this match is 1% or 2%, or if we believe there is no match at all. On the other hand, if we think that a 50% match is only 25% or is closer to 100%, the assignment tool might make a bad choice. Even if it picks the right location due to the overlap, it will underestimate or overestimate the impact on available capacity.

Figure 6 depicts the effect of sampling fingerprints, using the same 21-client fingerprint collection. The x-axis depicts the sampling rate, with the left-most point corresponding to the ground truth of analyzing all fingerprints. As the graph moves to the right, the sampling rate is reduced. There are three curves, corresponding to margins of error $\beta = 5\%$, $\beta = 10\%$, and $\beta = 20\%$. The y-axis shows the fraction of clients with an error outside the specified β range. For a moderate margin of error there is little or no error until the sampling rate is lower than $\frac{1}{1024}$, though if one desires a tighter bound on β , the error rate increases quickly.

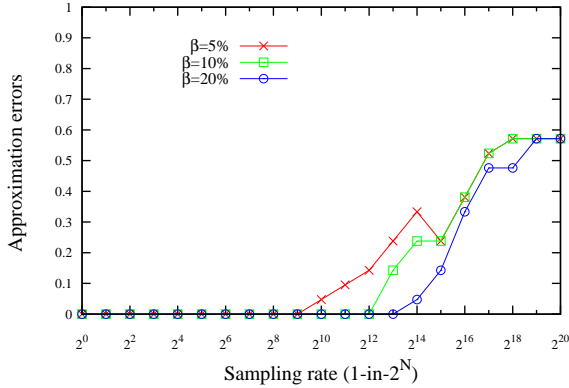


Figure 6: For a given sampling rate, on the x-axis, of $\frac{1}{2^N}$, we compute what fraction of clients have their biggest overlap approximated within an given error threshold β .

6.3 Algorithm Comparison

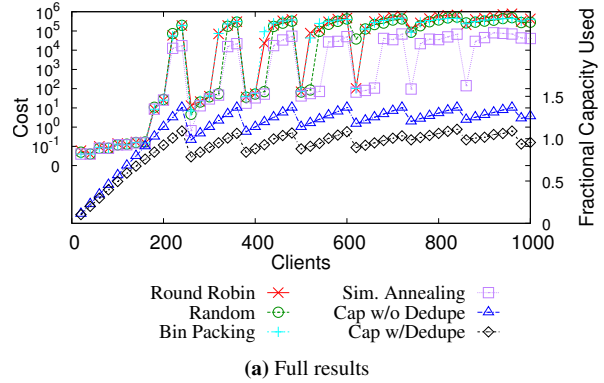
In general, any of the algorithms described in §4.2 work well if the system is not significantly loaded. As capacity or throughput limits are reached, however, the system can accommodate the greatest workloads through intelligent resource allocation. This is especially true if there is significant overlap among specific small subsets of clients.

In our analysis here, we focus on capacity limitations rather than throughput. This is because backup storage appliances are generally scaled to match throughput and capacity, so it is rare to experience throughput bottlenecks without also experiencing capacity shortages. Since it can occur with high-turnover data (a good deal of data being written but then quickly deleted), the cost function does try to optimize for throughput as well as capacity.

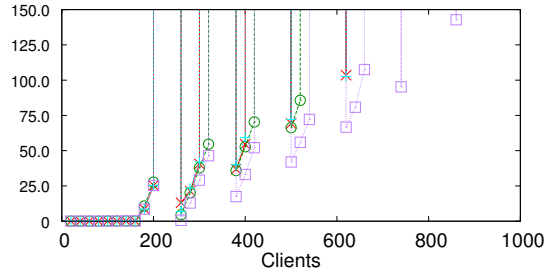
Incremental Assignment

We first compare the four algorithms as clients and backup storage are repeatedly added, using the artificial dataset described in §6.1 and new servers every 120 clients.

Figure 7 shows the results of this process with the number of clients increasing across the horizontal axis and cost shown on the left vertical axis. (Part (a) shows the full range of cost values on a log scale, while (b) zooms in on the values below 150, on a standard scale, to enable one to discern the smaller differences.) The two capacity curves in 7(a) reflect the ratio of the estimated capacity requirements to the available backup storage, with or without considering the effects of the best-case deduplication, and are plotted against the right axis. A value over 1 even with deduplication would indicate a



(a) Full results



(b) Zoom of results

Figure 7: An artificial, homogeneous client population is added 20 hosts at a time, with new backup storage added every 120 hosts after the first 240. A small set of clients match each other with 30% deduplication and otherwise hosts of the same type match 10% of their data. The costs are shown by the curves marked on the left axis. The capacity requirements are shown by the curves at the bottom of the top graph, marked on the right axis.

condition in which insufficient capacity is available, but any values close to or above 1 indicate potential difficulties.

In Figure 7, the general pattern is for the “simple” algorithms to fail to fit all the clients within available constraints, once the collective requirements first exceed available capacity, while SA cycles between being able to accommodate the clients and failing to do so (but still being an order of magnitude lower cost even when failing to fit them). There is a stretch between 600–700 clients in which it does particularly well; this is because in this iteration of the outer loop, the number of distinct clusters of highly overlapping clients equals the number of storage appliances, and the system balances evenly.

While the sequence depicted in Figure 7 is a case in which explicit pair-wise overlap is essential to fitting the clients in available capacity, the sequence in Figure 8 adds fewer clients per storage appliance. Clients almost always fit, though SA improves upon the other approaches some of the time. As expected, RR is not quite

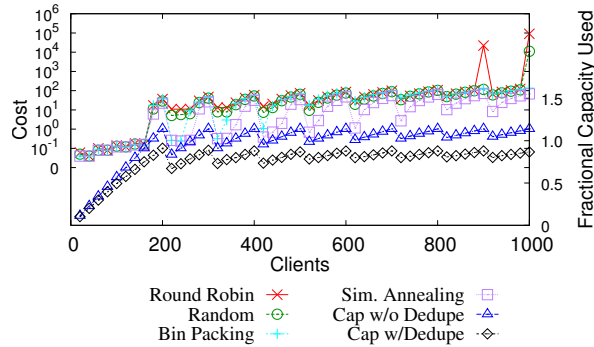


Figure 8: The same artificial, homogeneous client population is added 20 hosts at a time, with new backup storage added every 100 hosts after the first 200. The costs are shown by the higher curves, marked on the left axis. The capacity requirements are shown by the curves at the bottom of the graph, marked on the right axis.

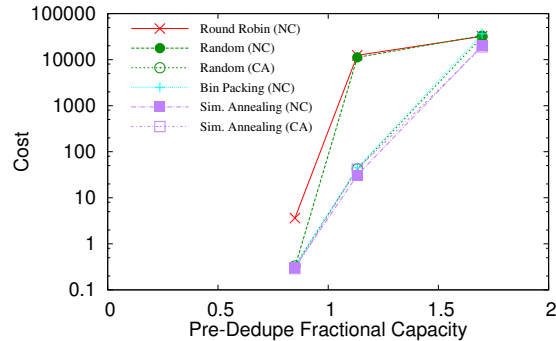
as good as BP; when the number of clients is high, there are cases where RR exceeds capacity because it considers only *whether* a client fits and not how *well* it fits, and because it is constrained by earlier assignments. RAND similarly fails when 1000 clients are present.

In summary, we find that under high load RAND, RR, and even BP fail to have acceptable costs in a large number of cases, but SA shuffles the assignments to better take advantage of deduplication and fits within available capacity when possible. While the SA results overlap the BP results in some cases, whenever there is a purple square without a matching aqua + overlaid upon it in Figure 7, SA has improved.

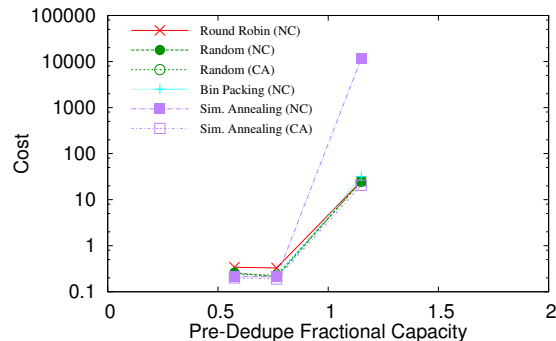
Full-Content Client Dataset

Here we describe the effect of assigning the 21-client dataset to a range of backup appliances. The overlaps of the datasets are derived from the full set of fingerprints of each client, but in the case of Host1, which is the host that is relatively small but has high overlap, we artificially increase its backup sizes by two orders of magnitude to represent a significant host rather than a trivially small one. Including this change, the clients collectively require 2.92TB before deduplication and 2.46TB or more after deduplication. They are assigned to 2–4 storage appliances with either 0.86TB (“smaller”) or 1.27TB (“larger”) capacity each.³ For the smaller servers, the clients take from about 70–140% (post-dedupe) of the available storage as the number of backup systems is

³These numbers are taken from early-generation Data Domain appliances and are selected to scale the backup capacity to the offered load. In practice, backup appliances are 1–2 orders of magnitude larger and growing.



(a) Smaller servers



(b) Larger servers

Figure 9: Cost as a function of relative capacity, pre-dedupe, for the modified 21-host dataset, for two backup appliance sizes. Algorithms are either content-aware (CA) or not content-aware (NC).

reduced, corresponding to 85–170% pre-deduplication. For the larger ones, they take 46–92% (deduplicated) or 58–115% (undeduplicated). That is, even with deduplication, at the highest utilization the clients cannot fit on only two of the smaller servers, but they fit acceptably well on the larger ones.

Figure 9 shows the cost as a function of pre-deduplication utilization. For RAND and SA, it presents two variants: one, the content-aware version, is the default; the other selects the lowest cost assuming there is no overlap, then recomputes the cost of the selected configuration with overlap considered. For BP and RR, overlap is considered only to the extent that two clients are in the same class, and the adjustment is made after a given client is assigned to a server (refer to §4.2).

Using smaller servers (9(a)), RR has a slightly higher cost under the lowest load; both RR and RAND (NC) are overloaded under moderate load, and all algorithms are overloaded under the highest load with just two servers. While it is not visible in the figure, SA without factoring content overlap into its decisions is about 6% higher cost than the normal SA which uses that information.

Using larger servers (9(b)), the costs across all algorithms are comparable in almost all cases. The notable exception is SA at the highest load: it is overloaded if it ignores content overlap, but fine otherwise. Interestingly, RAND does just as well with or without content overlap, as its best random selection without taking overlap into account proves to be a good selection once overlap is considered.

In other words, at least for this workload, there are times when it is sufficient to load balance blindly, ignoring overlap, and have overlap happen by happy coincidence. But there are times when using overlap information is essential to finding a good assignment: an approach that considers overlap can better take advantage of shared data.

Large Customer Dataset

We ran the assignment tool on the clients extracted from the largest customer backup trace, as described in §6.1. It has nearly 3,000 clients requiring about 325TB of post-deduplicated storage. Using 3 Data Domain DD880s totaling 427TB, these use about 76% of capacity, and all four algorithms assign the clients with a low cost: the maximum is 0.80 for round robin, while BP and SA are 0.24 and 0.23 respectively. It is worth noting that most of the ten RAND runs had costs over 2, but one was around 0.4 and the best was identical to the BP result. SA took over four hours and only improved it from 0.24 to 0.23.

What about overload conditions? If these were just 2 DD880s (285TB), the average storage utilization goes to 114% so no approach can accommodate all the clients. Even so, the cost metric is a whopping 184K for BP, 183K for RR, and 159K for RAND (which, by taking the lower costs of client overlap into account when comparing alternatives is able to find a slightly better assignment). These high costs are dominated by the “fit penalty” due to about 130–160 clients, out of 2983, not fitting on a server. SA, however, brought the cost down to 25K (of which 12K is from 12 clients not fitting). However, it did this by running for 5.5 cpu-days (see the next subsection).

Obviously one would not *actually* try and place 3,000 clients, totaling 325TB of post-dedupe storage, on a pair of 142TB servers. This example is intended to show how the different approaches fair under overload, and it also provides an example of a large-scale test of SA. The large number of clients to choose from poses a challenge, in that a cursory attempt to move or swap assignments may miss great opportunities, but an extensive search adds significant computation time (see the next subsection). Tuning this algorithm to adapt to varying workloads and scales and deciding the best point to prune the search are future work.

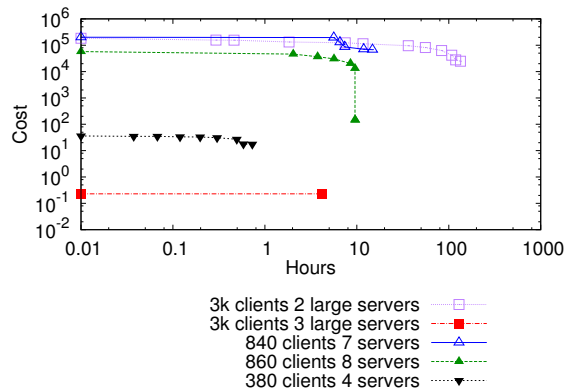


Figure 10: Cost as a function of simulated annealing analysis time for several cases. Both axes use a log scale. Except for the right-most points, any points that appear within a factor of 1.5 in both the x and y values of a point already plotted are suppressed for clarity.

6.4 Resource Usage

While our results have shown that SA can produce better assignments than the other algorithms in certain cases, there is a cost in terms of resource requirements. All three “simple” algorithms are compact and efficient. For example, the unoptimized Perl script running bin-packing on the nearly 3,000 clients and two small servers in the preceding subsection took 163M of memory and ran in 23s on a desktop linux workstation. Running SA on the same configuration took over 5 days, and the complexity of the problem is only increased when pair-wise rather than per-class overlaps are included. For the iterative problem with up to about 1,000 clients and pair-wise overlaps, the script takes several gigabytes of memory and runs for over a half day on a compute server.

Figure 10 shows timing results for five examples of earlier experiments. Two are the large-scale assignments described in the previous subsection, with nearly 3,000 clients that either fit handily or severely overload the servers. The horizontal line at the bottom represents the case where SA runs for over four hours with no effective improvement over a cost that is already extremely low. The curve toward the top with open squares is the same assignment for $\frac{2}{3}$ of the server capacity. SA dramatically reduces the cost, but it is still severely overloaded. The curve (with open triangles) near that one represents one of the incremental assignment cases in which the system is overloaded regardless of SA, while the one just below that has 20 more clients but one additional server and, in the case of SA, has a relatively low cost after a long period of annealing (the sharp drop around the 10-hour mark is an indication of SA finally succeeding in rearranging the assignments to fit capacity). Finally, the re-

maining triangle curve represents a smaller test case in which the cost starts low but SA improves it beyond what BP initially did.

In some cases (not plotted), there is a drop followed by a long tail without improvement. Ideally the process would end after a large score decrease if and only if no substantial decreases are still possible; since there is the potential to miss out on other large improvements, we let SA continue to search and hope for further decreases. Generally, with our default parameters, SA runs for seconds to hours on a desktop computer, but when configuring or updating a backup environment, that is not unreasonable, and the “best solution to date” can be used at any time. The more excess capacity there is, the easier it is for SA to hone in on a good solution quickly. For assignments of thousands of clients in an overloaded environment, some sort of “divide and conquer” approach will be necessary to keep the problem manageable.

7 Variations

In this section we discuss a couple of variations on the policies previously described: “forgetting” assignments and biasing in favor of small clients in the cost function.

7.1 Forgetting Assignments

As described to this point, whenever new clients are added to an existing set of assignments, the first assignments are “carved in stone” for the simple algorithms: they cannot be modified, and only the new unassigned clients can be mapped to any server. The SA algorithm is an exception to this, in that it can perturb existing assignments in exchange for a small movement penalty.

Here we consider a simple but extreme change to this policy: ignore all existing assignments, map the clients to servers using one of the algorithms, and pay movement penalties depending on which clients change assignments. When the assignment that takes previous assignments into account does not cause overflow, starting with a clean slate usually results in a higher cost because the movement penalties are higher than the other low costs from the “good” and “warning” operating regions. But when there would be overflow, it is often the case that rebalancing from start avoids the overflow.

Figure 11 repeats Figure 7(a), with one change: for RR, RAND, and BP, each point is the minimum between the original datapoint and a new run in which the previous assignments were ignored during assignment.⁴ Ig-

⁴Due to the high cost of SA, we do not re-run each SA experiment but instead take the minimum of the SA run and the “forgotten” BP run; that is, SA could have started from the lower BP point rather than the previous one that considered previous assignments. It might improve the cost beyond that point, something not reflected in this graph.

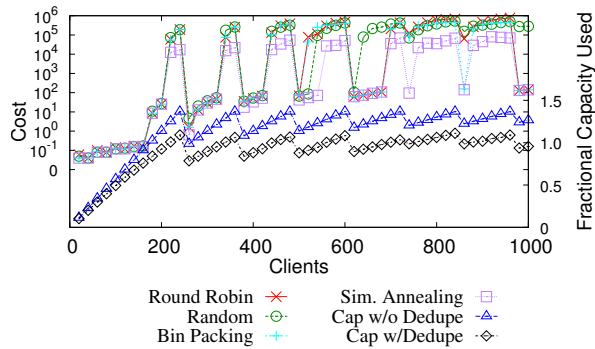


Figure 11: The same clients and servers are assigned as in Figure 7(a), but previous assignments can be ignored in exchange for a movement penalty.

noring initial assignments improved the cost metric in 35% of the cases overall, and in 43% of the cases in which the cost was over 1000 (indicating significant overload): it is frequently useful but no panacea.

The most notable difference between Figure 7(a) and Figure 11 is in the range of 600–700 clients. Previously we noted that SA does especially well in that range because of overlap, but if BP and RR start there with completely new assignments, they too have a low cost due to keeping better deduplicating clients together.

7.2 Counting Overflow

As described, the cost function biases in favor of large clients: it assumes that it is more important to back up a larger client than a smaller one, so it removes clients in order of size, smallest first, to count the number of clients that do not fit on a server. This approach is intuitive, in that a large client probably *is* more important than a small one, and it also simplifies accounting because if clients are added in decreasing order of size, we can remove a small client without affecting the deduplication of a larger one that remains.

An alternative cost function would minimize the number of occurrences of overflow by removing the largest client(s) to see if what remains will fit. This has the effect of minimizing the extra per-client penalty while still penalizing for exceeding the capacity threshold. In essence, it encourages filling $N-1$ servers to just below 100% utilization, then placing all the remaining (large) clients on the N^{th} server.

Figure 12 compares the smallest-first and biggest-first penalties for the example used in Figure 7, modified to exclude the pair-wise 30% deduplication of specific combinations of clients. (This is because recomputing the impact of removing a client against which other clients have deduplicated would require a full re-evaluation of

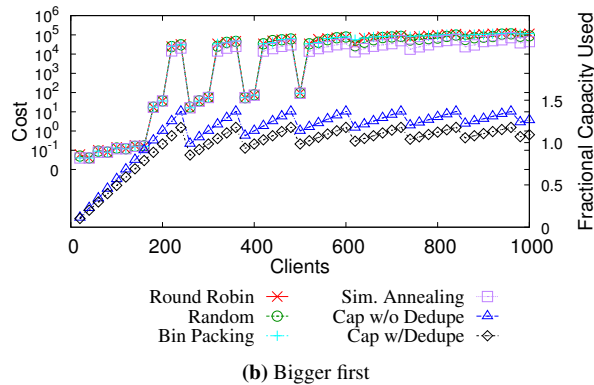
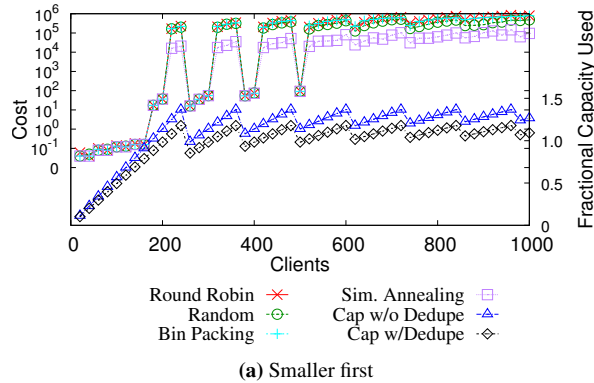


Figure 12: The same clients and servers are assigned as in Figure 7, but deduplication is only considered within a class (big, medium, or small) rather than having greater deduplication for specific pairs. C_{fit} is computed by removing the (a) smallest or (b) largest clients first.

the cost function, compared with class-wise deduplication, and has not been implemented.) The two graphs look quite similar, but because of the change to the value of C_{fit} the peak values are about one order of magnitude lower when the largest clients are counted. There is no qualitative difference in this example beyond a narrowed gap between SA and the other approaches.

8 Discussion and Future Work

Assigning backups from clients to deduplicated storage differs from historical approaches involving tape because of the stickiness of repeated content on the same server and the ability to leverage content overlap between clients to further improve deduplication. We have accounted for this overlap in a cost function that attempts to balance capacity and throughput requirements and have presented and compared several techniques for assigning clients to backup storage appliances.

When a backup system has plenty of resources for the clients, any assignment technique can work well, and

there is little difference between RAND and our most advanced technique with SA. The more interesting case is when capacity requirements reach beyond 80% of what is allocated. We have found that RAND and RR tend to degrade rapidly, while bin-packing and SA continue to maintain a low cost until capacity becomes over-subscribed. In cases of significant overlap, SA is able to use client overlap to increase the effective capacity of a set of deduplicating backup servers, deferring the point at which the system is overloaded.

There are a number of open issues we would like to address:

- evaluation of overlap in a wider range of backup workloads
- evaluation of overlap beyond the “best match” for those cases where cumulative deduplication beyond one other host is significant
- full integration between client assignment and backup software
- use of the assignment tool to manage transient bursts in load due to server failures or changes in workload
- additional evaluation of the various weights and cost function
- optimization of the SA algorithm for large-scale environments; and
- additional differentiation of clients and servers, for instance to route backups to different types of devices automatically depending on their update patterns and deduplication rates.

Efforts to integrate content affinity with pre-sales sizing are already underway.

Acknowledgments

We thank Windsor Hsu, Stephen Manley, Hugo Patterson, Hyong Shim, Grant Wallace, and Jason Warlikowski for helpful comments on the design of the system and on earlier drafts. Mike Mahler, Jason Warlikowski, Yuri Zagrebina, and Jie Zhong provided assistance with the development of the assignment tool. Thanks to Thomas Wang for backup traces and to Benjamin Fitch for the `MachineLearning::IntegerAnnealing` library. We are especially grateful to the anonymous referees and our shepherd, Doug Hughes, for their feedback and guidance.

References

- [1] Bloom, B.: Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 13(7), 422–426 (Jul 1970)
- [2] Chamness, M.: Capacity forecasting in a backup storage environment. In: *LISA'11: Proceedings of the 25th Large Installation System Administration Conference* (Dec 2011)
- [3] Chapin, S.J.: Distributed and multiprocessor scheduling. *ACM Comput. Surv.* 28 (March 1996), <http://doi.acm.org/10.1145/234313.234410>
- [4] Dong, W., et al.: Tradeoffs in scalable data routing for deduplication clusters. In: *Proceedings of the 9th USENIX conference on File and storage technologies. FAST'11, USENIX Association* (February 2011)
- [5] EMC Corporation: Data Domain Boost Software (2010), <http://www.datadomain.com/products/dd-boost.html>
- [6] EMC Corporation: Unified backup and recovery with EMC NetWorker (Feb 2010), http://www.emc.com/collateral/software/white-papers/h3399_nw_bu_rec_wp.%pdf
- [7] Gmach, D., Rolia, J., Cherkasova, L., Kemper, A.: Capacity management and demand prediction for next generation data centers. *IEEE International Conference on Web Services* (2007)
- [8] Harchol-Balter, M., Downey, A.B.: Exploiting process lifetime distributions for dynamic load balancing. *ACM Trans. Comput. Syst.* 15, 253–285 (August 1997), <http://doi.acm.org/10.1145/263326.263344>
- [9] IBM Corporation: Tivoli Storage Manager (2011), <http://www-01.ibm.com/software/tivoli/products/storage-mgr/>
- [10] Jain, N., Dahlin, M., Tewari, R.: Taper: tiered approach for eliminating redundancy in replica synchronization. In: *FAST '05: Proceedings of the 4th USENIX Conference on File and Storage Technologies*. pp. 21–21 (2005)
- [11] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* 220(4598), 671–680 (1983), <http://www.sciencemag.org/content/220/4598/671.abstract>
- [12] Meyer, D.T., Bolosky, W.J.: A study of practical deduplication. In: *Proceedings of the 9th USENIX conference on File and storage technologies. FAST'11, USENIX Association* (February 2011)
- [13] Milojicic, D.S., Douglass, F., Paindaveine, Y., Wheeler, R., Zhou, S.: Process migration. *ACM Comput. Surv.* 32, 241–299 (September 2000), <http://doi.acm.org/10.1145/367701.367728>
- [14] Mitzenmacher, M.: The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.* 12(10), 1094–1104 (2001)
- [15] Quinlan, S., Dorward, S.: Venti: a new approach to archival storage. In: *FAST '02: Proceedings of the 1st USENIX conference on File and Storage Technologies* (2002)
- [16] da Silva, J., Gudmundsson, O., Mosse, D.: Performance of a parallel network backup manager. In: *USENIX (ed.) Proceedings of the Summer 1992 USENIX Conference: June 8–12, 1992, San Antonio, Texas, USA*. pp. 217–226. *USENIX* (Summer 1992)
- [17] da Silva, J., Gumundsson, O.: The Amanda network backup manager. In: *USENIX (ed.) Proceedings of the Seventh Systems Administration Conference (LISA VII): November 1–5, 1993, Monterey, CA, USA*. pp. 171–182. *USENIX* (Nov 1993)
- [18] Soundararajan, V., Govil, K.: Challenges in building scalable virtualized datacenter management. *SIGOPS Oper. Syst. Rev.* 44, 95–102 (December 2010)
- [19] Symantec Corporation: Next generation data protection with Symantec NetBackup 7 (2011), http://eval.symantec.com/mktginfo/enterprise/white_papers/b-next_generation_data_protection_with_sym_nbu7_WP_20999878.en-us.pdf
- [20] Zhang, X., Du, D., Hughes, J., Kavuri, R.: Hptfs: A high performance tape file system. In: *Proceedings of 14th NASA Goddard/23rd IEEE conference on Mass Storage System and Technologies* (2006)
- [21] Zhu, B., Li, K., Patterson, H.: Avoiding the disk bottleneck in the Data Domain deduplication file system. In: *FAST '08: Proceedings of the 6th Conference on File and Storage Technologies*. pp. 269–282 (February 2008)