HyperProtect: A Large-scale Intelligent Backup Storage System

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Yaobin Qin

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prof. David J. Lilja

January, 2022

Acknowledgements

I would like to express my sincere gratitude to my academic advisor, Prof. David J. Lilja, for the continuous financial support and for his research and personal guidance throughout my undergraduate research opportunities program and entire Ph.D. study at the University of Minnesota. I have been in Prof. David J. Lilja's research group since I was a senior undergraduate student. As my advisor, he is always patient enough to hear all kinds of my research ideas and willing to help me grow my research ability. He has taught me more than I could ever give him credit for here. A learning that I gained from him and applied into my research is that the research idea we come up with should intuitively make sense first.

I also would like to thank Prof. Pen-Chung Yew, Prof. John Sartori, and Prof. Changhyun Choi at the University of Minnesota, for their support as members of my Ph.D. committee and for their invaluable comments and suggestions.

As a member of the Center for Research in Intelligent Storage (CRIS), I would like to thank all the industry project mentors, Aaron Christensen, Brandon Hoffmann, Weibao Wu, Xianbo Zhang, and Yuwei Wang. My research projects could not have succeeded without their technical support and great collaboration. I also want to thank the following group members, CRIS members, and friends for their support and help during my Ph.D. study: Bingzhe Li, Cong Ma, Jinfeng Yang, Ming-hong Yang, Zhichao Cao, Hao Wen, Fenggang Wu, Ziqi Fan and Xiongzi Ge.

Finally, I would like to thank the National Science Foundation (grant no. IIP-1439622) and the member companies for funding my projects.

Dedication

To my family, who always stay with me and provide me with all kinds of support. Especially to my Mom (Huiling Yin), my dad (Ganhu Qin) for their advice, their faith and their endless love.

Abstract

In the current big data era, huge financial losses are caused when data becomes unavailable at the original storage side. Protecting data from loss plays a significantly important role in ensuring business continuity. Businesses generally employ third-party backup services to save their data in remote storage and hopefully retrieve the data in a tolerable time range when the original data cannot be accessed. To utilize these backup services, backup users have to handle many kinds configurations to ensure the effective backup of the data. As the scale of backup systems and the volume of backup data continue to grow significantly, the traditional backup systems are having difficulties satisfying the increasing demand requirement of backup users.

The fast improvement of machine or deep learning techniques has made them successful in many areas, such as image recognition, object detection, and natural language processing. Compared with other system environments, the backup system environment is more consistent due to the backup nature; the backup data contents are not changed considerably, at least in the short run. Hence, we collected data from real backup systems and analyzed the backup behavior of backup users. By using machine learning techniques, we discovered that some patterns and features can be generalized from the backup environment. We used them to as a guid in the design of an intelligent agent called HyperProtect, which aims to improve the service level provided by the backup systems.

To apply machine or deep learning techniques to enhance the service level of the backup systems, we first improved the stability and predictability of the backup environment by proposing a novel dynamic backup scheduling and high-efficiency deduplication. Backup scheduling and deduplication are important backup techniques in backup systems. Backup scheduling determines which backup starts first and which storage is assigned to that backup for improving the backup efficiency. Deduplication is used to remove the redundancy of the backup data to save the storage space. Besides the backup efficiency and storage overhead, we considered maintaining the stability and predictability of the backup environment when processing the backup scheduling and deduplication.

When the backup environment became more stable, we applied machine learning to improve the reliability and efficiency of the large-scale backup system. We analyzed data protection system reports written over two years and collected from 3,500 backup systems. We found that inadequate capacity is among the most frequent causes of backup failure. We highlighted the characteristics of backup data and used the examined information to design a backup storage capacity forecasting structure for better reliability of backup systems. According to our observation of an enterprise backup system, for a newly created client, there are no historical backups, so the prefetching algorithm has no reference basis to perform effective fingerprint prefetching. We discovered a backup content correlation between clients from a study of the backup data. We propose a fingerprint prefetching algorithm to improve the deduplication rate and efficiency. Here machine learning and statistical techniques are applied to discover backup patterns and generalize their features.

The above efforts introduced machine learning for backup systems. We also considered the other direction, namely, backup systems for machine learning. The advent of the Artificial Intelligence (AI) era has made it increasingly important to have an efficient backup system to protect training data from loss. Furthermore, maintaining a backup of the training data makes it possible to update or retrain the learned model as more data are collected. However, a huge backup overhead will result from always making a complete copy of all collected daily training data for backup storage, especially because data typically contains highly redundant information that does not contribute to model learning. Deduplication is a common technique of reducing data redundancy in modern backup systems. However, existing deduplication methods are invalid for training data. Hence, we propose a novel deduplication strategy for the training data used for learning in a deep neural network classifier.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The extraordinary increases in the scale of backup systems and the daily data generated by large enterprise companies are making it extremely challenging to perform data protection efficiently. With their fast improvement, AI techniques have been exhibiting state-of-the-art performance in many areas, thus making it possible to apply them to optimize the efficiency and reliability of backup systems. We aim to design an intelligent agent called HyperProtect and apply it to the current backup infrastructure to leverage its efficiency and reliability.

As for the backup infrastructure, early backup systems used simple command-line tools to perform backup and recovery operations [1]. The shortcomings of these systems included limitations in scaling, poor archive management, and strict backup windows [2, 3, 4]. These shortcomings were addressed using a client-server backup model [5, 6, 7, 8], where the backup system configurations are set on the server side. The server that has the configurations of the backup system, including backup frequency and backup scheduling, is called the master server. To improve the bandwidth-related bottleneck of the master server and the network, media servers have been added to modern backup systems, as shown in Figure 1.1. These servers can manage backup storages and support deduplication to reduce the redundancy of backups before they are allocated to backup storages [9]. Most modern backup systems support a three-tier domain [10, 11, 12, 13, 14, 15], which features clients, a master server, and multiple media servers, as shown in Figure 1.1. Clients are sources that generate backup data, and they are protected in a backup domain. Each client holds a single type of data, such as device, operating

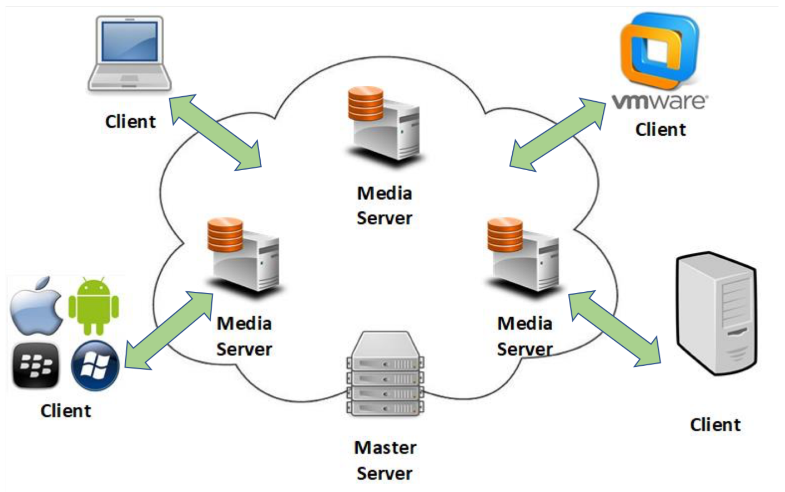


Figure 1.1: Infrastructure of modern backup systems

system, and virtual machine data.

In modern backup systems, some backup operations and techniques have been developed for data protection. We collected data from approximately 600 million jobs ran on 3,871 backup domains and summarized the frequencies of different backup operations and techniques, as listed in Table 1.1. To ensure the consistency of backup and reduce downtime, many systems take a snapshot of the state of the data and then proceed with the backup. To enhance the reliability of the backup system, management operations are scheduled to duplicate the backup images. If the original copy of the data becomes unavailable, recovery will be requested to retrieve the data. The recovery operation has the lowest frequency, as shown in the table. However, its performance cannot be compromised, since huge financial losses will be incurred if the data cannot be recovered in a tolerable time period.

Full and incremental backup operations are the most frequent operations in backup systems, and their predominant scenario is shown in Figure 1.2. In this system, the clients' backups are scheduled in a backup window (e.g., 12:00 a.m. to 3:00 a.m.) that has little effect on regular business. If full backups are constantly performed, the system downtime will be long, and the overhead caused by the backup resources will be large. Therefore, full backups are typically scheduled weekly, whereas incremental backups are scheduled daily. The backup and full intervals are the durations between consecutive

Job operation	Percentage
Full and Incremental	73.0%
Snapshot	6%
Restore (Recovery)	0.9%
Management Operations	20.1%

Table 1.1: Percentage of various job operations on 3,871 backup domains running data protection

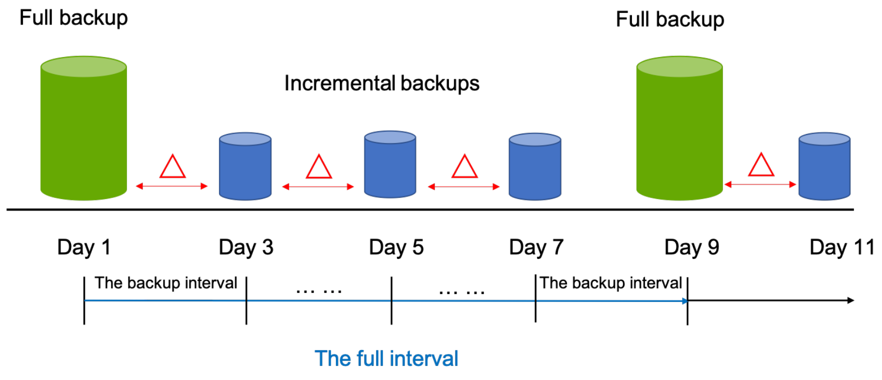


Figure 1.2: Backup scenario including full and incremental backups

backups and full backups respectively. Regarding recovery, the system needs to restore the latest full-backup data along with all incremental data up to the latest recovery point. An increased number of incremental backups scattered among the backup devices can negatively affect the recovery performance. Therefore, another full backup is used to consolidate the backup image. The interval of scheduling full and incremental backups is specified by the backup frequency.

With the increasing scale of backup systems and the exponential growth of the amount of backup data generated from clients, it is becoming hard to satisfy the service level needed by backup users. The thesis will present the design of the intelligent agent HyperProtect, which is intended to be applied into the large-scale backup systems. The main contributions of this thesis are summarized as follows:

- A novel dynamic backup scheduling algorithm, HyperProtect scheduling, is proposed to enhance the stability and predictability of the backup environment while maintaining good backup efficiency.

- A high-efficiency deduplication scenario is introduced to the current large-scale backup system to effectively and efficiently reduce the redundancy of the backup data while preserving the spacial locality of the backup.
- A new structure is presented to effectively forecast the backup storage capacity and predict the time that the storage usage will reach this capacity.
- A machine learning-based deduplication algorithm is developed to effectively reduce the redundancy of the initial backups for newly created clients by efficiently prefetching backup content correlated fingerprints.
- An effective deduplication algorithm is proposed to reduce the amount of training data that contribute little to optimizing the learning network, while maintaining good classification accuracy.

The rest of this thesis is organized as follows: Chapters 2 and 3 present the smart backup scheduling and high-efficiency deduplication of existing clients, which are performed to leverage the stability and predictability of the backup environment. Chapters 4 and 5 present the design of backup storage capacity forecasting structure and the backup content-aware deduplication for newly created clients, which are performed using machine learning techniques. Chapter 6 presents the novel deduplication engine intended for reducing the redundancy of deep learning classifier training data.

Chapter 2

HyperProtect Backup Scheduling

In the large-scale backup system, it is always not trivial to schedule the large amount of the backup data generated from the clients to the backup storage. The backup efficiency of the backup system is a critical performance metric, but the stability and the predictability are still cannot be seriously traded. The good stability and predictability are significantly important for the recovery performance and the implementation of the designed AI agent. This chapter mainly describe the work [16] to enhance the stability and predictability of the backup system, while maintaining a good backup efficiency.

2.1 Introduction

The scale of backup systems are becomeing increasingly large and the amount of data of modern workloads is growing rapidly which results in significant complexity for backup administrators working to maintain the efficiency of backup systems. Amvrosiadis *et al* [1] presented a study of 40,000 enterprise data protection systems and analyzed over 1 million weekly reports. They found that one of the main reasons behind the inefficiency of the data protection is the misconfiguration of job scheduling. Because backup systems are in a constant state of flux, even a perfectly designed static schedule will eventually decay in efficiency as the backup environment changes. The inefficiency of the static configuration drives the need to design an intelligent scheduler to handle the configuration of the backup system dynamically [17][18][19].

The modern backup domain typically consists of three tiers [1]: clients, master server, and media servers, as shown in Figure 1.1. Clients are the sources that produce the backup data. The master server holds information on backup images and configurations of the backup system. Also, it facilitates the communication between the clients and media server. Media servers where the actual backup image residies achieve high efficiency on management of storage media such as tapes and disks.

A lot of research has proposed several ways to improve the efficiency and robustness of the three-tiered backup system introduced above. Considering the vast amount of backup operations that take place in a backup window, the network traffic becomes serious issue. Giat *et al* [20] proposed data prefetching and intelligent scheduling to improve the network issues caused by high latency and limited bandwidth. Van de Ven *et al* [21] created an optimization model to configure the backup frequency in order to alleviate the impact of network congestion. Data loss caused by limited storage space is also a serious problem that occurs when the backup system fails to manage the backup storage devices effectively. Mark [22] designed a storage capacity forecasting tool to predict the availability of storages and perform dynamic storage management before they become unavailable.

A Backup operation, the critical process of data protection, copies the data from the data sources to the remote storage. There are two predominant backup operations, full backup and incremental backup [1]. Full backups store the entirety of the contents to backup storages, while incremental backups only store the newly created and modified contents [6]. The backup resource overhead to perform a full backup is huge. More often than not, the frequency to schedule a full backup is pretty low, for example, every 5 days or fewer, while the incremental backup takes place almost every day [1]. Therefore, it is necessary to design an intelligent backup scheduling algorithm for incremental backups to accelerate the backup system, which is one of the contributions of this chapter.

A backup schedule is a policy to determine which backup starts first and which storage is assigned to that backup. Combining with the finding by Amvrosiadis *et al* [1], Cherkasova *et al* [23] also found that improper schedules of backups degrade the efficiency of a backup system. Cherkasova *et al* [24][23] proposed an effective scheduling algorithm, Longest Backup First (LBF), which achieves good improvement on the backup time, but it solely concentrates on the full backup and fails to account for the restore performance.

In this chapter, we propose an intelligent scheduling algorithm called HyperProtect which schedules backups efficiently while reacting to the backup environment and adapting to the changing system. The HyperProtect algorithm achieves significant reduction on the complexity of backup storage management and the overhead of the restore operation while maintaining good backup performance, which are measured by our proposed metrics (see Section 2.3). The main contributions of this chapter are summarized as follows:

- Proposing storage throughput utilization, backup time consistency, and storage switching to evaluate and compare the performance between various schedulers.
- Creating an automatic scheme to implement Longest Backup First [24] on the backup system as one of the baselines .
- Designing an innovative intelligent backup scheduling algorithm, HyperProtect, to improve the storage switching while maintain good storage throughput utilization and time consistency.
- Performing the empirical comparison and analysis between various schedulers in the backup systems with different scales.

This chapter is organized as follows: Section 2.2 introduces the related work about the backup scheduling. Section 2.3 demonstrates the performance metrics used on the evaluation of the schedulers. Section 2.4 describes the details of the HyperProtect scheduling. Section 2.5 evaluates and analyzes the performance of HyperProtect and compares it to baselines, random and LBF.

2.2 Backup schedule

A backup schedule is a policy to determine which backup starts first and which storage is assigned to that backup. One of the primary causes of inefficiencies in the current data protection system comes from the manual misconfiguration of the backup schedules [1]. The configuration of backup order and storage allocation are set by backup users or administrators manually before processing and will not be re-visited. This static scheduling, even designed efficiently at the beginning, gradually becomes chaotic and ineffective due to the constant change of the backup system. Because of this eventual ineffectiveness, it becomes promising to move from a static backup system to a dynamic backup system. Cherkasova *et al* [23] found that improper schedules of backups will increase the time to complete backup tasks. An example is shown in Figure 2.1. When backups are scheduled as in ascend order of time, the completion time of the backup system is 14 hours, as shown in Figure 2.1a. Longest Backup First (LBF) [24], a backup scheduling idea to schedule backups as in descend order of time, achieves reduction of the backup time by 4 hours, compared with the suboptimal backup schedule. However, we need to obtain the time to process every backup before running the system when applying the LBF algorithm. In this paper, we design an automatic scheme to implement the LBF using historical data to estimate the time needed for every backup, which is introduced in Section 2.4.

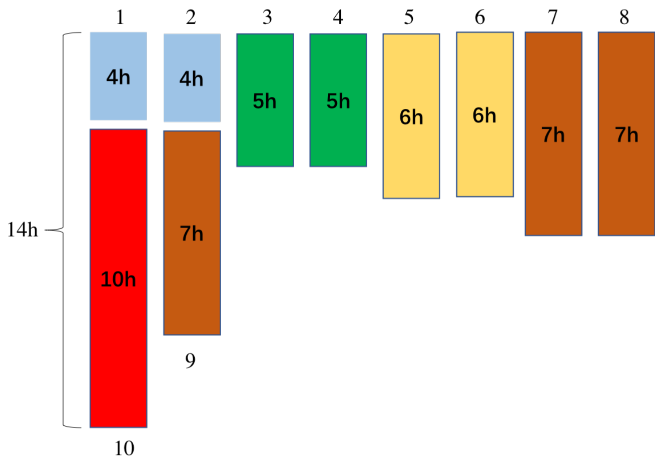
2.3 Backup Performance Metrics

To analyze and compare the performance between various schedulers, we introduce three metrics: storage throughput utilization, backup time consistency, and storage switching. These metrics help characterize backup performance, restore performance and the complexity of storage management.

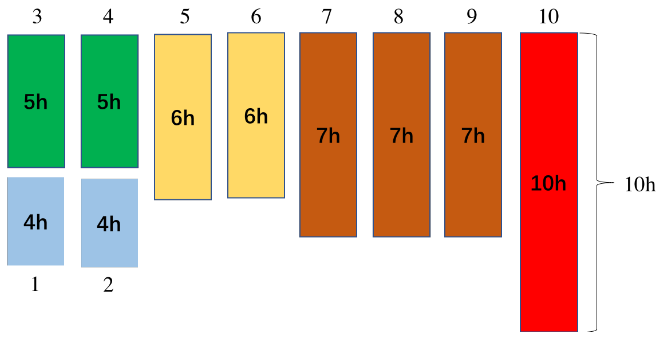
- Storage Throughput Utilization

  Backup time is a commonly used metric to measure the completion time needing for backup workloads. However, to compare and analyze the backup efficiency of various backup scheduling algorithms, we need to normalize the traditional metric, backup time, to storage throughput utilization. The most efficient scheduling of backups is one which fully utilizes the backup resources as measured using storage throughput utilization formulated by Equation 2.1, where *total data size* is the size of all clients' backups and *total time* is the time to complete all clients' backups. A larger value of storage throughput utilization means the scheduling algorithm achieves better utilization of backup resources.

$$Storage\ Throughput\ Utilization = \frac{\frac{total\ data\ size}{total\ time}}{storage\ throughput} \tag{2.1}$$



(a) The inefficiency caused by suboptimal scheduling



(b) Improved scheduling using Longest Backup First

Figure 2.1: Demonstration of inefficiency due to misconfiguration of scheduling

- Backup Time Consistency

  Most of the modern businesses want to maximize uptime and minimize time down for backups and therefore set tight windows for backups. More often than not, they schedule backup processes during a specific backup window with the least impact on regular business. Therefore, it is necessary for the backup time to be predictable and consistent, which allows companies to better schedule their backups during the downtime. This metric, backup time consistency is quantified by the backup time variation , as shown in Equation 2.2, where N is the number of backup times among different days and t_i is the completion time for backup process at day i. To clearly visualize the backup time consistency, we fit various backup times for different days on the same system to Gaussian distribution. The backup time of the scheduling is less consistent when the time variation calculated by Equation 2.2 is larger.

$$\textit{Backup time variation (N)} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(t_i - \overline{t})^2} \tag{2.2}$$

- Storage Switching
  Allowing backup jobs from the same client to write to different storage units on different days brings challenges and complexities to storage management. Too many pieces of backup data for a client spread across to too many storage units can result in a negative impact on the restore performance, particularly for the deduplication case. Even though the restore frequency is as low as 0.8% [1], restore operations remain of utmost importance to customers due to the value of any downtime while they are occurring. Therefore, it is important to design the backup scheduling algorithm to reduce the storage switching while maintaining good storage throughput utilization and backup time consistency. The storage switching is examined by what percentage of backups which are scheduled to different storage units from the previous day. It is quantified by Equation 2.3, where $s_i(d)$ represents the storage unit where the backup generated by $client_i$ is assigned at day d and N is the total number of backups. Larger storage switching computed by Equation 2.3 causes more negative impact on the storage management and restore performance.

$$\textit{Storage switching (d)} = \frac{\sum_{i=1}^{N}(s_i(d) \neq s_i(d-1))}{N} \tag{2.3}$$

2.4 Intelligent Backup Scheduling

Intelligent backup scheduling determines which backup starts first and which storage is assigned to that backup. The current data protection system does not schedule workloads intelligently when more than one client ask for backups simultaneously. As mentioned in Section 2.2, the improper schedules result in inefficiencies of the backup system.

The novel scheduling algorithm called HyperProtect achieves significantly better storage switching while maintaining similar storage throughput utilization and backup time consistency, compared with Longest Backup First [23].

We propose an automatic scheme to schedule a large amount of backups that are requested at the same time, as shown in Figure 2.2. More details are described in the following steps to determine which backup starts first and to which storage unit the backup is assigned:

1. **Select the storage unit with highest throughput availability.**
   The current backup system sets their backup jobs to storage units statically. Based on our observation, that configuration causes an imbalance of storage allocation. When some storage units are occupied with too many workloads some of the allocated backup jobs cannot start within the scheduled window leading to unprotected data. Even having a good storage allocation at the beginning, the efficiency of that static allocation will decay with a change of the backup environment. The backup time is optimized when the storage throughput is fully utilized. Therefore, we define the throughput availability of storage devices to use as a heuristic to determine which storage is most able to handle additional load. The available throughput availability is determined by the total throughput of the storage and the number of backups is running. The throughput assigned to every backup is deceased when the number of the processing backups running on it is increased. The throughput availability is quantified by Equation 2.4.

$$Throughput\ availability = \frac{total\ throughput}{\#\ of\ processing\ backups} \tag{2.4}$$

2. **Estimate the backup time for every pending backups based on the selected storage.**
   To improve the storage throughput utilization, the idea of Longest Backup First needs to estimate the time for completing every backup job. Since the bandwidth of the storage will be fully utilized at the steady state when using our proposed automatic scheduling scheme, the estimated backup time for every backup are quantified by Equation 2.6.

$$Estimated\ backup\ time = \frac{backup\ size}{\frac{Storage\ throughput}{Bandwith\ of\ the\ storage}} \tag{2.5}$$

3. **Select n candidate backups with longest estimated backup time and sort them in backup time consistency list.**

   To improve the time consistency of the backup system while maintaining good storage throughput utilization, select top n candidate backups with longest backup time to optimize for backup time consistency. The candidate backup which mostly reduce the variation of the backup time has highest priority to be scheduled. The variation of every candidate backup is quantified as the difference between the estimated backup time and the average historical backup time, as shown in Equation 2.6. Sort candidate backups based on the priority of backup time consistency. The candidate backup with longer estimated backup time has higher priority when they have the same time variation.

$$\begin{aligned} \textit{Backup time variation } &= (\textit{estimated backup time} \\ &- \textit{average}[\textit{historical backup times}])^2 \end{aligned} \tag{2.6}$$

4. **Select n candidate backups with longest estimated backup time and sort them in storage switching list.**

   Similarly, select top n candidate backups with longest backup time to optimize for reducing storage switching. The candidate backup whose storage unit was assigned at previous day is the selected storage has higher priority to be scheduled. Sort candidate backups based on the priority of storage switching. The candidate backup with longer estimated backup time has higher priority when they have the same priority of storage switching.

5. **Assign priority values based on weighted orders from 3) and 4)**

   After sorting lists based on backup time consistency and storage switching, every candidate backup is assigned priority values, P_{tc}, P_{sw} based on those two lists. For the priority value based on the storage switching, the candidate backup is assigned a high priority value $P_{sw} = 0.9$, if its storage unit which is allocated at previous day is the same as the current selected one. The others are assigned $P_{sw} = 0.2$. The priority values based on the backup time consistency are assigned to candidate backups using a linear function which is shown in Equation 2.7, where n is the number of candidate backups selected from backups with longest completion time,

Rank	1	2	3	4	5
Backup #	backup 1	backup 3	backup 2	backup 5	backup 4
P_{tc}	1	0.8	0.6	0.4	0.2

Table 2.1: Mapping 5 candidate backups to their priority values based on respective sorting list

Figure #	2.3a	2.3b	2.3c
Time variation	551 seconds	725 seconds	554 seconds

Table 2.2: Time variation at seconds in 60 simulated days for different configurations shown in Figure 2.3a, 2.3b, 2.3c

r is the rank based on the backup time consistency list. An example is shown in Table 2.1. The backup with highest total priority is quantified by the weighted sum of those three priority values, which means $P_{total} = w_{tc}P_{tc} + w_{sw}P_{sw}$, where $w_{tc} + w_{sw} = 1$.

$$P_{sw} = \frac{n\ +\ 1\ -\ r}{n} \tag{2.7}$$

$$\textit{Storage switching (n)} = (1 - \frac{1}{\#\ \textit{of storage units}})^n \tag{2.8}$$

2.5 Experimental Results

We constructed a backup simulator based on Java in order to emulate the realistic backup systems for the performance testing of various intelligent scheduling algorithms. The simulator is capable of generating backup systems with different scales, allowing us to design and test intelligent scheduling algorithms, and emulate the backup process. Section 2.5.1 presents the performance behavior of intelligent scheduling algorithms introduced in Section 2.4, by varying parameters in the algorithm. Section 2.5.2 does

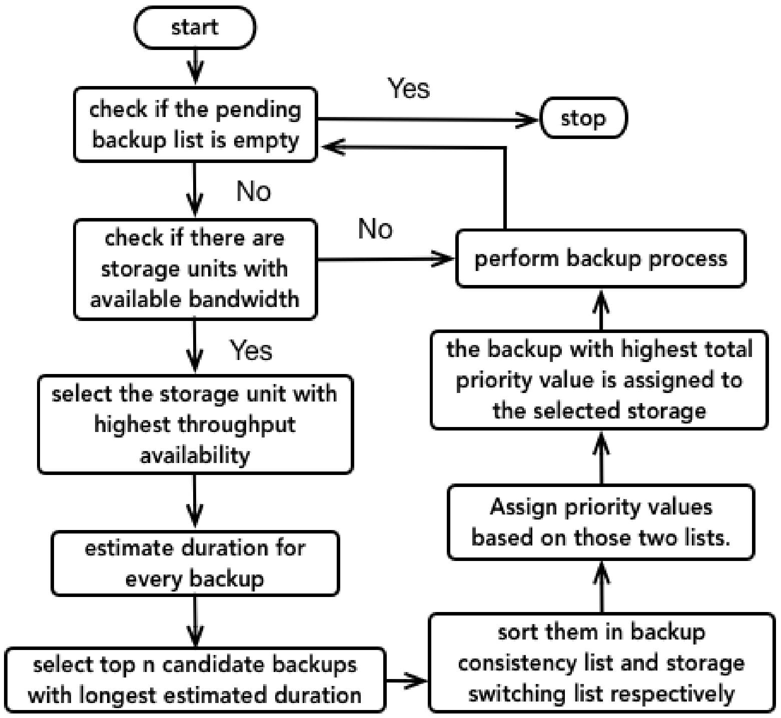

start
check if the pending backup list is empty
Yes
stop
No
check if there are storage units with available bandwidth
No
perform backup process
Yes
select the storage unit with highest throughput availability
the backup with highest total priority value is assigned to the selected storage
Assign priority values based on those two lists.
estimate duration for every backup
sort them in backup consistency list and storage switching list respectively
select top n candidate backups with longest estimated duration

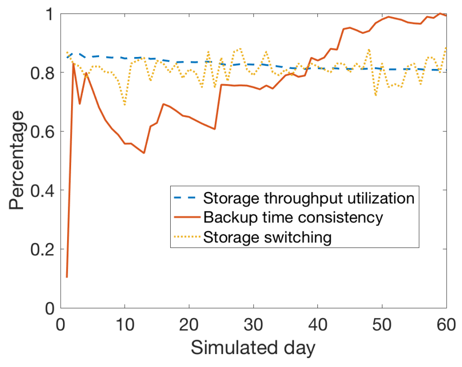
Figure 2.2: Flowchart describing the intelligent scheduling algorithm

a comparison between different scheduling algorithms based on performance metrics introduced in Section 2.3.

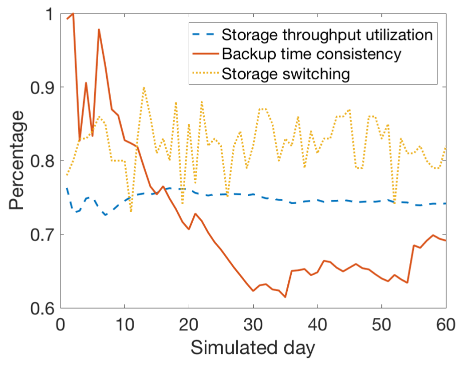
2.5.1 Evaluation of Intelligent scheduling algorithm

The automatic scheme of the backup scheduling algorithm gives us a convenient way to test various scheduling algorithms focusing on storage throughput utilization, backup time consistency, and storage switching by tuning three parameters, n, w_{tc} and w_{sw}. We use our simulator to construct the backup system with 100 clients and 6 storage units and simulate a backup process with different configurations of parameters, which is shown in the following:

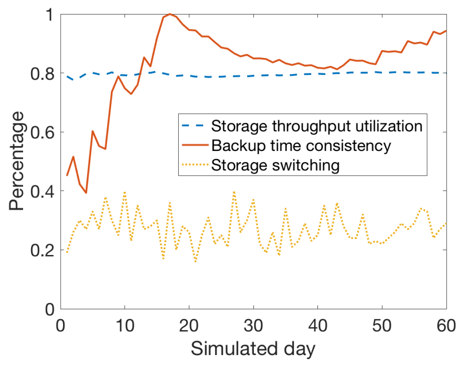
- Storage throughput utilization oriented: $n = 6$, $w_{tc} = 0$, $w_{sw} = 0$
  Our automatic scheme can generalize the existing scheduler, Longest Backup First (LBF) by this setting of parameters. The experimental results for 60 simulation days based on storage throughput utilization, backup time consistency, and storage



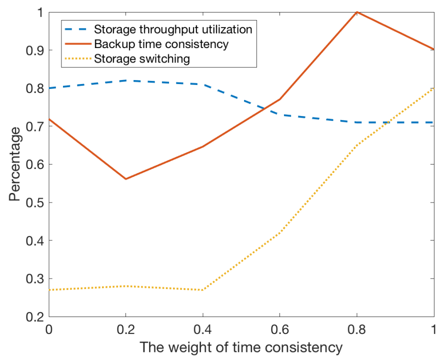
(a) $n = 6$, $w_{tc} = 0$, $w_{sw} = 0$



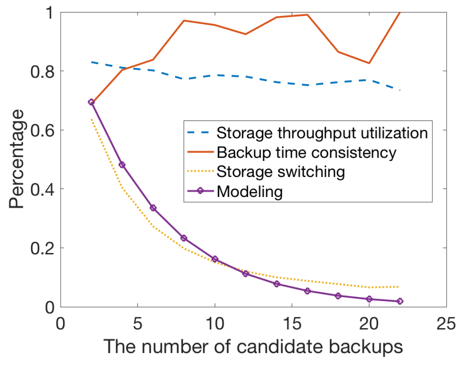
(b) $n = 6$, $w_{tc} = 1$



(c) $n = 6$, $w_{sw} = 1$



(d) $n = 6$, $w_{tc} = 0, 0.2, 0.4, ..., 1$ and $w_{sw} = 1 - w_{tc}$



(e) $n = 2, 4, 6...24$, $w_{sw} = 1$

Figure 2.3: The variation in performance from tuning parameters n, w_{tc} and w_{sw}

switching are shown in Figure 2.3a. The storage throughput utilization, backup time consistency, and storage switching are calculated by Equation (2.1) - (2.3). The backup time consistency is normalized to a percentage by dividing the maximum value of time variation among different days. From the results, we can see the storage throughput utilization remains stable at around 81% from day 1 to day 60. However, the normalized variation of backup time increases from 45% to 100% from day 10 to day 60. The storage switching occurs frequently at around 80% in 60 simulation days.

- Backup time consistency oriented: $n = 6$, $w_{tc} = 1$
  If only accounting for a scenario to reduce the variation of backup time, the experimental results in Figure 2.3b shows the normalized variation of backup time keeps decreasing, but storage throughput utilization is reduced by around 5% and storage switching still keeps in high values around 80%.

- Storage switching oriented: $n = 6$, $w_{sw} = 1$
  If only accounting for a scenario to reduce storage switching, the experimental results in Figure 2.3c shows that storage switching is reduced by around 45% while storage throughput utilization maintains at 80% and the normalized variation of backup time remains at a similar value after day 20. Even though the time variation is reduced as the increase of simulation day by tuning parameters to optimize backup time consistency, as shown in Figure 2.3b, the value of the variation is 725 which is the largest compared to other two configurations as shown in Table 2.2. The automatic scheme with $n = 6$, $w_{tc} = 1$ has a similarly small time variation to that of LBF ($n = 6$, $w_{tc} = 0$, $w_{sw} = 0$).

- Vary weight w_{tc} and w_{tc}: $n = 6$, $w_{tc} = 0, 0.2, 0.4, ..., 1$ and $w_{sw} = 1 - w_{tc}$
  The experimental results in Figure 2.3d exhibit the behavior of those three performance metrics by varying the parameters w_{tc} and w_{sw}. We can see that the variation of the backup time and the value amount storage switching keeps increasing while storage throughput utilization is deceasing as we add more weight to w_{tc}. The introduction of w_{tc} can reduce the normalized variation of backup time when the simulation day increases, as shown in Figure 2.3b, but the overall

time variation is the largest compared to the other two configurations of scheduling algorithms, as shown in Table 2.2. Also, storage throughput utilization is reduced and storage switching has no improvement, compared to those of Longest Backup First. Based on those results we find that those three performance metrics are not independent of each other. The backup time of every day is minimized when backups fully utilize backup storage units using Longest Backup First, and subsequently the variation of backup time is also reduced. Hence, the overall performance can be optimized when the system can fully utilize storage throughputs while controlling storage switching, which can be achieved by setting $w_{sw} = 1$.

- Vary the number of candidate backups: $n = 2, 4, 6...24$, $w_{sw} = 1$
  The last parameter, n, inside the algorithm is the number of candidate backups selected from backups with longest estimated backup time. The other explanation of n is the value of freedom to be introduced to control storage switching. The larger n is the more control the scheduling has on storage switching. The performance of various configurations of n is shown in Figure 2.3e. The results show that storage throughput utilization is reduced from 82% to 75% while storage switching significantly decreases from 63% to 8%. The relationship between storage switching and n can be derived by Equation 2.8, which is shown by circle (purple) line. We can see the modeling of storage switching is almost the same as the storage switching measured by real data which is shown by the dotted (yellow) line. From the modeling, we find that the reduction of storage switching become less significant when n reaches 6 which is the number of storage units in the backup domain. Simply, n can be configured as the number of storage units to maintain good storage throughput utilization while significantly reducing storage switching. Hence, HyperProtect is implemented using the automatic scheme introduced in Section 2.4 with $n = 6$, $w_{sw} = 1$.

2.5.2 Performance comparison between different scheduling algorithms

In this section, we compare performance of HyperProtect with the existing scheduling algorithm [24], Longest Backup First (LBF) and our designed baseline algorithm, Random, using metrics introduced in Section 2.3. Random is an algorithm designed for the baseline and to simulate how inefficient backups can be when misconfiguring the schedule. LBF is the other baseline and we are going to verify if it can obtain a better backup efficiency through improving the storage throughput utilization, compared to Random. HyperProtect proposed in this paper aims to reduce the storage switching while maintaining good storage throughput utilization and backup time consistency. The experiment is set up in the following:

- Set up backup systems with various scale.
  The scale of the current backup system is small. For example, a single master server protects a small amount of clients [1]. Our project aims at large scale backup systems where a single master server protects over 100 clients and each client has more than 100GB data. We test three backup systems - System A: 100 clients and 6 storage units, System B: 300 clients and 6 storage units and System C: 500 clients and 6 storage units.

- Simulate the backup process using the three scheduling algorithms for 60 simulated days.
  The backup behavior of the system is that all clients in the same domain ask for the backup of whole system simultaneously every day. Intelligent scheduling algorithms determine which backup starts first and which storage unit is selected for the backup. Due to the naturally changing size of backups for all clients, intelligent algorithms are supposed to automatically update schedules and make them adapt to the dynamic environment.

- Collect the performance metrics of various backup processes.
  To perform comparison between various scheduling algorithms, three metrics - storage throughput utilization, backup time consistency, and storage switching introduced in Section 2.3 are used to evaluate backup efficiencies, restore performance and the complexity of storage management.

- Repeat system tests for five times to get performance results with 95% confidence level.
  The performance of the scheduling varies due to the randomness introduced to the sizes of backup data for all clients. Therefore, we repeat the experiments for 5 iterations to get performance results with 95% confidence level.

The experimental results for storage throughput utilization and storage switching tested in different scale backup systems are shown in Figure 2.4. The worst storage throughput utilization of Random indicates that the improper configuration of backup scheduling causes the inefficiency of the backup system. LBF achieves around a 30% improvement on the storage throughput utilization compared with Random. However, LBF has similar storage switching to Random, at around 81%. The large number of the changing storage units assigned to every client results in a negative impact on the restore performance due to the scattering of the backup data. The frequent storage switching also increases the complexity of storage management. HyperProtect achieves a reduction of storage switching of around 55% compared to LBF.

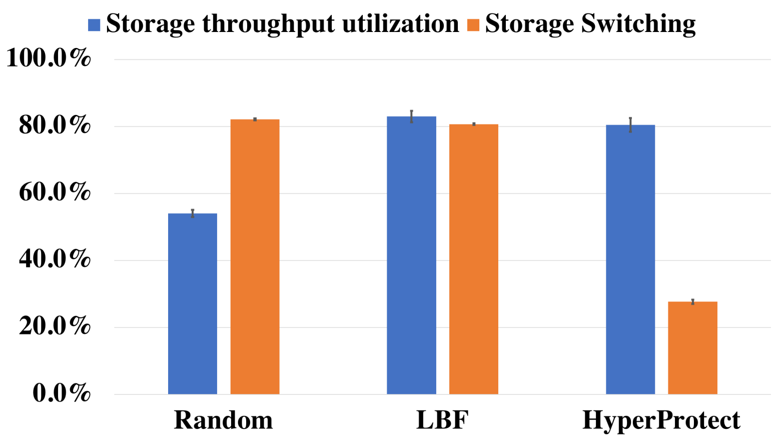
Based on the experimental results as shown in Figure 2.4, we can see that an intelligent backup scheduling algorithm is significant for enhancing efficiencies of backup systems. HyperProtect achieves significant reduction on storage switching while maintaining similar storage throughput utilization. We also find that the performance of HyperProtect is robust to backup systems with different scales.

To perform comparison of backup time consistency between different scheduling algorithms, we fit the daily backup time of the backup system for 60 simulated days to the Gaussian distribution for visualization. The backup time consistencies presented with Gaussian distribution (μ, σ) of different scheduling algorithms are shown in Figure 2.5. The less variance of the distribution, the better the backup time consistency.

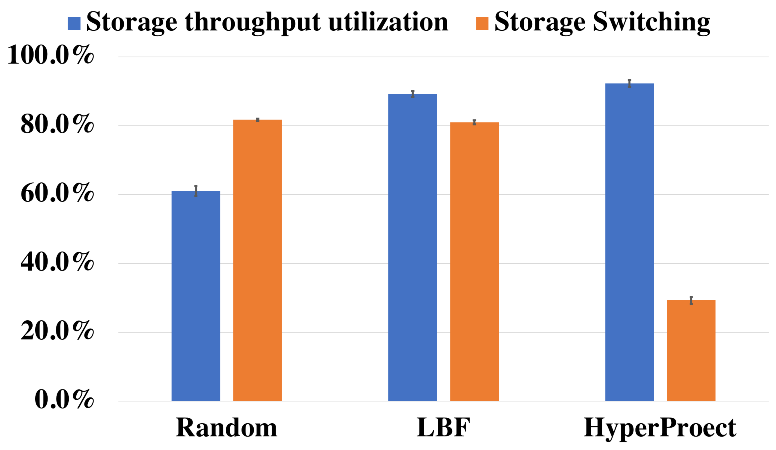
From the experimental results as shown in Figure 2.5, the wide distribution of Random exhibits largest variance of the backup time compared to LBF and HyperProtect. Among those three scheduling algorithms, LBF has the best storage throughput utilization, which has the best backup time consistency. The backup time of HyperProtect is also quite consistent, stable and robust to different scales of backup system, which is within 10% of LBF. Therefore, HyperProtect succeeds in improving overhead of the restore and ease of storage management, but also maintaining efficient resource allocation and effectiveness of backup window schedule compared with LBF.

2.6 Conclusion

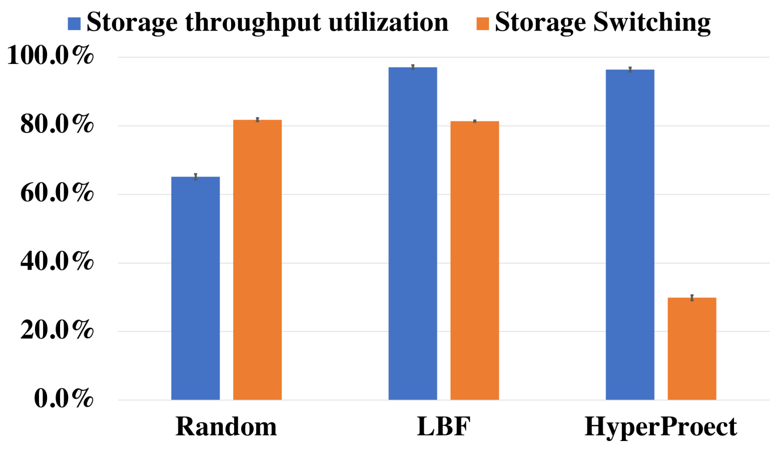
One of critical reasons behind the inefficiency of data protection, misconfiguration of backup schedules [1], drives the need for moving to dynamic backup system with intelligent scheduling algorithms. To design and evaluate the performance of backup scheduling algorithm for dynamic backup systems, we propose three performance metrics: storage throughput utilization, backup time consistency, and storage switching. Based on the experimental results, we verified that an unintelligent scheduling algorithm, Random, hurts efficiencies of data protection substantially. The LBF improves storage throughput utilization by around 30%, which significantly reduces the down time of the regular business. However, the frequent storage switching of LBF results in a negative impact on the restore performance and increases the complexity of storage management. HyperProtect significantly reduces the storage switching by around 55% while maintaining similar storage throughput utilization and backup time consistency within 1% and 10% of LBF respectively.



(a) Performance test of various schedules in Backup System A

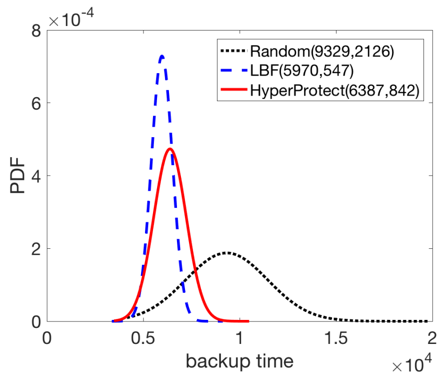


(b) Performance test of various schedules in Backup System B

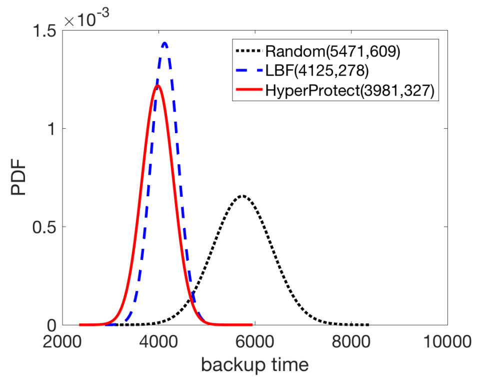


(c) Performance test of various schedules in Backup System C

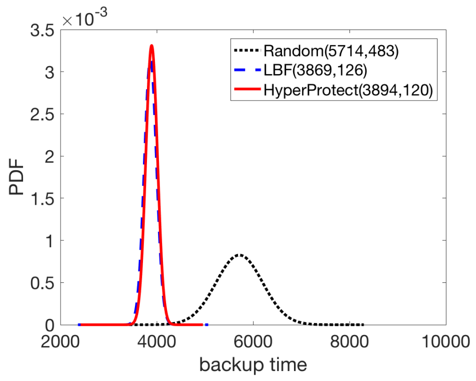
Figure 2.4: Comparison of storage throughput utilization and storage switching between Random, LBF and HyperProtect



(a) Backup time variation(μ, σ) of different schedules in Backup System A



(b) Backup time variation(μ, σ) of different schedules in Backup System B



(c) Backup time variation(μ, σ) of different schedules in Backup System C

Figure 2.5: The backup time consistency of different dynamic scheduling algorithms compared to a Gaussian Distribution

Chapter 3

Backup Deduplication for Existing Clients

Deduplication is a core technique to reduce the storage overhead by removing the redundancy of the data. In the backup system, the most storage space will be saved if lookup all historical backup data for deduplication. However, that deduplication process will very time-consuming and seriously hurt the stability and predictability of the backup system. In this chapter, we are going to introduce the high-efficiency deduplication for the existing clients, which is being implemented on the current backup system [15].

3.1 Introduction

Deduplication ("dedupe") is a core technique for effectively detecting and reducing the redundancy of the data to enable the storage of 10-50 times more data in the same storage space [9]. Dedupe systems typically divide the data streams into segments based on a certain segment algorithm and each of them is cryptographically hashed to the signature called a fingerprint by using a known secure hashing algorithm [25]. Fingerprints are looked up to determine whether the segment content is unique, and only the unique segment content is saved into the dedupe system. Dedupe not only saves storage space [25, 26, 27, 28], but also improves the network bandwidth utilization [29, 30, 31, 32] by eliminating the transmission of duplicated data.

To ensure the efficiencies of the data protection, most enterprise backup systems [12, 13, 15, 33] employ a categorized backup infrastructure to configure backups instead of using one-client-fits-all-backup infrastructure. Without the loss of generalization, the enterprise backup system we focus on employs a most common used infrastructure, client-policy, to manage backups [12, 13, 15]. A backup administrator creates multiple *backup policies* (herein referred to as simply "policies") to manage and protect data contents with different backup types and different expected service levels. Each policy holds a set of configurations shared by a group of *backup clients* (herein referred to as simply "clients"). The configurations are set based on the backup type and the service level, which mainly includes the clients that will be protected by this policy, a full or incremental backup that will be scheduled, the storages that will be assigned to this policy. The client is a source that generates the backup data and could be a virtual machine, a laptop, or a device. To improve backup performance and reducing the backup overhead, the backup stream is deduplicated and stored into the storage units called containers [9], which could be large files in a file system or continuous blocks in storage.

In order to ensure the best dedupe rate, some dedupe systems employ *full indexing* - the maintenance of completed indexing information for all unique backup segments processed so that all duplicates can be detected and eliminated. The size of the indexing is proportional to the amount of data to be managed. Due to the limitations in the memory space and the explosive increase in unique backup segments, the index information has to be stored on the scalable disks. Therefore, a number of on-disk indexes will be accessed during the fingerprint lookup, which results in the poor dedupe performance [26, 34, 35]. However, most backup services requires high dedupe throughputs and allows for the tolerance of some duplicates. In this case, most dedupe systems, particularly inline dedupe systems that deduplicates the backup data before or while it is sent to backup storages, use *partial indexing* to reduce the number of storage reads. In the partial indexing, a portion of the fingerprints are prefetched to the memory for dedupe, which is our focus. The fingerprints that are not in the memory are considered unique, and their corresponding contents are stored in the containers or blocks. Because of the cost considerations, the memory is not adequate for holding all unique fingerprints, and therefore, maintaining the effective fingerprints of on-doing backup jobs in the memory for use in indexing is critical in order to avoid the frequent accessing of the on-disk index and achieve an optimal dedupe rate.

3.2 Background and Motivation

This section will first introduce the client-policy dedupe systems that use partial indexing, which we are targeting. Then we will present the motivation for exploring a more efficient and effective fingerprint prefetching algorithm for today's enterprise backup systems.

3.2.1 Deduplication Processes

Most modern dedupe backup systems [10, 11, 12, 13, 14, 15] use a client-server infrastructure to implement data protection. A copy of the data to be protected is generated from the client side and then sent across the network to be stored on the server side, where there are a number of storage areas for maintaining copies of the backups. Generally, there are three kinds of dedupe processes implemented on modern backup systems, which are shown in the Figure 3.1.

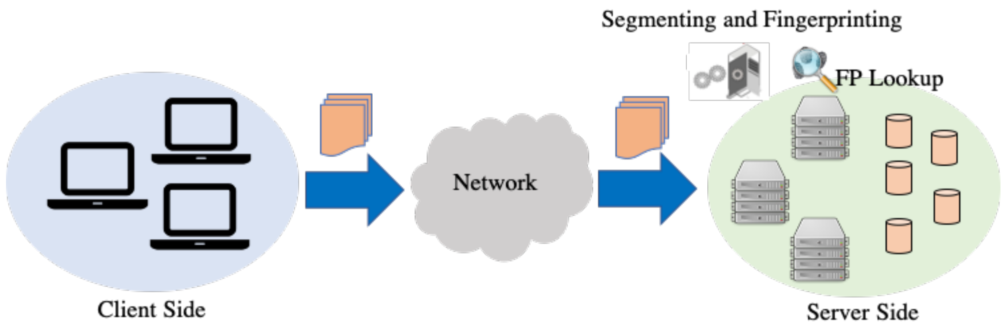
The dedupe process is implemented on the server side as depicted in Figure 3.1a. Backups from the clients are sent to the servers, where fingerprint lookup processing is done for dedupe. However, a big scalability issue exists with this method. When a large number of clients are added to the backup system, the explosive growth of data results in the severe network traffic issues. In addition, an exponential increase of the backup overhead is incurred for additional CPU, memory, and the other resources on the server side to maintain the service levels. To address this issue, the dedupe process shown in Figure 3.1b performs the segmenting and fingerprinting on the client side based on the fingerprint lookup on the server side. However, in the backup systems that protect a large amount of data, the serious dedupe performance issues such as network traffic problems and long latency still exist due to the cross-network fingerprint lookup. The other dedupe process is described in Figure 3.1c which is the one this paper focuses on. In this system, the dedupe is distributed to the client side by prefetching effective fingerprints from the server side, which significantly alleviates the network traffic and latency issues. Also, this dedupe process has good scalability because the computing resources required are provided by the clients themselves. Modern dedupe backup systems aapply one or a hybrid of these three dedupe processes.

The dedupe system we are looking at uses the client-policy infrastructure to manage backups, as shown in Figure 3.2. The backup administrators typically create multiple policies based on different backup types and different agree-upon service levels. A single policy protects multiple clients, and the clients in a policy share the same backup frequency. A new full or new incremental backup is scheduled based on the configuration of backup frequency. Due to the different characteristics of full and incremental backups, the incremental backups generally have much lower redundancy than do full backups. Our paper only considers the full backup, the terms "backup" in the remaining paper refers to a full backup. The duplicates of the backup stream will be identified and eliminated, and then the unique data segments are saved to the containers in a log fashion to preserve the locality. Due to the dedupe, the containers may be shared by several backups.

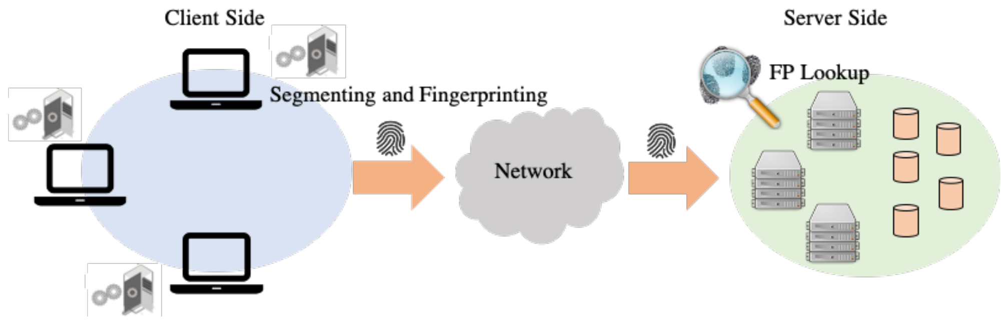
3.2.2 Fingerprint Prefetching for Partial Indexing

Unlike full indexing, partial indexing chooses to sacrifice some dedupe rates for dedupe performance. In partial indexing, a portion of fingerprints (indexing information of processed backup contents) are prefetched to memory and used for identifying and eliminating duplicates within the backup stream. The duplicates that are not identified are treated as unique contents and stored in the containers.

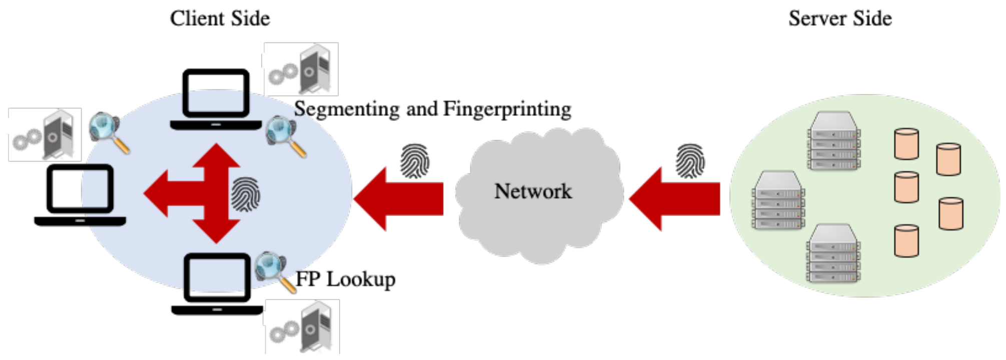
A good prefetching algorithm can improve the dedupe rate and provide good dedupe performance while consuming limited memory. Sparse Indexing [34] samples a small portion of fingerprints (referred to as hooks in some studies) from each block based on the ratio of the number of indexes and the storage capacity and then counts the number of hook hits. All fingerprints in the containers with a high number of hook hits are prefetched to the memory. However, the blocks defined in Sparse Indexing contains duplicates, which reduces the number of effective hooks that can be sampled. Therefore, the Progressive Sampling [9] changes from the block fashion to the container, where only the unique segments are stored, but the order of original files is preserved. However, the dedupe rate of the Progressive Sampling is seriously affected when the number of backups increase due to the decreasing quality of the sampled fingerprints.



(a) The dedupe is processed on the server side.



(b) The segmenting and fingerprinting are processed on the client side, while the centralized fingerprint lookup is on the server side.



(c) The dedupe is distributed to the clients by prefetching effective fingerprints from the server side.

Figure 3.1: Three kinds of dedupe processes implemented on modern backup systems.

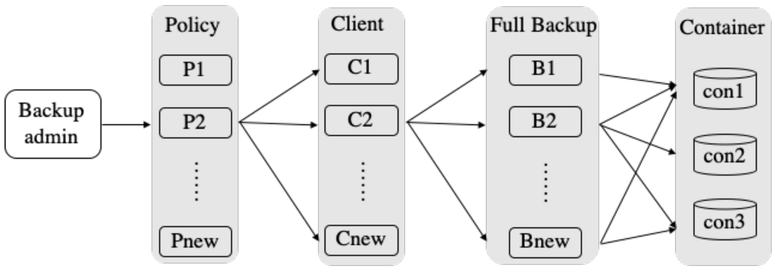


Figure 3.2: Client-policy infrastructure of dedupe systems

3.3 Backup Content Correlation

This section introduces the backup content correlation among backups protected by client-policy backup systems. The correlation is generalized from the real-world dataset. Backup content correlation is signified as follows: temporal locality, spatial locality, and similarity. *Temporal locality* indicates the correlations between backups in the time dimension, where two backups from the same client in a row will share the most data. *Spatial locality* refers to container use of a backup. If a portion of the data is saved in a container, neighboring containers are likely to be used to store the remaining data. *Similarity* refers to the correlations between backups in terms of the shared segments. The more data that are shared by two backups, the greater the similarity of the two backups.

We collected real-world data from a production system and focused on the five policies and 55 clients from January to July in 2019, when around total 250 total full backups were scheduled in different clients. The dataset is consisted of policy ID, client ID, backup ID, fingerprint of each segment, size of each segment, and container ID of each segment.

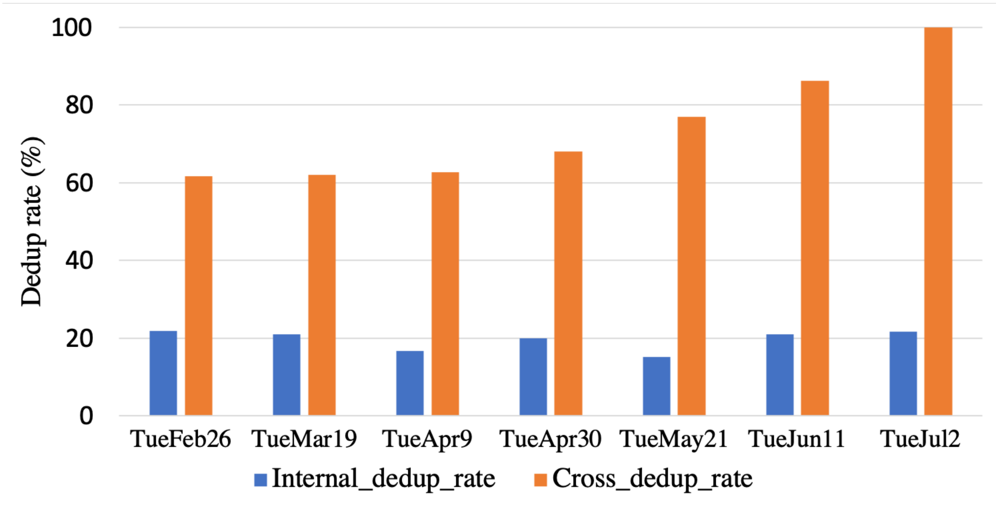
Backup administrators configure the interval for each client to run a full backup based on the agreed-upon service-level objectives, typically every five days or more often [1]. More duplicates will be identified if the fingerprints from the highly correlated backups are prefetched. The two metrics of internal dedupe rate, *internal*, and cross-dedupe rate, *cross*, are used to quantify two kinds of redundancy. Here, *internal* quantifies the redundancy within a $backup_i$ and is formulated as the total size of the backup, *total*, and the size of the unique segments that form the backup, *unique*, which is shown in Equation 5.1. In addition, $cross_{ij}$ specifies the total redundancy of backup$_i$

that can be removed when prefetching backup$_j$ as the base for dedupe and is quantified as shown in Equation 5.2. The $unique_{ij}$ specifies the $unique_i$ that excludes the size of segments from the backup$_j$. By this definition, the calculation of $cross_{ij}$ includes the internal dedupe. Therefore, the minimum of $cross_{ij}$ equals the $internal_i$, which means the prefetched backup$_j$ does not offer any effective fingerprints for dedupe.

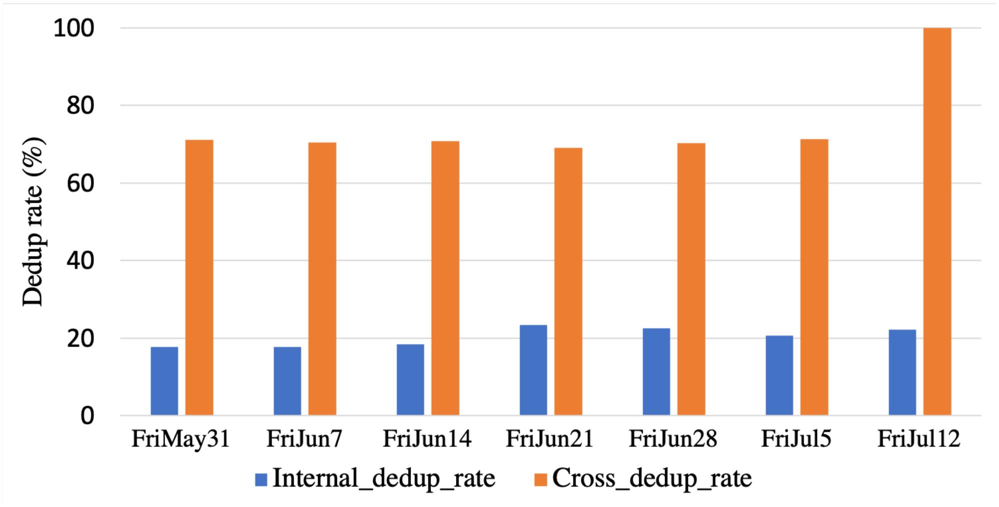
$$internal_i = \frac{total_i - unique_i}{total_i} \times 100\% \tag{3.1}$$

$$cross_{ij} = \frac{total_i - unique_{ij}}{total_i} \times 100\% \tag{3.2}$$

When a new backup in an existing client is scheduled, the prefetching approach will determine the highly correlated base backups, then fetch their fingerprints to the memory for improving the cross-dedupe rate. We performed a study on a real-world dataset and generalized the correlation between backups from each client. All clients in our dataset follow two dedupe patterns in terms of internal dedupe rate and cross-dedupe rate, which are shown in Figure 3.3. The internal dedupe rates of the seven latest backups in each client were calculated, and we found that the internal dedupe rates of backups in a client are similar. Cross-dedupe rates between the latest backup (TueJul2 or FriJul12) and previous backups in a client were calculated. The calculation of the cross-dedupe rates for TueJul2 and FriJul12 shown in Figure 3.3a and Figure 3.3b were to prefetch themselves for dedupe, and thus, the cross-dedupe rates attain 100%. Excluding the 100% cross-dedupe rate, the other cross-dedupe rates of all clients can be categorized as two patterns: Increased Cross and Consistent Cross. The cross-dedupe rate in Increased Cross rises when when the base backup prefetch time is closer to the time when the processing backup that is requested, while the cross-dedupe rates on different generated days in Consistent Cross are similar. The Increased Cross results from the change in backup data that happens in different locations over time. In contrast, the Consistent Cross results when the backup data is changed only in a certain range over time. In the dataset, 52 out of 55 clients satisfied the criteria for Increased Cross, and only 3 out of the 55 clients followed Consistent Cross, which shows that the backups in the same client have **high temporal locality**. In addition, we can see the **high levels of similarity** among the backups generated in the same client, with at least a 3x dedupe rate improvement in cross-dedupe rate compared to the internal dedupe rate.



(a) Increased Cross: 52/55 clients followed this dedupe pattern.



(b) Consistent Cross: 3/55 clients followed this dedupe pattern.

Figure 3.3: Two patterns of cross-dedupe rate in Bk-ExClient.

3.4 Conclusion

This Chapter presents the dedupe infrastructure and process implemented in the current backup systems. From the study of the backup data collected from the real backup systems, we found the backup contents in the same clients have **high level similarity** and **temporal locality**. Besides, combined with container-version backup, the **high spatial locality** is gained by prefetching the latest backups to the memory in client side for dedupe, which enhances the stability and predictability of the backup systems.

Chapter 4

Backup Storage Capacity Usage Forecasting

The first two chapters introduced the work on improving the stability and predictability of the backup system by the scheduler and deduplication. This chapter is going to present the application of machine learning techniques to forecast the backup storage capacity [36].

4.1 Introduction

Data protection plays a critical role in ensuring business continuity. Due to a combination of steep growth in data generated as well as limited budgets, most businesses employ enterprise data protection services, such as Veritas' NetBackup [15], EMC's Avarmar [14], and CommVault's Simpana [12]. However, backup systems are quickly becoming unacceptable, as the service demand from the backup users increases, service level goals of different data types vary and a static system for a non-static backup environment.

The increased complexity of the backup environment makes the current backup system that requires manual configuration less efficient and more expensive. To enhance the efficacy and reduce the cost of the backup system management, a dynamic backup system is needed to effectively and automatically handle the configurations [17, 18, 19, 16].

The current backup system requires backup users to allocate storage capacity for every client. More often than not, they use intuition, inadequate default values, or a trial and error approach to allocate the storage capacity [37]. As they inaccurately predict the storage usage, the misconfiguration of the storages leads the storages to reach capacity. To effectively schedule the storage capacity, backup users need sufficient knowledge of the backup system and accurate prediction of the amount of the data generated and stored by clients. However, based on our analysis of the data protection reports in two years, from 2017 to 2018, the job failures due to the insufficient storage capacity is quite often, which is always one in the top two errors occurred at the backup system. It is hardly possible to manually predict the storage capacity accurately in a dynamic backup environment, since the storage capacity usage is effected by many factors, including changing data size, diverse data types, and various backup polices of different clients.

Prior research [38] presents the observations and statistics of the file systems and storage systems, which provides a guide to design a next generation backup storage system. However, they described and exhibited the characteristics of the backup workloads at a single point in time, which is not time series patterns that are necessary for the storage capacity forecasting. Three-year data protection reports collected from 40,000 enterprise systems were analyzed about the backup behavior of the real world systems [1]. Nevertheless, its observations were too general and lacked details, thus not useful for helping to explore a learning structure of backup storage capacity prediction. Chamness [22] uses statistical techniques and piece-wise linear regression to forecast the storage usage. Chamness' predictor works well on predicting the size of the storage in the near future. However, it ignores the mechanisms and hides the characteristics inside the backup system, so it fails to accurately capture the temporal details of the backup data. Vauhn [37] uses more advanced forecasting techniques including *K-means* and *ARIMA* to develop a storage capacity prediction tool called Soothsayer. But Soothsayer mainly targets on the backup storage served by a single client. Its learning structure heavily relies on the synthesis data and ignores the generation mechanism of the real-world backup data.

In this paper, we presented a study on a backup storage that served 174 clients in two years (2017 to 2018). The discussion of the analysis and statistics on the storage data is included and used for exploring a learning structure of forecasting storage capacity usage. We develop a novel forecasting structure consisting of the prediction of backup data size generated from clients and the associative deduplication ratios in terms of the day. Our forecasting structure has good robustness and accuracy for predicting a range of time when the storage will reach capacity. The main contributions of this paper are summarized as follows:

- Presenting the analysis and statistics of the real-world backup dataset which is used as a guide to help us develop a forecasting structure in backup storage systems.
- Designing a structure for predicting the amount of data generated by a set of clients by using statistical and time series learning techniques.
- Developing an effective Backup Frequency Filter (BFFilter) to predict if the client continue to be used in the future by obtaining the frequency of different job operations.
- Exploring learning algorithms for predicting the deduplication ratio based on the characteristics of the real-world dataset.
- Developing a forecasting structure for predicting capacity usage by a set of clients with different configurations in the future.

The structure of the paper is organized as follows. Section 4.2 describes the information and our study on the backup storage dataset. Section 4.3 and 4.4 present learning structures for predicting the amount of data generated by a set of clients and associative deduplication ratios. Section 4.5 verifies the forecasting structure using the enterprise system data. Section 4.6 compares the predictive performance of our forecasting structure to the related proposed forecasting tool.

4.2 Backup storage data

This section presents the information of the backup storage data collected from the real-world system. The information describes 263 clients's backup behavior and their effects on a backup storage in two years (2017 to 2018). We performed an analysis on the information and used it to guide in creating a forecasting structure.

4.2.1 Dataset information

The dataset depicts 22,526 jobs scheduled to the storage in chronological order. Each job is characterized by six attributes which are *jobId, day, clientId, dataType, scheduleType, jobSize* and *dedupRatio*. More details about some attributes are as follows.

dataType. Each job scheduled by the client in the dataset belongs to one of three data types: *Devices data, Operating system*, and *Virtual Machine*. These three kinds of data are generated in the same backup domain which consisted of three tiers, clients, media servers, and backup storages. *Device data* refers to the data of the raw devices are protected. *Operating system* means operating system data such as Windows or UNIX are protected. *Virtual Machine* protects VM images from virtual environment such as VMware or Hyper-V. The descriptions about these three data types in the dataset are summarized in Table 4.1.

scheduleType. The dataset we studied has two schedule types, *full* and *incremental*, used for protecting those three kinds of data. While *full* is to backup the whole volume of the data, *incremental* only backups the changed data from last (full or incre-nental) backup [6]. The dataset shows 80% of jobs are scheduled by *incremental* and the remaining 20% are scheduled by *full*.

dedupRatio. Deduplication is one of commonly used backup techniques to reduce the redundancy of the backup data for saving the storage space. Deduplication ratio in terms of a job is a metric to measure the extent of removing the redundancy, which is quantified by the *jobSize* and the size after duplicate elimination, *sizeAfterDedup* as shown in Equation (4.1) [39].

$$dedupRatioPerJob = \frac{jobSize - sizeAfterDedup}{jobSize} \tag{4.1}$$

4.2.2 Data types of each client

Clients are the sources that generate backup data. In our dataset, we found each client only protects single data type and has its own configuration of *full* and *incremental* frequency. The particular data type and backup configuration make each client have its own backup patterns. Therefore, it is useful to study the backup behavior and develop the predictor of storage capacity usage through segmenting clients.

4.2.3 Data size of different schedule types

The amount of data generated by clients in the backup system is largely stationary at least in a shorter period [22], which is verified in our dataset. The largely stationary backup data makes the capacity usage in the backup system predictable. We studied the backup behavior of 273 clients. Three examples with different data types are shown in Figure 4.1. In most clients protecting *Devices* data and *Virtual Machine*, the amount of data in *full* is larger than *incremental* as shown in Figure 4.1a and 4.1c. However, this is not consistently true in some clients protecting *Operating system* shown in Figure 4.1b. The variation of the data size in Operating systems is also larger than the other two. The incremental size is similar to its previous full size at around day 170. Therefore, we cannot assume the size of full is always larger than *incremental* in a client.

4.2.4 Backup frequency of each client

Backup frequency is one of configuration options in the backup system and refers to how often to schedule *full* and *incremental* backups. Such configuration is handled by backup admins and is consistently stable, which is shown in Figure 4.1. This information gives us a clue to predict if the client is still "alive" and will be used in the near future. More details about this will be introduced in Section 4.3.

4.2.5 Deduplication ratio and job size

The amount of deduplication ratio depends on the redundancy of the data and the deduplication algorithm that is used. Soothsayer [37] employs time series forecasting algorithm to predict the deduplication ratio for *full* and *incremental* backups respectively. Time series forecasting algorithms only captures the temporal structure of the deduplication ratio, but it ignores the internal mechanism of deduplication. Soothsayer [37] assumes deduplication ratios follow different patterns based on different schedule types.

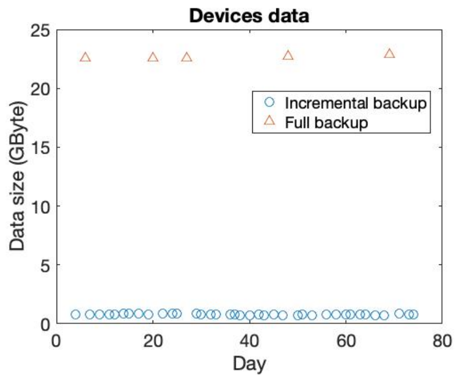
Data type	**Data contents**
Devices data	Data in hard disks or tapes
Operating system	Windows Volumes
Virtual Machine	VM image in VMware

Table 4.1: Data contents protected in the dataset

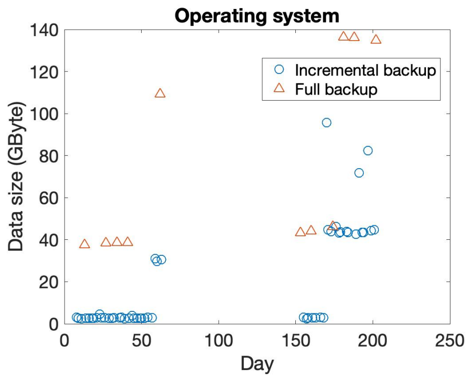
Based on this assumption, it segments the dataset to two schedule types *full* and *incremental* for predicting the deduplication ratios. However, the assumption is not true. A counterexample is shown in Figure 4.2c. The deduplication ratios of *incremental* are similar to *full* at around 600 GByte. Based on our study with the dataset, the deduplication ratio in the backup system is high and around 87%. Intuitively, in a client whose backup jobs share the same configurations, backup data has higher redundancy as the size is increased. Examples of three clients are shown in Figure 4.2. The mean values of deduplication ratio of larger data size are higher than those of smaller data size. Configurations and deduplication algorithm may be different between clients, thus, each client has its own data size range and associative deduplication ratios, which are shown in Figure 4.2. Therefore, it is necessary to predict the deduplication ratio by segmenting clients. More details about the prediction of deduplication ratio will be introduced in Section 4.4.

4.3 Prediction of data Size

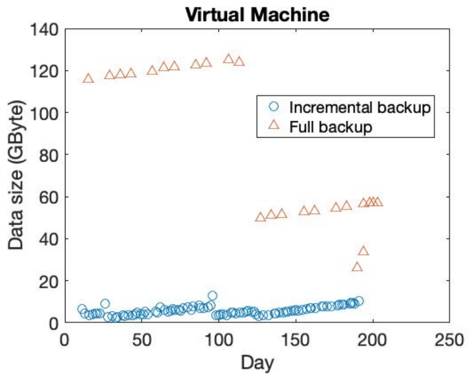
To predict the capacity usage served by a set of clients, we create a forecasting structure which consists of the prediction of backup data size generated from clients and the associative deduplication ratios. In this section, we present the structure to predict the range of the data size from multiple clients, which is shown in Figure 4.4. As aforementioned in Section 4.2, segmentation of clients is necessary due to the different backup behaviors of each client. The learning structure of the data size for each client is shown in Figure 4.4. The input of the learning structure is a list of historical backup storage data of a client and each data includes all attributes that are introduced in Section 4.2.1. The output after the *Stochastic time series forecasting* is a set of predictions on future data size of a client.



(a) The size of Devices data



(b) The size of Operating system



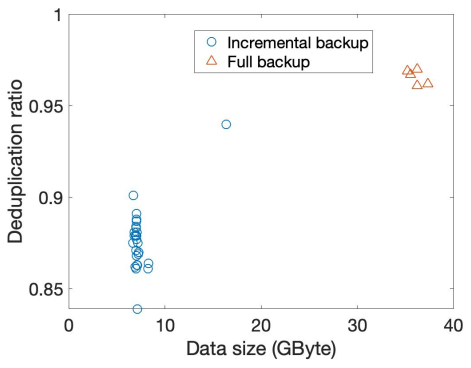
(c) The size of Virtual Machine

Figure 4.1: Data size of different data types in *full* and *incremental*

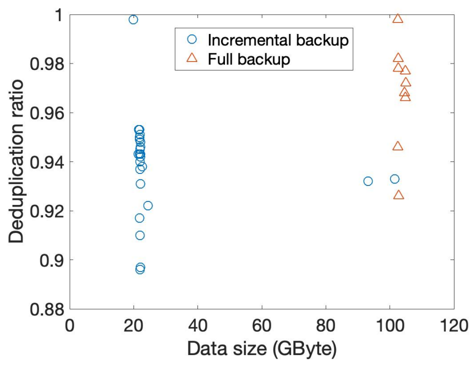
Merge. Some clients scheduled backup jobs multiple times per day, but the forecasting structure in the paper aims to predict the backup storage capacity usage in terms of the day. Therefore, we unified the dataset by employing the merge function to obtain the daily job sizes for every client.

BFFilter. As the introduction in Section 4.2.4, the configuration of the backup frequency is statically configured by the backup admins. We can see the intervals between *full* and *incremental* are largely stationary at least in a shorter period in Figure 4.1. Therefore, the information about the backup frequency is saved for the prediction of storage capacity usage.

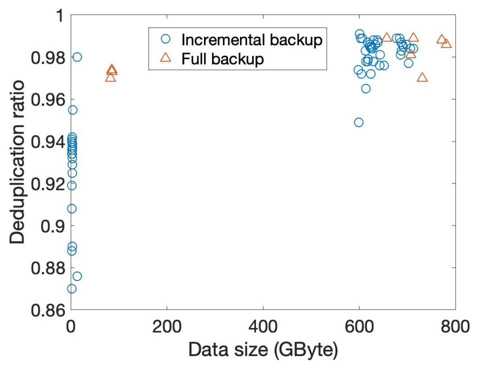
Live check. Based on our observation of the dataset, we found a case that some clients were removed from the backup domain and would not be used. Such case is well



(a) Client A: 8 to 39 GByte



(b) Client B: 20 to 100 GByte



(c) Client C: 0 to 800 GByte

Figure 4.2: The correlation between deduplication ratio and data size

estimated through the information of backup frequency captured by the BFFilter. Two examples are shown in Figure 4.5. They used the data from the previous 7 days to predict if the client will continue to be used in the near future. The historical backup intervals are obtained, which are [2, 1, 1, 1, 2] in Figure 4.5a. The check point interval is the day interval between the check point and the last backup, which is 1. The live check is employed by comparing the check point interval to the maximum value of the historical backup intervals. The larger or equivalent value of the check point interval predicts the client as still alive, as shown in Figure 4.5a. Otherwise, the client is predicted as "dead", as shown in Figure 4.5b.

Stochastic time series forecasting. Instead of only providing the prediction of storage capacity usage at the single point of the time, we aim to develop a forecasting

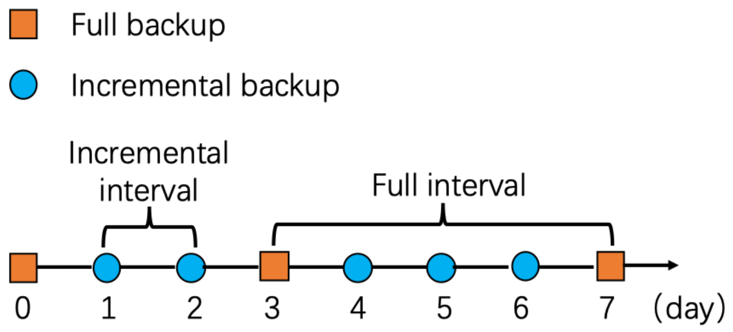


Figure 4.3: Incremental and full intervals

structure to estimate the range of time when the storage usage reaches capacity. Similarly to Soothsayer [37], the stochastic scenario is introduced to the training dataset. Based on an analysis of the data, we found data generated by the client can be simply divided into two groups based on size, which is shown in Figure 4.2. However, as shown in Figure 4.2c, the amount of data cannot be always simply separated by *full* and *incremental*. Therefore, a commonly used clustering algorithm, *K-means*, is employed to segment the data into two groups.

Two groups of *jobSize* are **fitted to the distribution** respectively. An automatic distribution selection scenario is used to fit the data to an optimal distribution. The optional distributions are normal, lognormal, gamma, expenential, and Weibull. The distribution with highest significant value by *chi-square Goodness-of-Fit test* is selected. Two new groups of *jobSize* are **reconstructed** by their selected distributions as shown in Figure 4.6.

The reconstructed dataset is as the training input of the common used time series forecasting algorithm, *AutoRegressive Integrated Moving Average* (ARIMA) [40]. The output of the ARIMA model is one set of the future data size.

Cumulate. To evaluate the learning structure in Figure 4.4 for the prediction of storage capacity usage, a **cumulate** function is used to obtain the amount of data transferred to the backup storage. The repetitive employments of aforementioned approaches give us a set of predictions of a client's storage capacity usage.

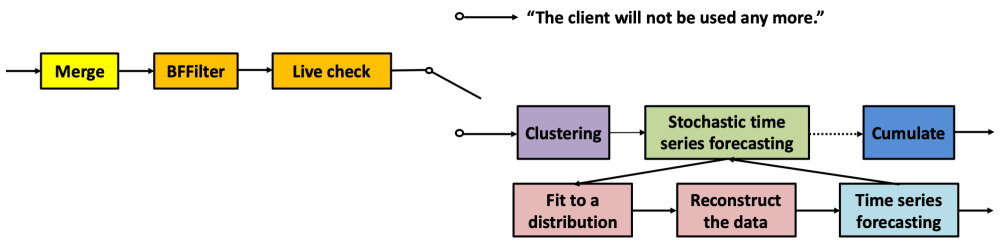
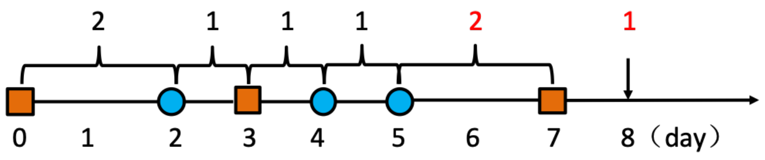
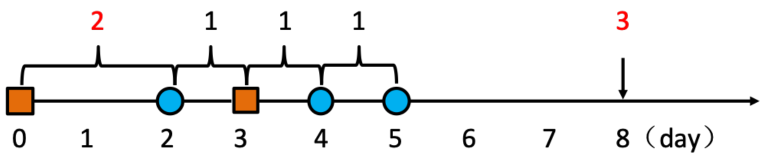


Figure 4.4: The structure for forecasting cumulated size before duplicate elimination in terms of day



(a) LiveCheck = 1



(b) LiveCheck = 0

Figure 4.5: Employ live check by using the information of backup frequency captured by the BFFilter

4.4 Prediction of deduplication ratio

For a client, the duplication ratio can be predicted from the job size, as we studied in Section 4.2.5. The prediction of deduplication ratios is used to forecast the backup storage usage in terms of the day. Therefore, we transformed the deduplication ratio per job as quantified by Equation (4.1) to the deduplication ratio per day described in Equation (4.2). To obtain the $dedupRatioPerDay_i$ at specific day i, the data sizes in terms of the day i before and after deduplication, $sizePerDay_i$ and $sizeAfterDedupPerDay_i$ need to be computed by the sum of all the data sizes at the same day as shown in Equation (4.3) and (4.4). The $sizeAfterDedupPerJob$ can be calculated using the $jobSize$ and

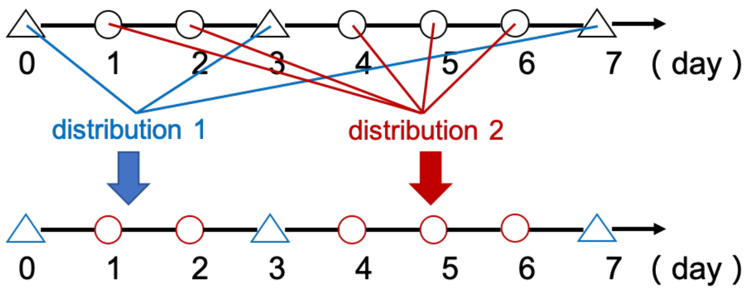


Figure 4.6: Reconstruct the data from the selected distributions

dedupRatioPerJob provided in the dataset, which is shown in Equation 4.5.

$$dedupRatioPerDay_i = 1 - \frac{sizeAfterDedupPerDay_i}{sizePerDay_i} \tag{4.2}$$

$$sizePerDay_i = \sum_{\forall jobSize\ at\ day\ i} jobSize \tag{4.3}$$

$$sizeAfterDedupPerDay_i = \sum_{\forall jobSize\ at\ day\ i} sizeAfterDedupPerJob \tag{4.4}$$

$$sizeAfterDedupPerJob = jobSize * (1 - dedupRatioPerJob) \tag{4.5}$$

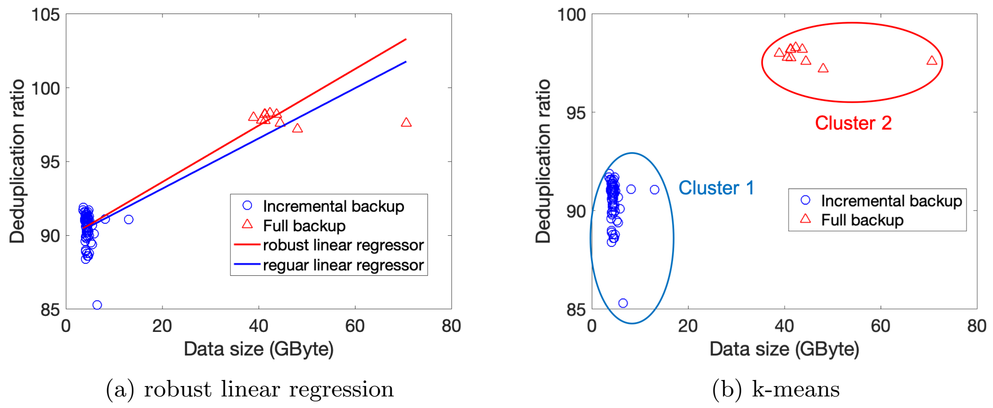
We implement the prediction of deduplication ratio in terms of the day, *dedupRatioPerDay*, for a client by using *sizePerDay*. Two optional learning algorithms, *robust linear regression* and *K-means* are discussed as follows.

4.4.1 Robust linear regression

Pearson correlation [41] is employed to verify the linear relationship between the job size and associative deduplication ratio. It shows that the deduplication ratio is positively correlated with the job size in a client. Therefore, we applied the *robust linear regression* to predict the deduplication ratios as shown in Figure 4.7a. The weights in the linear model are optimized by *iteratively reweighted least-squares* [42], which reduces the effects by the outlier data. Also, we found the deduplication ratios in the backup system are generally high and the variance is small. Therefore, we add a constraint that the predicted $dedupRatioPerDay$ is controlled within the maximum and minimum value of the historical values.

4.4.2 K-means

Based on our observation of the dataset as shown in Figure 4.2, the deduplications ratios are divided to two groups in terms of the data size. The small variation of the deduplication ratios of each group makes it possible to have a good prediction by assigning the mean values of the grouping data. Hence, we also employ the commonly used clustering algorithm, *K-means*, to assign two predicted deduplication ratios to two groups of data based on the data size as shown in Figure 4.7b. At the training phase, each client learns two clustering values of deduplication ratio for two clustering data based on data size. At the test time, the deduplication ratio is predicted as the clustering value whose data size is closest to.



(a) robust linear regression (b) k-means

Figure 4.7: Prediction of deduplication ratios using *robust linear regression* and *k-means*

4.4.3 Implementation of forecasting storage capacity usage

The forecasting structure is used to predict the backup storage capacity usage by the predictor of data size (see Section 4.3) and the predictor of deduplication ratio introduced above. The forecasting structure is shown in Figure 4.8. The deduplication ratio and backup size predictors of each client are obtained by training the historical data. The future values of $sizePerDay$ is predicted for every client, subsequently use them

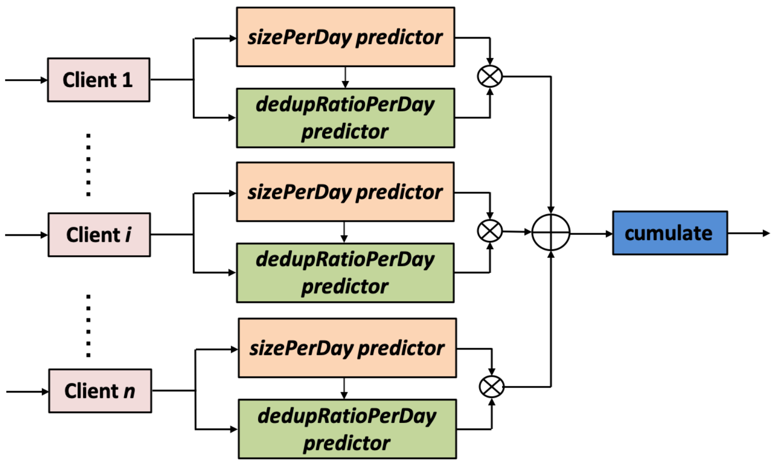


Figure 4.8: The forecasting structure for the prediction of backup storage capacity usage

to estimate the values of $dedupRatioPerDay$. The values of $sizeAfterDedupPerDay$ of each client are obtained by applying Equation (4.4). The storage capacity usage is then forecasted by the sum and cumulation of all clients' predicted data size after duplicate elimination. Since the stochastic scenario is introduced to the size prediction, one iteration of executing the forecasting structure gives one set of storage capacity usage prediction. The distribution of the storage capacity usage in terms of the day is obtained by multiple iterations.

4.5 Evaluate the forecasting structure

This section presents the evaluation of the forecasting structure using the real-world data collected by the enterprise backup system. In order to compare the predictive performance of our forecasting structure with the proposed tool, Soothsayer [37], we also divided the dataset into two parts: 70% dataset (from day 1 to day 142) used to train the forecasting structure and the other 30% dataset (from day 143 to day 200) used to test the predictive performance.

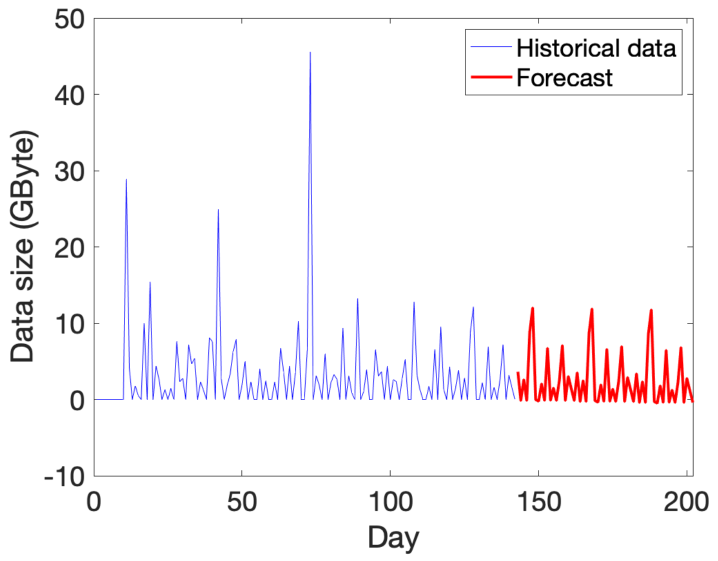


Figure 4.9: The prediction of data size generated by a client using ARIMA

4.5.1 Evaluate the prediction of data size

This section presents the prediction of cumulated data size using the learning structure in Figure 4.4. First, we evaluated the performance of the **live check** using the backup frequencies captured from **BFFilter**. The confusion matrix is used to present the results, which are shown in Table 4.2. The results show that the **live check** achieves high accuracy (95%) at predicting if the client is still used. Some clients stop being used right after the checkpoint, which results in the mis-prediction. But such case is rare and only occurred in 2% of all clients. The **live check** fails to handle the case where clients stopping to be used for a period and then are recovered as shown in Figure 4.1b, however, such case only accounts for 3% of all clients.

Second, we tested the predictive ability of the ARIMA model. A good time series learning model for backup systems should be capable of capturing the temporal structure of the historical data including the data size and frequency of scheduling *full* and *incremental*, as well as give a good prediction of the near future. One example of predicting the amount of data generated by a client is shown in Figure 4.9. We can see the ARIMA model gives a good prediction from a dataset with seasonalities which is

	predicted:	**predicted:**
263 clients	**alive**	**dead**
Actual: alive	197	8
Actual dead	5	53

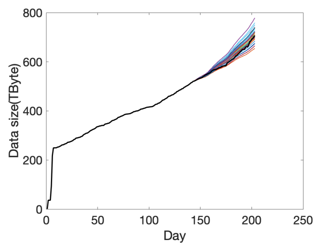
Table 4.2: Evaluation of the live check

a data pattern consisting of changes that repeat over a time period. The results show that the data size and frequency of *full* & *incremental* are well predicted by the ARIMA model using the historical data.

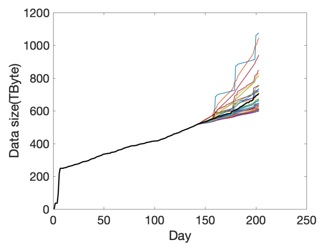
Third, the predictions with different distribution selection scenarios are shown in Figure 4.10. The black line describes the actual meastured data and the others are predictions of 50 iterations. We can see the Lognormal distribution gives unstable prediction and whose performance is worse than the others. While Normal distribution produces over-prediction, Gamma, Exponential, and Weibull provide under-prediction most of the time (from day 143 to day 200). In contrast, the automatic distribution selection offers stable and more accurate prediction. Hence, the automatic distribution selection scenario succeeds in fitting the data with different policies to respective optimal distributions.

4.5.2 Evaluate the prediction of deduplication rate

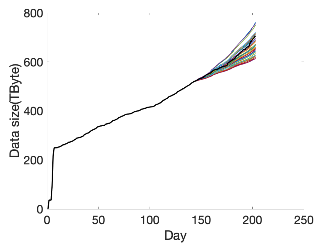
This section evaluates the performance of *robust linear regression* and *K-means* for predicting the deduplication ratios per day. Combined with data size prediction, deduplication ratio prediction is used to forecast the storage capacity usage. We present the data size after the deduplication predicted from two learning algorithms and its cumulated format (storage capacity usage) in Figure 4.11. From Figure 4.11a, we can see the data size whose deduplication ratio is predicted by *robust linear regression* is closest to the test data. That means that the deduplication ratios are well predicted. However, the prediction is a little over-predicted, which consequently results in the storage capacity usage being a little under-predicted as shown in Figure 4.11b. In contrast,



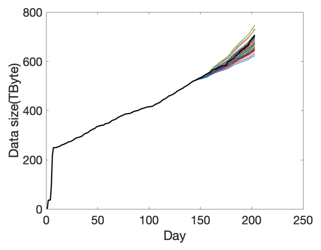
(a) Reconstruct the data from Normal distribution



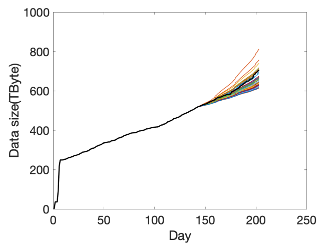
(b) Reconstruct the data from Lognormal distribution



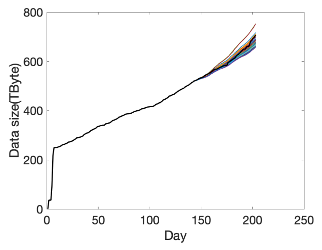
(c) Reconstruct the data from Gamma distribution



(d) Reconstruct the data from Expoential distribution



(e) Reconstruct the data from Weibull distribution



(f) Automatic distribution selection

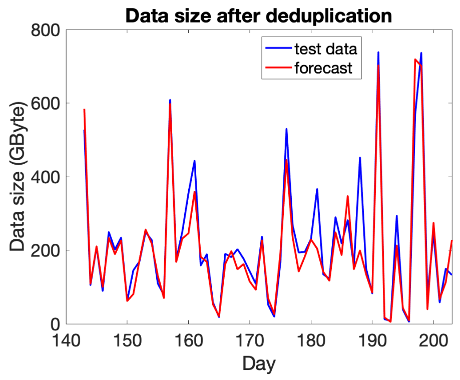
Figure 4.10: Evaluate the prediction of data size by *stochastic ARIMA*

the deduplication ratios predicted by *K-means* are better than those by *robust linear regression*, as shown in Figure 4.11c and 4.11d.

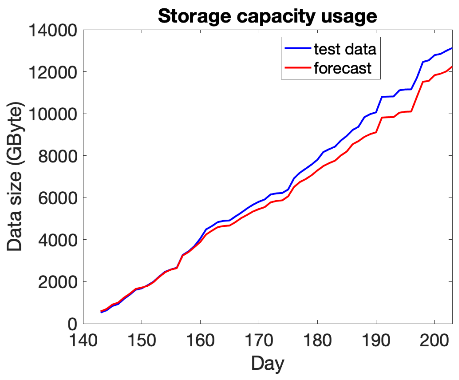
A Linear regression model is more effective when applied to a continues dataset. However, for backup system data, we found that the backup data in a client is well divided to two clusters based on size. The size of two clusters are significantly different. The clustered data with larger size has consistently larger deduplication ratios, which is introduced in Section 4.2.5. We also found the variance in deduplication ratios for each clustered data is small. Therefore, the clustering algorithm, *K-means* gains more accuracy for the deduplication ratio prediction.

4.5.3 Evaluate the storage capacity usage forecasting

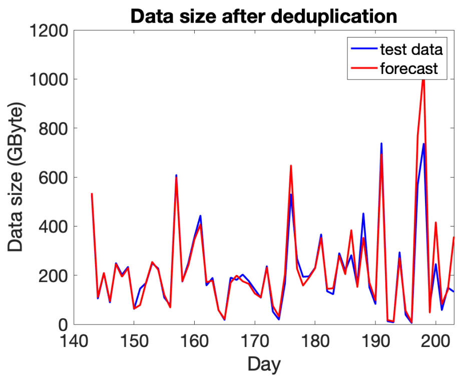
This section presents the performance of the forecasting structure in Figure 4.8 for predicting the day of reaching to various storage capacities. A set of predictions using different deduplication learning algorithms and their histogram representations are



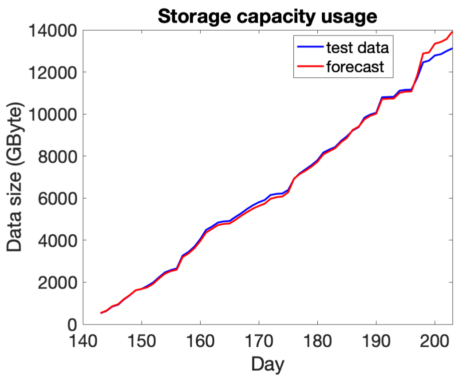
(a) Deduplicated data size using robust linear regression



(b) Storage capacity usage using robust linear regression



(c) Deduplicated data size using K-means



(d) Storage capacity usage using K-means

Figure 4.11: Evaluation of deduplication ratio prediction by using *robust linear regression* and *K-means*

shown in Figure 4.12 and 4.13. We varied the storage capacity to evaluate the predictive performance of different days to reach the capacity. The results show that the variance of predictions is close to zero in the near future and then increases over time. The forecasting structure uses the historical data to provide determined prediction at the beginning. As the increase of time, more random factors effect the prediction. The predictions by the forecasting structure are more stochastic as the growth of the time.

The associative predictions are fitted to histograms for visualizing the predictive performance. The perfect predictions shown in the histogram are normally distributed around the actual time of reaching to the capacity. Based on this, we can see the results

Storage Capacity	32TB	34TB	36TB	38TB
Actual time	157	163	176	185
Linear model				
Mean (error)	158 (1)	167 (4)	177 (1)	186 (1)
95% Interval	158 - 159	167 - 168	177 - 178	185 - 187
K-means model				
Mean (error)	157 (0)	166 (3)	176 (0)	185 (1)
95% Interval	157 - 158	166 - 167	176 - 177	185 - 187
Soothsayer [37]				
Mean (error)	148 (9)	150 (13)	153 (23)	155 (30)
95% Interval	147 - 148	149 - 151	152 - 154	154 - 156

Table 4.3: Results of forecasting structure with different learning algorithms. (The numbers without unit represent days)

of two deduplication learning algorithms give good predictions. The quantitive measurements are shown in Table 4.3. From the results, we can see the error of predictions are controlled within 4 days, thus, our forecasting structure succeeds in providing a good and robust prediction of the storage capacity usage.

Even though the stochastic scenario is applied to our forecasting structure, the predictions' variation is small and provide good predictive performance. That is represented by the small confidential intervals whose ranges are less than 3 days and errors are within 4 days. The small confidential intervals result from the largely stationary backup data and from suitable distributions being selected for reconstructing training data.

Compared to the forecasting structure using *robust linear regression*, the *K-means* gains better predictive accuracy in the dataset. Particularly, the forecasting structure with *K-means* for deduplication ratio prediction achieves zero error when the storage capacities are 32TB and 36TB. The forecasting structure with *K-means* for deduplication ratio prediction has better prediction of the time reach capacity due to the small variance of deduplication ratios for two clustered data, which matches our evaluation in Section 4.5.2.

4.6 Related work

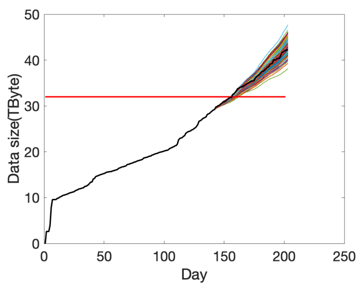
Vauhn [37] developed a storage capacity forecasting tool called Soothsayer for predicting a range of time for when the storage reaches capacity. The forecasted backup sizes and deduplication ratios are generated from following procedures:

1. Split the data of each client based on different schedule types which are *full* and *incremental* data.
2. Apply *k-means* to group the incremental size into three clusters and fit each clustered size to the most suitable distribution. Reconstruct the incremental size from the distribution.
3. Find the optimal distributions for full backup size. Reconstruct full backup sizes from these distributions.
4. Fit deduplication ratios of *full* and *incremental* data to their optimal distributions. Reconstruct deduplication ratios from these distributions.
5. Employ the ARIMA algorithm to the reconstructed data and obtain the predicted backup sizes and deduplication ratios.
6. Repeat the above steps for 100 iterations to obtain a set of predictions of the times when capacity is reached.

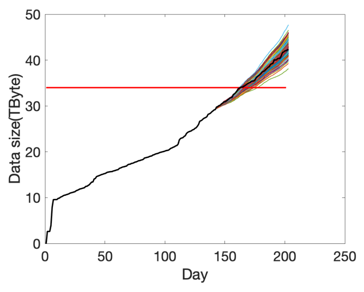
Soothsayer is mainly built on the top of the synthesis data, which leads to characteristics of the backup data being ignored. From the results shown in Table 4.3, Soothsayer always over-predict the capacity usage, which results in the under-prediction of the day to reach the capacity. The error is 9 days when the actual time is 157, and it increases over time. The error falls to 30 when the actual time is 185. One of reasons for the over-prediction of the capacity usage is that Soothsayer fails to provide a good prediction of which clients will stop being used, which is improved by **live check** in our forecasting structure as introduced in Section 4.3. Another is that it does not make sense to use a time series learning algorithm to predict the deduplication rates, which ignores the characteristics of deduplication ratio as introduced in Section 4.2.5. Finally, it will not always provide benefit when segmenting the training data into *full* and *incremental* to forecast the deduplication ratios, which is analyzed in Section 4.2.5. Our forecasting structure employs *K-means* to segment dataset in terms of the size and provides more accurate deduplication ratio prediction, which is presented in Section 4.5.2.

4.7 Conclusion

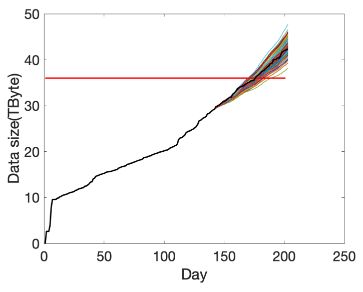
Insufficient storage ranks as the number two most common cause of backup failures. This issue is becoming increasingly common as the scale of the backup system grows. However, there is very little research into providing giving an efficient solution to mitigate this issue. Our paper presents a study on real-world data collected from enterprise backup systems and uses the characteristics of the data to create a forecasting structure. The forecasting structure has better and more robust prediction of backup data sizes generated by a number of clients, deduplication ratios, and the day of reaching capacity, compared to Soothsayer [37]. The predictions made by the forecasting structure can be used to guide backup admins to efficiently allocate the backup storage capacity, set the retention window for backup images and design next generation backup systems.



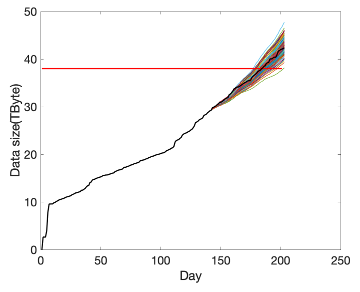
(a) Capacity = 32TByte



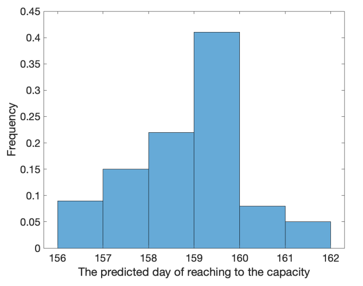
(b) Capacity = 34TByte



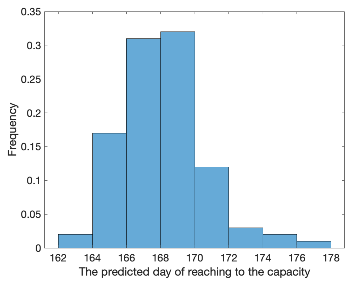
(c) Capacity = 36TByte



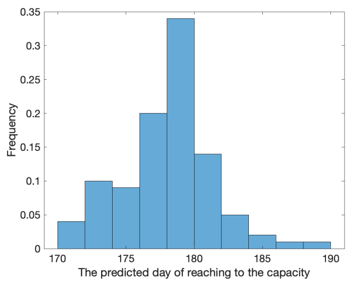
(d) Capacity = 38TByte



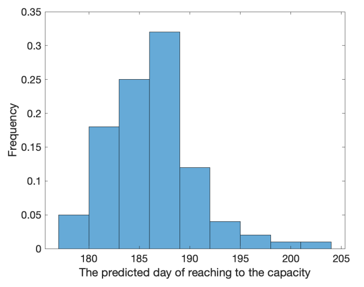
(e) Predicted day (error) of reaching the capacity = 158 (1)



(f) Predicted day (error) of reaching the capacity = 167 (4)

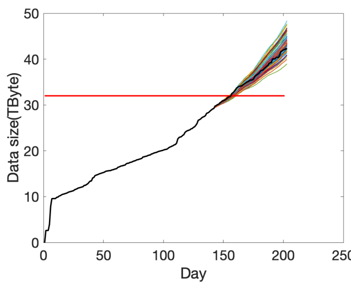


(g) Predicted day (error) of reaching the capacity = 177 (1)

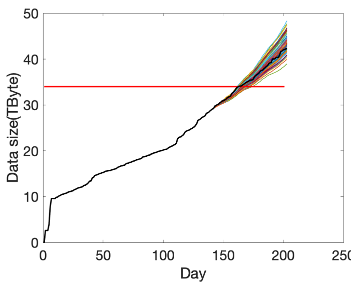


(h) Predicted day (error) of reaching the capacity = 186 (1)

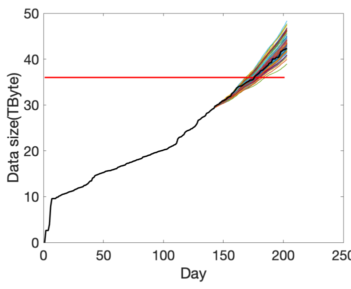
Figure 4.12: Predict the day of reaching different capacities using robust linear regression when predicting the deduplication ratio



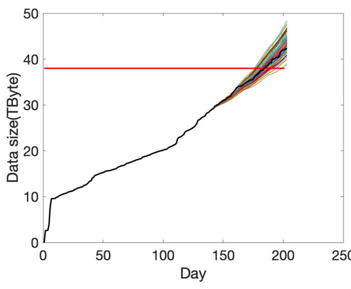
(a) Capacity = 32TByte



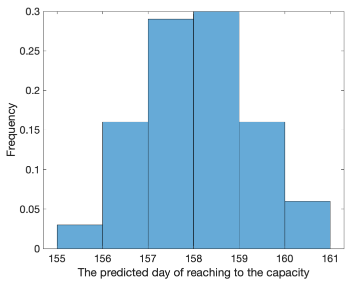
(b) Capacity = 34TByte



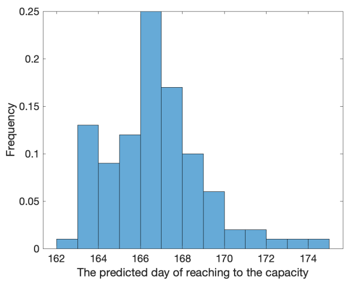
(c) Capacity = 36TByte



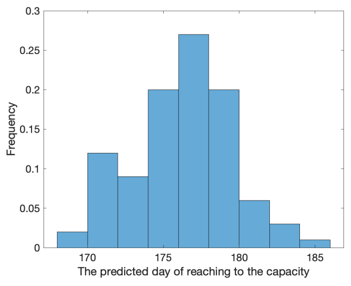
(d) Capacity = 38TByte



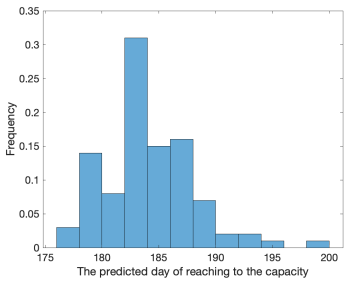
(e) Predicted day (error) of reaching the capacity = 157 (0)



(f) Predicted day (error) of reaching the capacity = 166 (3)



(g) Predicted day (error) of reaching the capacity = 176 (0)



(h) Predicted day (error) of reaching the capacity = 184 (1)

Figure 4.13: Predict the day of reaching different capacities using K-means when predicting the deduplication ratio

Chapter 5

Prefetch Backup Content Correlated Fingerprints

Chapter 3 presented the high-efficiency deduplication for the existing clients. This chapter is going to introduce the work on applying machine learning techniques to improve the prefetching performance of deduplication for newly created clients [43].

5.1 Introduction

The fast development of artificial intelligence, deep learning, IoT, and 5G has resulted in an explosive growth of the data that are required to be stored and protected. It is becoming extremely challenging for storage systems to handle the task of storing and protecting such a large amount of data with shrinking storage budgets. Combined with the increasing service level requested, it is now significant and necessary to improve the efficiency of storage systems using data reduction techniques. Many studies have discovered that there are high levels of redundancy occurring in the data in storage systems such as backup systems [38], archival systems [9, 26, 44, 45, 46, 47, 25, 48], primary storage systems [49, 50, 51, 52, 53, 28, 54, 55, 56, 57], and data centers [58].

The client-policy infrastructure enhances the content correlation among backups from the same client and cross clients. Based on our observation of the backup system, the data content from a client is not changed significantly at least in a short time period. The dedupe system achieves a good dedupe rate by prefetching the backups from the same client as introduced in Chapter 3. However, for the new backup from the newly created client, there are no historical backups so that the prefetching algorithm has no reference basis to perform effective fingerprint prefetching from the other clients or policies. The commonly used generic prefetching algorithm for the container-fashion backup system is Progressive Sampling [9]. It randomly samples a fraction of the fingerprints from each container to the memory and count the number of hits. The fingerprints in the containers with high number of hits are all prefetched to the memory. The prefetching performance of this approach highly relies on the number and the quality of the sampled fingerprints. The prefetching performance will be severely degraded when more backup data are accumulated, which is resulted from the less representative fingerprints are sampled in the same memory.

Machine learning techniques have already widely implemented on the various fields. Particularly in backup systems, different learning techniques were applied to dynamic backup scheduling [16], backup storage capacity forecasting [36], and training data deduplication [59] for optimizing the system performance. We designed a smart prefetching algorithm called prefetching backup content correlated fingerprint (PBCCF). Combined with machine learning and statistical techniques, the PBCCF are proposed to optimize the dedupe rates of the backups from the newly created client and maintain high dedupe performance by discovering and prefetching the highly correlated fingerprints from the other clients. During the design, we considered minimizing any additional time and storage space overhead incurred with finding the best possibility for matches. The PBCCF uses lightweight machine learning, density-based clustering and kernel density estimation, to discover and capture the patterns and features of incoming backup data and historical data. The patterns and features are saved for assistance in prefetching. Rather than the fingerprints, the input to the PBCCF for obtaining patterns and features is high-level metadata, which are the container usage data that consist of container IDs (logical location) and the amount of data represented by the fingerprints. The main contributions of this paper are as follows:

- Discovering the backup content correlations among the backups from different clients using a real-wold backup dataset collected from an enterprise backup system;

- Employing light-weight machine learning techniques to discover the patterns and generalize the features of the backups.
- Proposing an efficient and practical way to measure the similarity between backups using their container usage data.
- Designing a prefetching approach, PBCCF to identify the highly correlated backups that have the most similar container usage.
- Introducing an efficient and effective approach to determine the highly correlated fingerprints from the identified backups by the PBCCF.
- Testing the prefetching performance of PBCCF and comparing to the existing prefetching algorithm in container-fashion dedupe system, Progressive Sampling.

The remainder of this paper is organized as follows: Section 5.2 discusses the study of the backup content correlation using the real-world dataset. Section 5.3 presents the design of the PBCCF. Finally, the evaluation is discussed in Section 5.4 and conclusions are drawn in Section 5.5.

5.2 Backup Content Correlation

The duplicates exist among different clients due to the behavior of backup user on the client-policy backup system. For example, the emails with files attachments are sent to different clients, the same image is applied to create virtual machine of multiple clients, the clients are running on the same platform. These are very common behavior and frequently occur at the real-world backup system, which results in the backup content correlation among clients.

This section explores the backup content correlation among backups from different clients protected by client-policy backup systems. We collected real-world data from a production system that protected different types of data that includes files, virtual machine and operating systems. We focused on the five policies and 55 clients from January to July in 2019. There are around total 250 total full backups and the size of a backup is from around 5 to 500 GB. The data is consisted of policy ID, client ID, backup ID, fingerprint of each segment, size of each segment, and container ID of each segment.

New clients are created over time for various reasons, such as new hosts joining the system or the replacement of the old machines. We used the dataset to study the correlation between clients. Dedupe rate is a commonly used metric to measure the extent of duplicates removal. The two metrics of internal dedupe rate, *internal*, and cross-dedupe rate, *cross*, are used to quantify two kinds of redundancy. Here, *internal* quantifies the redundancy within a $backup_i$ and is formulated as the total size of the backup, *total*, and the size of the unique segments that form the backup, *unique*, which is shown in Equation 5.1. In addition, $cross_{ij}$ specifies the total redundancy of backup_i that can be removed when prefetching backup_j as the base for dedupe and is quantified as shown in Equation 5.2. The $unique_{ij}$ specifies the $unique_i$ that excludes the size of segments from the backup_j. By this definition, the calculation of $cross_{ij}$ includes the internal dedupe. Therefore, the minimum of $cross_{ij}$ equals the $internal_i$, which means the prefetched backup_j does not offer any effective fingerprints for dedupe.

$$internal_i = \frac{total_i - unique_i}{total_i} \times 100\% \tag{5.1}$$

$$cross_{ij} = \frac{total_i - unique_{ij}}{total_i} \times 100\% \tag{5.2}$$

The other two metrics, average internal dedupe rate, $\overline{internal}$, and average cross-dedupe rate, $\overline{cross}$, are used to study the correlation between clients. Since the dedupe rate is a rate metric, we applied a harmonic mean to calculate the average values of two dedupe rates of n clients, which are shown in Equations 5.3 and 5.4.

$$\overline{internal} = \frac{n}{\sum_{i=1}^{n} \frac{1}{internal_i}} \tag{5.3}$$

$$\overline{cross} = \frac{n}{\sum_{i=1}^{n} \frac{1}{cross_{ij}}} \tag{5.4}$$

The average internal dedupe rate of all clients was 10.6%. We also calculated average cross-dedupe rate of all pairs of clients, which reached 30.0% when the best base backup can always be identified and prefetched for the given backup of a client. The worst average cross-dedupe rate was as low as the average internal dedupe rate, 10.6%. That means the dedupe rate will have no improvement if prefetching uncorrelated backups for dedupe, while the dedupe rate will have around a 3x improvement if a prefetching algorithm succeeds at identifying highly correlated backups (most similar backups) from the other clients and fetching their fingerprints.

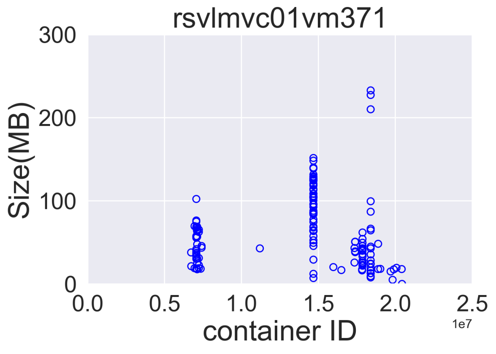
However, it is costly to calculate the similarity between two backups in an enterprise backup system by comparing their fingerprints one by one due to the extremely large number of backups. Based on the dedupe nature, only the unique contents will be saved to the containers, while the duplicates will maintain the reference pointers to the existing data in the container. Therefore, the similarity between the backups in terms of their data contents can be approximated by the container usage similarity. We further investigated such a correlation by calculating the cross-dedupe rates among backups from different clients and visualizing their container usage data. The container usage data includes the container ID and the size of data contents saved in the container for that backup. An example of cross-dedupe rates between a backup from the client 371 and the other clients (175, 404, and 340) is shown in Table 5.1, and their container usages in 2D plots are shown in Figure 5.1. The example shows that $cross_{371,404}$ yielded the highest dedupe rate, while the container usage of client 404 had the most similarity to the client 371. Therefore, the dedupe rate can be improved when prefetching the backups with highest container usage similarity.

Backup	**Base**	**Cross-dedupe rate**
Client371	Client175	$cross_{371,175}$ =3.2%
	Client404	$cross_{371,404}$ =47.9%
	Client340	$cross_{371,340}$ =7.5%

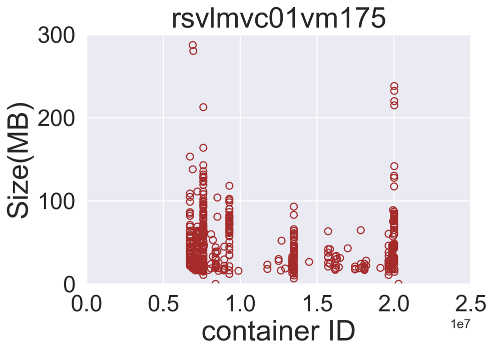
Table 5.1: Cross-dedupes comparing client175, client404, and client340.

5.3 PBCCF

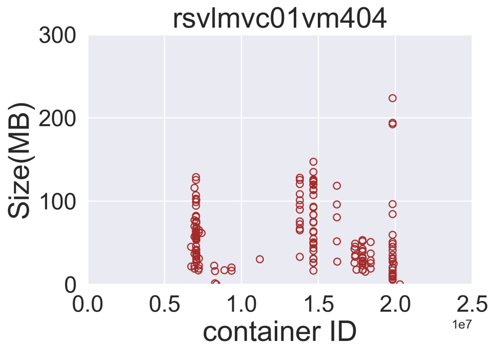
This section firstly introduces the architecture of implementing PBCCF on the backup system. The container usage pattern mining then presents how to discover the patterns and generalize the features only using container usage data by applying machine learning



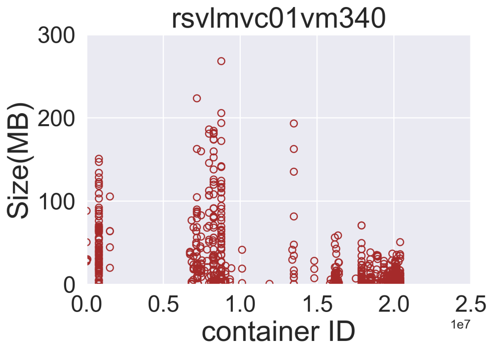
(a) The backup from client371



(b) The backup from client175



(c) The backup from client404



(d) The backup from client340

Figure 5.1: Container usage of four clients: 371, 175, 404, and 340

and statistical techniques. Finally, an efficient and effective approach is introduced to measure the container usage similarity between backups.

The Architecture of PBCCF

The architecture and approaches to perform PBCCF are shown in Figure 5.2. Most modern backup systems [10, 11, 12, 13, 14, 15] use a client-server infrastructure to implement data projection. Backups are generated from the client side and then stored on the server side in scalable storage.

On the server side, a number of backups exist, but not all backups can provide effective fingerprints for the new coming backup. The backups from other policies have very low content correlation and almost have no benefit for dedupe. Therefore, we keep the fingerprints of the backup from the same client (referred to as the candidate backups) in the Fingerprint Catalog maintained in fast storages, such as solid-state drives. A summary process is run on the Fingerprint Catalog to obtain the container usage data of each backup, as visualized in Figure 5.1. The container usage data of each backup is an input to the learning component called Container Usage Pattern Mining (CUPM) for discovering the patterns and generalizing the features of each backup. The features of pattern data are saved for the characteristics of the backup.

On the client side, the PBCCF is activated when the initial backup of a new client is added (referred to as *Bnew* in Figure 5.2). The initial backup is segmented and then hashed to fingerprints. A small fraction of the fingerprints from the beginning of the initial backup (referred to as a lookup window) are looked up in the Fingerprint Catalog for dedupe. The container usage of the lookup window is used to approximate the container usage of the whole backup. The CUPM is applied to the container usage data of the lookup window for acquiring container usage patterns. The container usage similarity between the coming backup and the candidate backups is measured by calculating the distance of container usage. A distance list is created to keep track of the distance measurement between the coming backup and the candidate backups. The candidate backups with the shortest distances to the coming backup are considered highly correlated.

However, not all the fingerprints of the highly correlated backups can contribute to the dedupe. To improve the effectiveness of the fingerprint prefetching, the number of each container shared by the candidate backups are counted as an example shown in Figure 5.3. The fingerprints (marked in red) whose container has high number of counts (con1,3) are prefetched to the memory for dedupe.

Container Usage Pattern Mining

The CUPM is a core approach in the PBCCF and is proposed to discover the patterns and generalize the features using container usage data. The patterns here can represent the spatial locality of container usage. From Figure 5.1, we found that the backups are not randomly assigned to containers. Instead, the containers for each backup are

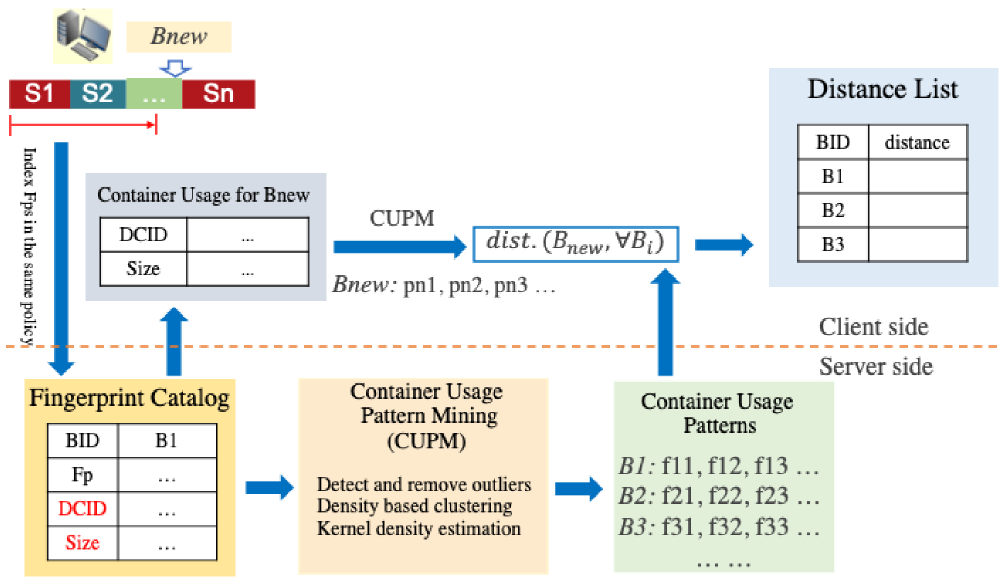


Figure 5.2: Implementation of PBCCF on the prefetching for the initial backup from the newly created client.

in several groups. The functions of the CUPM include outlier detection and removal, density-based clustering, and kernel density estimation, as shown in Figure 5.4.

The spatial locality here can also be regarded as the density of the data points in the figure. A cluster of data with high density forms a pattern. Each backup may have several patterns. The container usage of a backup that consists of three patterns is visualized with different colors in Figure 5.4. However, some data points, marked in blue, locate in the areas with low density. That means they do not show spatial locality and considered outliers. The outliers should be detected and removed to enhance the quality of the data. In the CUPM, a commonly used density-based clustering machine learning algorithm, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is applied to the lookup window of the coming backup and the candidate backups for cleaning and clustering the container usage data in terms of the density.

After obtaining the patterns of backups, the CUPM applies kernel density estimation (KDE) to fit each pattern of the candidate backup into a two-dimensional Gaussian distribution that is a commonly used probability function for analyzing the data. The

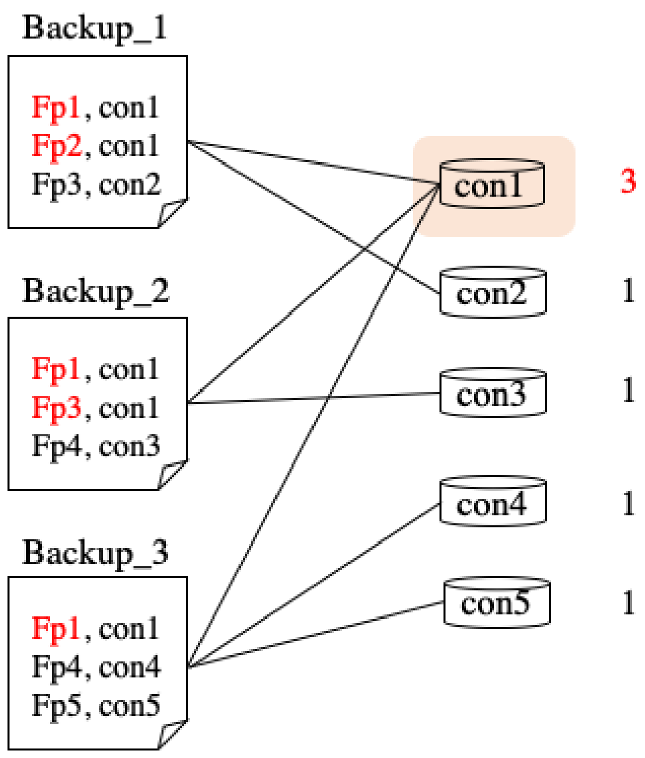


Figure 5.3: Determine the highly correlated fingerprints from the PBCCF identified backups by using the container usage data.

features of the distributions for different patterns include mean, covariance, and bandwidth. They are considered to be the characteristics of the backup and are saved in Container Usage Patterns as marked as f11, f12 , f13 etc., as shown in Figure 5.2, which is able to recover the distribution and approximately regenerate the container usage data. In contrast, the KDE is not applied to the initial backup ($Bnew$) since the lookup window does not have sufficient data to support KDE to learn a good distribution, thus only the patterns, pn1, pn2, pn3 etc. are maintained for measuring the distance to the candidate backups.

Container Usage Distance Measurement

The remaining problem is to measure the container usage distance between the

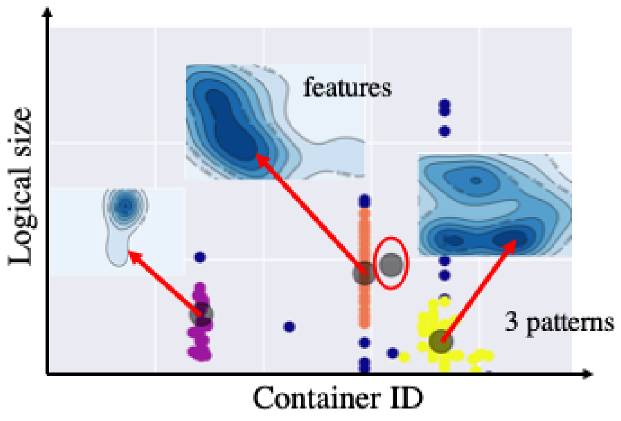
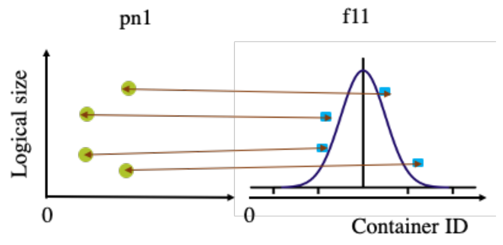
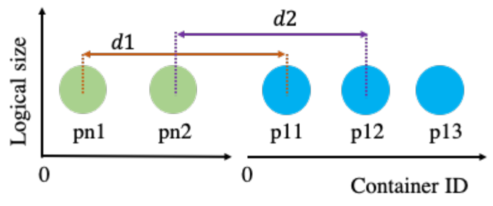


Figure 5.4: Application of CUPM to capture the patterns and features of the container usage data from a backup.



(a) Distance between two patterns, pn1 and f11.



(b) Distance between two backups, Bn and B1.

Figure 5.5: Container usage similarity measurement.

processing backup (the lookup window) and the candidate backups. The candidate backups with the shortest distance to the processing backup are considered as the affinity backups, and their fingerprints will be prefetched to the memory for dedupe. The commonly used approach is to fit the container usage data of the processing backup to the distribution, then measure the distance between two probability distributions using relative entropy. However, the distribution generalized from the limited data points will severely effect the representation of the processing backup. Also, it is not easy to calculate the relative entropy between two dimensional probability distributions. In this paper, we have proposed a more effective and practical way to measure container usage distance.

At the beginning, the calculation of the distance between two patterns (pn1 and f11) is introduced, as shown in Figure 5.5a. First, the probability distribution is recovered from the saved features of f11. Second, the data points with the same amount as pn1 are resampled from the recovered probability distribution. Third, a commonly used algorithm in computer vision, Nearest Neighbor, is applied to find the correspondence of data points between two patterns. Finally, the sum of euclidean distances between correspondent points is defined as the distance between two patterns. In most cases, backups consists of multiple patterns, as shown in Figure 5.5b. For each pattern (pn1 or pn2) of the processing backup, only the shortest distance ($d1$ or $d2$) to the pattern of the candidate backup (p11 or p12) is calculated. The sum of $d1$ or $d2$ is the distance between Bn and B1.

The CUPM is a core approach in the PBCCF. The CUPM is applied online with the processing backups that are scheduled for cleaning outliers and clustering into patterns. On the server side, the CUPM is applied offline to the container usage data of each client's latest completed backup for cleaning outliners, clustering into patterns, and probability features learning. The patterns and respective features are saved in the Container Usage Patterns for recording the backup behaviors of all clients, as shown in Figure 5.2. Since the frequency of full backups is low, typically every five days or less often, the frequency of the offline CUPM can be set to every week for updating the patterns and features.

Apart from the dedupe rate, the additional execution time and additional storage space are also critical to the implementation of PBCCF. The experiment was carried out on a regular machine with 16GB memory and 2.9GZ quad-core CPU. On the server side, the offline CUPM was applied to 4.5TB backups to identify the patterns and generalize the features. The data required for the CUPM is the summarized container usage data of each client's latest backup. The CUPM process was completed in three minutes. The additional storage space required for saving container usage patterns and features was only 3.5MB. On the client side, an additional several seconds were needed to capture patterns of the processing backup and accurately identify the fingerprints of highly correlated backups. From the testing, we can see that the additional overheads from the PBCCF, including execution time and storage space, can be ignored compared to a full backup process.

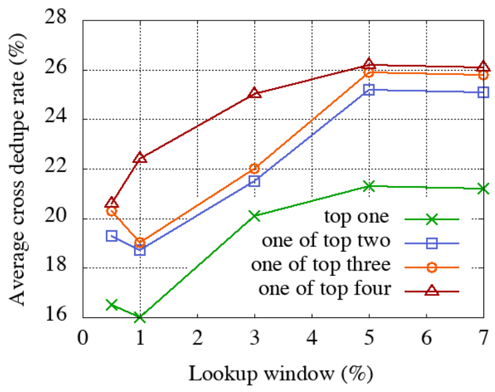


Figure 5.6: The evaluation of identifying the most similar backup by using the PBCCF with different lookup windows.

5.4 Evaluation of PBCCF

In order to evaluate the performance of the PBCCF, we carried out experiments to examine some important design parameters to provide significant insights. We also compared the performance of PBCCF to Progressive Sampling. The experiments were driven by the real-world dataset from a client-policy dedupe backup system. Some libraries used for implementing PBCCF are shown as follows:

- sklearn.cluster: Used to implement DBSCAN to cluster the data based on the density.
- sklearn.neighbors: Used to find the correspondence between points in two backups.
- scipy.stats.gaussian_kde: Used to fit the container usage data to the Gaussian distribution and resample the data from the learned distribution.
- seaborn: Used to calculate the container range given a sequence of container ID numbers.

First, we evaluated the performance of the PBCCF on identifying the best base backup. All fingerprints of the identified highly correlated backups are prefetched for dedupe. The average cross-dedupe rate of 55 clients was calculated. As presented in Section 5.2, the worst and best average dedupe rates were 10.6% and 30.0% when prefetching the fingerprints of a base backup for dedupe. The goal of the PBCCF is to achieve as close to the optimal dedupe rate (30.0%) as possible.

As introduced in Section 5.3, the CUPM uses the data of a small lookup window to approximate the container usage of the processing backup. The size of the lookup window affects the quality of the patterns mining and the dedupe rate. If the size of the lookup window is too small, the dedupe rate will be severely degraded due to the suboptimal quality of the pattern mining. Too large of a lookup window will result in a huge dedupe overhead from the client-server interaction. In order to evaluate the ability of the PBCCF to identify the best base backup for dedupe, we incremented the number of base backups to be prefetched from the distance list, but the best dedupe rate obtained from the one base backup was calculated. The evaluation of PBCCF with different lookup windows is shown in Figure 5.6. From the results, we can see that the dedupe rates of different prefetching numbers are increased at the beginning, then saturated at a 5% lookup window. Hence, only 5% lookup window in our dataset is sufficient for the CUPM to approximate container usage of the processed backup.

The result also exhibits the dedupe rate saturated at 21.3% when prefetching the top one backup from the distance list, which has a 10.3% improvement compared to the worst case. The gap to the optimal case comes from two factors: one is that the PBCCF does not identify the best base backup due to the bias in measuring the container usage similarity between two backups. The other one is from the bias in some backups using the container usage similarity to approximate the backup similarity .

The prefetching number was incremented to see the performance of the PBCCF to identify a good base backup. We can see the dedupe rate has a 4% improvement when increasing the prefetching number from one backup to two backups, then it is saturated at 26.2% when increasing the prefetching number to four backups, which presents the PBCCF as having a good ability to identify a good base backup. Therefore, there is a good possibility that the gap to the optimal case (30%) comes from the aforementioned second reason. However, the gap is less than 4% and we can still claim that the PBCCF succeeds in improving the dedupe rate for the initial backup from the newly created client.

Second, we evaluated the performance of the PBCCF on dedupe by varying different memory size from 30MB to 120MB for prefetching, which is shown in Figure 5.7. The memory size, 30MB, 60MB, 90MB, and 120MB allows to prefetch all fingerprints from two, three, four and five identified highly correlated backups for dedupe. From the results, the dedupe rate is improved from 25.1% at 30MB to 32.2% at 120MB. When the memory size is large enough to hold all fingerprints of backups from the other 54 clients, the dedupe rate can be reached 35.9%. We can see the PBCCF has only 3.7% less dedupe rate by prefetching fingerprints of five backups, compared to prefetching all fingerprints from the whole backup system.

As discussed in Section 5.3, not all fingerprints from the identified highly correlated backups can contribute to dedupe. In order to improve the effectiveness of the fingerprint prefetching, the number of the containers shared by the backups identified by the PBCCF are counted. The fingerprints are prefetched based on the order of the number of counts. Higher number of the counts, higher priority of the fingerprint is prefetched. For distinguishing two approaches, the PBCCF_1 is to prefetch the all fingerprints of the identified highly correlated backups, while PBCCF_2 is to prefetch the highly correlated fingerprints with high number of shared container counts.

Third, we tested the PBCCF_2 about the size of the memory required to maintain the dedupe rate as the PBCCF_1, which is shown in Table 5.2. From the results, we can see the PBCCF_2 saves at least 4x memory while maintaining the same dedupe rate as the PBCCF_1. Hence, it is significantly effective to estimate the container usage of the initial backup from the newly created client by using the number of the shared containers shared by the highly correlated backups.

Forth, we compared the PBCCF to the existing generic prefetching algorithm, Progressive Sampling, using in the container-fashion dedupe system. It randomly samples the fingerprints from each container based on the certain sample ratio that is determined by the available memory size. The number of the fingerprint hits are counted. The fingerprints in the containers with high number of hits are all prefetched to memory for dedupe. We varied the sample ratio to obtain the required memory of the Progressive Sampling to maintain the dedupe rate. Without considering the backup content correlation, the Progressive Sampling requires at least 40x memory to maintain the dedupe rate, as shown in Table 5.2. The prefetching performance of Progressive Sampling highly relies on the effectiveness of the sampled fingerprints. Therefore, it requires more memory to sample more fingerprints to maintain the prefetching performance. As an increase of the amount of backups, larger memory is required for Progressive Sampling, but it is not always possible for the enterprise backup system due to the limited budget.

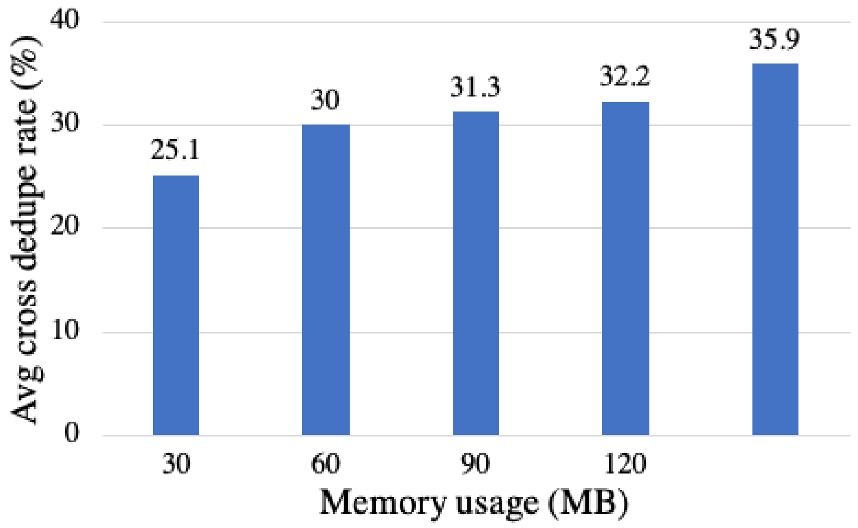


Figure 5.7: The average dedupe rate achieved by the PBCCF with different memory sizes.

Avg cross	Memory usage (MB)		
dedupe rate(%)	PBCCF_1	PBCCF_2	Progressive Sampling
25.1	30	4	156
30.0	60	8	624
31.3	90	16	809
32.2	120	32	1248

Table 5.2: Comparison of the memory usage to maintain the dedupe rate between different prefetching approaches.

5.5 Conclusion

Fingerprint prefetching is a critical approach for backup systems to accelerate the dedupe performance; however, very few studies have focused on this area. Particularly the initial backup from a newly created client, the dedupe rate is low due to no historical backups for the reference basis. Due to the behavior of the backup users, the backup content correlation exists among some different clients, which is explored by using a real-world dataset collected from a client-policy enterprise backup system. We proposed a prefetching algorithm called Prefetching Backup Content Correlated Fingerprint (PBCCF), which can be easily implemented on current backup systems. Instead of using deep learning techniques that require huge training and implementation overhead, the CUPM approach applies a combination of lightweight machine learning algorithms, but produces a significantly good performance on pattern capture and feature generalization. The input of the CUPM is the high-level metadata that provides the container usage information, so it does not concern the confidential information of the customers. The additional overhead of implementing the PBCCF including execution time and storage space is negligible compared to a full backup process. The experimental results shows PBCCF achieved good prefetching improvement while significantly saving memory, compared to the Progressive Sampling.

Chapter 6

Adaptive Embedding Spatial Encoder-Based Deduplication

The previous two chapter introduced the machine learning for backup systems. This chapter is going to present the work on the backup system for machine learning [59].

6.1 Introduction

As artificial intelligence (AI) techniques continue to develop, they reliably exhibit state-of-the-art performance in certain areas, including image recognition [60], object detection [61], natural language processing [62], and many others [63, 64, 65]. AI generally refers to the machine learning component, which includes a wide range of algorithms from statistical approaches to deep learning. In the Big Data era, the exponential growth of data and the proliferation of IoT and telemetry techniques enable streaming data to be collected more efficiently and easily. This capability is greatly changing the way we use data. More and more services in our lives, such as voice-activated assistants and self-driving cars, seek to replace human resources with AI, which can enhance service levels by continuously learning from the real environment.

The dramatic increase in the amount and importance of data makes it necessary to have efficient backup systems to protect data from loss to ensure business continuity. The success of AI has become an important part of enhancing the service levels provided by backup systems, which is also called AI for backup systems. A significant number of studies have been carried out toward applying machine learning techniques to backup systems, including automatically scheduling backup tasks [16], effectively improving data deduplication by learning an optimal indexing strategy [66], dynamically optimizing backup system configurations by predicting backup capacity usage [36], and effectively prefetching the accurate fingerprints for improving the deduplication rate of the backup storage systems [43]. However, little effort has been made in the other direction, on backup systems for AI.

Machine learning or deep learning techniques are classes of data-driven models that use many collected data points to explore a model structure and its parameters to optimize performance objectives. As increasing quantities of data are collected over time, the learned model is supposed to be updated or retrained to improve performance after obtaining more information using new data combined with previous data, which makes it necessary to save or back up the training data.

This large amount of data will result in huge overhead if all data are always backed up to storage. Duplicates exist in many kinds of data generated from a data center or a department. For example, multiple virtual machines are created using the same image, different computers run on the same platform, emails with file attachments are sent to multiple machines, and so on. Deduplication is one of the core components of a modern backup system, which enables 10–50 times more data to be stored in the same storage space by identifying and reducing duplicates [9]. Deduplication applies certain segment algorithms to divide the data streams into segments, after which each segment is hashed to a fingerprint using a known secure hashing algorithm [25]. Fingerprints are used to identify whether the segment content is unique, and only unique segment content is saved to backup storage. Due to their different purposes and characteristics, training data rarely contain this kind of duplication, which means that existing deduplication approaches no longer work on training data.

However, most training data have some extent of redundancy in the view of machine learning, particularly machine learning or deep learning a classification task [67]. For example, in a dataset, if three large pictures contain only one different pixel, current deduplication approaches will assign three different fingerprints to them and consider them unique pieces of content. However, these three pictures are highly similar, which means that only one of them will significantly contribute to learning in a classifier.

Intuitively, the best approach is to back up one of the pictures and consider the other two as redundant. However, it is not trivial to measure similarity among high-dimensional data. Even if a perfect similarity measurement approach is designed, it will trigger huge overhead to identify redundancies in large amounts of data.

Hence, this paper proposes a novel backup strategy based on current backup infrastructure for protecting the training data of a deep learning classifier. The effective deduplication in the backup strategy is called **ada**ptive **emb**edding spatial **encoder**-based deduplication (AdaEmb-encoder) and can reduce training data redundancy in the streaming version while preserving data structure. The main contributions of this paper are:

- Improving the service levels of current backup systems by supporting efficient backup of machine learning data.
- Proposing a novel backup strategy for effectively protecting training data over time.
- Introducing a new idea to adaptively encode high-dimensional data while preserving data structure during backup.
- Designing an efficient approach to measure and reduce redundancy in training data.
- Testing the performance of the backup strategy for training data and varying the design parameters to study the tradeoffs.

The remainder of this paper is organized as follows: Section 6.2 introduces the background of the current deduplication system and the machine learning techniques that will be used in the proposed backup strategy. Section 6.3 presents the design of the AdaEmb-encoder-based deduplication system. Its evaluation is discussed in Section 6.4, and conclusions are drawn in Section 6.5.

6.2 Background and Motivation

This section will first introduce the modern backup infrastructure that can be used to protect training data collected from the real environment. The relevant machine learning techniques that are used in the proposed backup strategy to assist in efficient backup will also be introduced.

6.2.1 Modern backup infrastructure

The client-server infrastructure is commonly used in current enterprise backup systems [10, 11, 12, 13, 14, 15]. Clients are the sources that generate the backup data. They include the laptops that create files, the virtual machines that run application services, and the servers that support database transactions. This paper looks at Internet-based mobile devices, embedded sensors, or other devices that collect data to train learning models, as shown in Fig. 6.1. The media servers' task is to handle deduplication of data from clients and to manage storage media. The storage area consists of different kinds of storage media that protect backup data. The master server holds the backup strategies that schedule and process backups to storage. The backup infrastructure in shown in Fig. 6.1 describes the copying of the training data collected from multiple mobile devices and their transfer across the network to the backup system for protection. The training data can be retrieved from the backup system when they become unavailable at the client side and are no longer used for training. The raw training data typically are useless, and therefore the other option is to recover the machine learning model from the backup system by training using the data on the backup system side. To reduce the impact on the client side, the backup process is scheduled during downtime, for example, from 12 midnight to 3 AM.

6.2.2 Deduplication process

A huge storage and computing overhead will be incurred if all data generated from the client side are always backed up. This is not always possible for most backup users due to limited budget. Therefore, deduplication is used in most modern backup systems to reduce backup data redundancy. Fig. 6.2 shows an example that illustrates the deduplication approach. A large mass of backup data is chunked into a number of small segments using a segmentation algorithm, after which a hashing algorithm is used to obtain fingerprints such as A, B, C, and E of the first backup. The data contents of A, B, C, and E are all backed up to storage because there is no fingerprint reference in

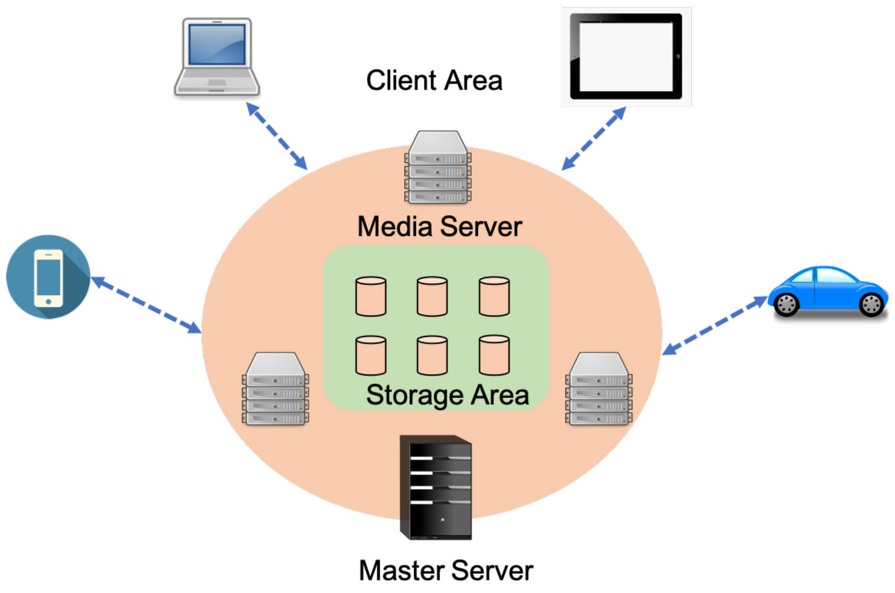


Figure 6.1: Backup infrastructure for training data

the initial state. For the second backup, the deduplication engine in the media server is activated to identify duplicates, which are A and B. Only the unique data content E needs to be backed up, and the other two duplicates back up only their indices. This deduplication process achieves good duplicate reduction performance with regular data types such as files and block data, but it becomes ineffective if applied to general training data.

Because this paper focuses on the proliferation of data collection techniques and the excellent performance of AI or machine learning on image recognition, a large number of pictures were generated and processed by the learning models to obtain optimal performance on problems that rely on image recognition, such as face recognition, object detection, and autonomous vehicle operation. As for the data collected to train neural networks, they are typically structured so that the pictures consist of the same number of pixels. To preserve picture structure, the current backup system chunks the picture backups based on a picture-by-picture scan and hashes them to their fingerprints. The same deduplication is used to reduce the redundancy of the picture backup. A very low deduplication rate is obtained because a very small number of duplicates exist

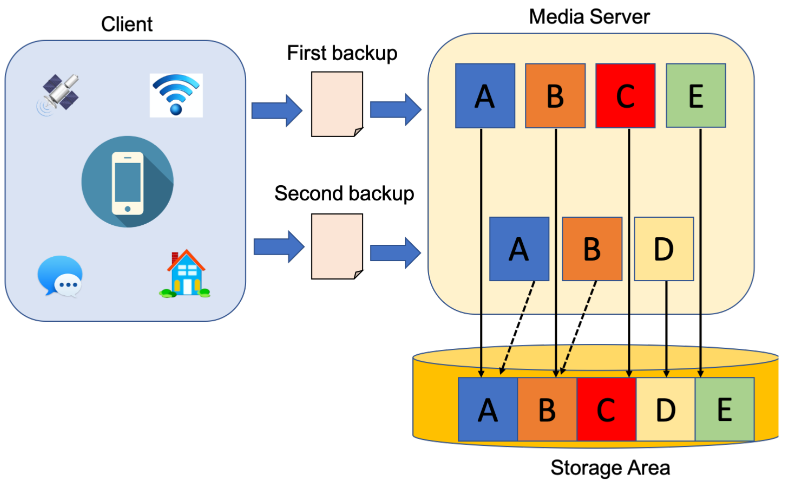
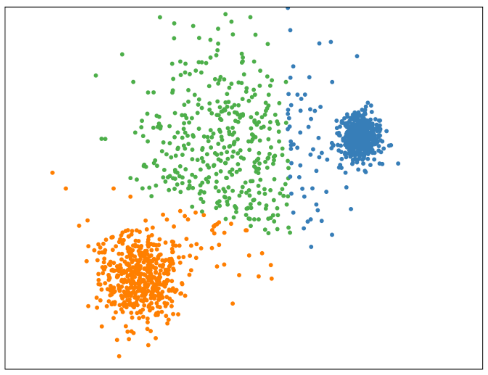


Figure 6.2: Deduplication in backup systems.

among the training data. In this deduplication process, two pictures with only one pixel different are both identified as unique pieces of content. A high level of redundancy typically occurs in training data. The accumulation of a large quantity of training data results in the need for a high-complexity learning model to explain the data and a higher training overhead to obtain the optimal model parameters. High-redundancy data require large storage capacity, but contribute very little to learning, which severely reduces the efficiency and the service level of backup systems.

6.2.3 Data clustering

Data clustering is used to group a set of data based on a specific rule. Two frequently used data clustering algorithms are k-means and density-based spatial clustering of applications with noise (DBSCAN). These two clustering algorithms yield similar performance on grouping the data in some kinds of dataset, as shown in Fig. 6.3. Both algorithms achieve good performance on grouping the data into three clusters. DBSCAN can additionally identify the outliers that lie alone in low-density regions. However, for some kinds of dataset, as shown in Fig. 6.4, the DBSCAN algorithm outperforms the k-means on preserving data structure during data grouping, and therefore DBSCAN was chosen for the proposed backup strategy.


(a) k-means

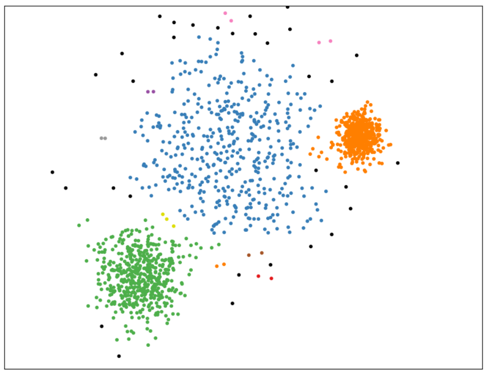
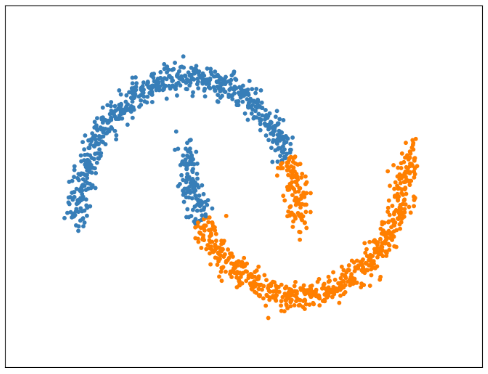

(b) DBSCAN

Figure 6.3: Grouping data into three main clusters.


(a) k-means

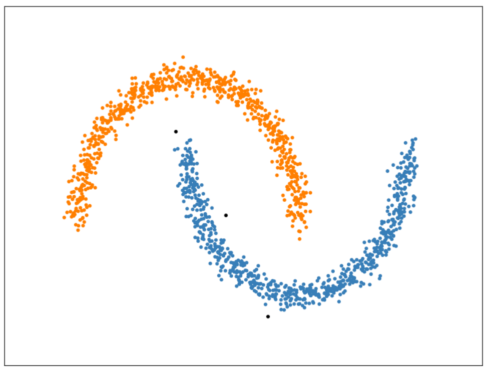

(b) DBSCAN

Figure 6.4: Grouping data into two main clusters, where DB-SCAN is better at preserving data structure.

6.2.4 Embedding space

High dimensional data frequently exists in the real life. A common example is a picture with hundreds of pixels or more. For AI or machine learning, the data with high dimensions generally requires more complicated learning model to explain the data and more data to train the model, which results in huge training overhead. Additionally, particularly in the classification task, not all pixels are capable of providing valuable information for learning the classification strategy. Therefore, the dimensionality reduction techniques were developed to project the data into the low dimension space which is called embedding space in many studies, which is shown in Fig. 6.5.

In the AI or machine learning fields, Principle Component Analysis (PCA) [68], t-Distributed Stochastic Neighbor Embedding (t-SNE) [69] and Uniform Manifold Approximation and Projection (UMAP) [70] are three commonly used dimensionality reduction techniques to project the high dimensional data into the embedding space while trying to preserve the global structure of the data. PCA assumes the training data potentially lies in the linear space and statistically learn an embedding space that maximize the variance of the data, while t-SNE assumes the data lies in the nonlinear space explained by the Gaussian distribution. A low dimensional t-distribution is learned from the data to form a embedding space for explaining the data by minimizing the cross entropy between those two probability distributions. UMAP is the other nonlinear dimensionality reduction technique. It assumes the data uniformly distributes on a Riemannian manifold and learns an optimal topological representation of the data in the embedding space. The PCA requires the lowest training overhead compared to the other two techniques. However, the linear projection of the data makes the PCA fail to capture the nonlinear dependences between the dimensions. The introduction of the nonlinear projection of t-SNE gives a great improvement on explaining the data in the embedding space, but it incurs a large training overhead. The UMAP outperforms the other two not only on the dimensionality reduction, but also requiring much less training overhead compared to the t-SNE [70]. Therefore, the UMAP is used in our backup strategy for deduplicating the high dimensional training data.

6.3 AdaEmb-encoder Deduplication

This section presents the design of AdaEmb-encoder deduplication for deep neural network classifier training and introduces how to implement it on the current backup infrastructure.

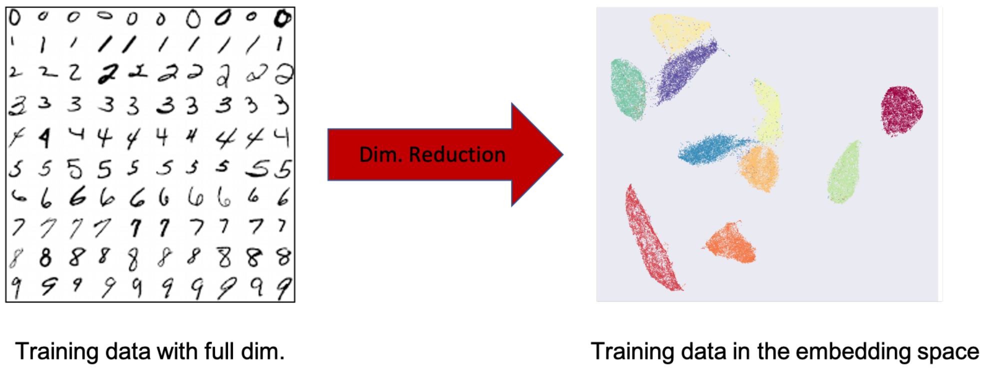

Dim. Reduction
Training data with full dim.
Training data in the embedding space

Figure 6.5: Project the high dimensional data to the embedding space

A backup strategy for training data

The proposed backup strategy aims to provide efficient backup of high-dimensional deep neural network training data for classification engines, such as CNNs. To achieve this goal, a novel deduplication engine has been developed for measuring and reducing training data backup redundancy while preserving critical information for learning a good neural network on the classification task.

The example shown in Fig. 6.6 illustrates the strategy used in training data backups over two consecutive days, day i and day i+1. On day i, as in traditional backup systems, the backup generated from the client side first goes to the deduplication engine. Instead of using the hashing algorithms, the training data are encoded as the location in the embedding space that was learned from the data using the UMAP algorithm. Compared to the hashing algorithms, the UMAP encoding somehow preserves the global structure of the training data, which makes it possible to measure the redundancy of the backup data using the UMAP locations. Only low-redundancy data are considered unique and are saved into backup storage because high-redundancy data make little contribution to classifier learning. During downtime, the UMAP encoding model in the deduplication engine is updated offline to the latest version after obtaining more data and saved for the next backup, for example, on day i+1.

AdaEmb-encoder

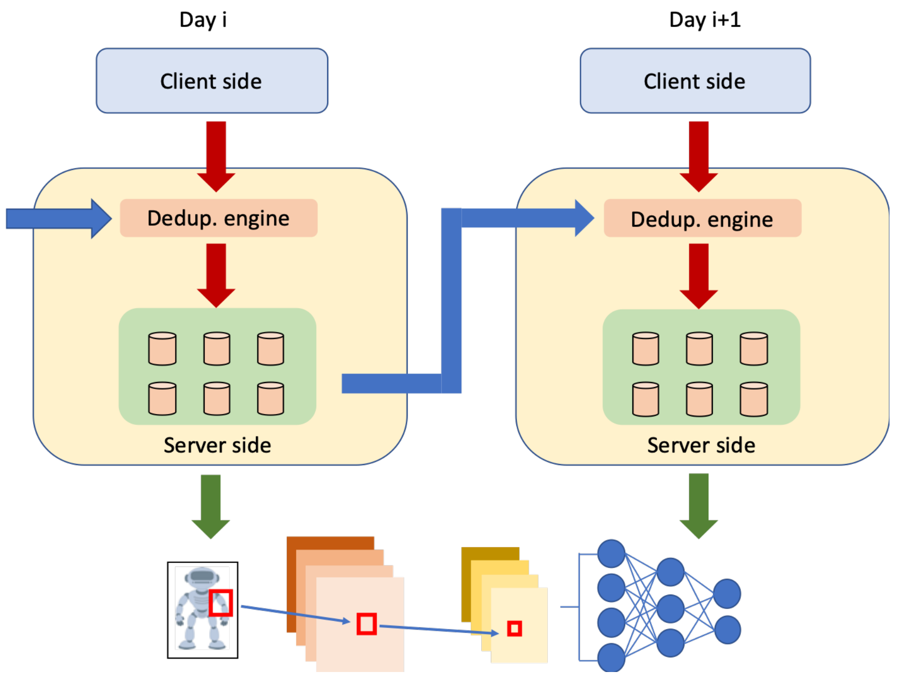


Figure 6.6: A backup strategy used to protect training data.

The hashing algorithm used in traditional deduplication, for example, the MD5 algorithm [71], cryptographically encodes each piece of data into a hash value. For a piece of data to be successfully identified as a duplicate using the hash value, the data must be exactly identical, a situation that does not commonly exist in high-dimensional training data and results in a low deduplication rate. This study used the UMAP algorithm to encode the training data. Fig. 6.7 shows an example involving the encoding of 10,000 784-pixel MINST handwritten digits. Each handwritten digit is encoded as its coordinates (x,y) in the 2D embedding space, which is called an adaptive embedding encoder (AdaEmb-encoder) in this paper. The example reveals that ten data classes from '0' to '9' are well separated in the 2D-UMAP embedding space. Due to similar structure and bias introduced by handwriting, some data classes are much less distinguishable, such as [4,7,9] and [3,5,8]. This situation is clearly represented by the distances among the encoded data. The encoded data of [4,7,9] and [3,5,8] are close to each other and relatively far away from the other classes [0,1,2,6]. Therefore, the AdaEmb-encoder yields good performance on encoding high-dimensional data using a low-dimensional space,

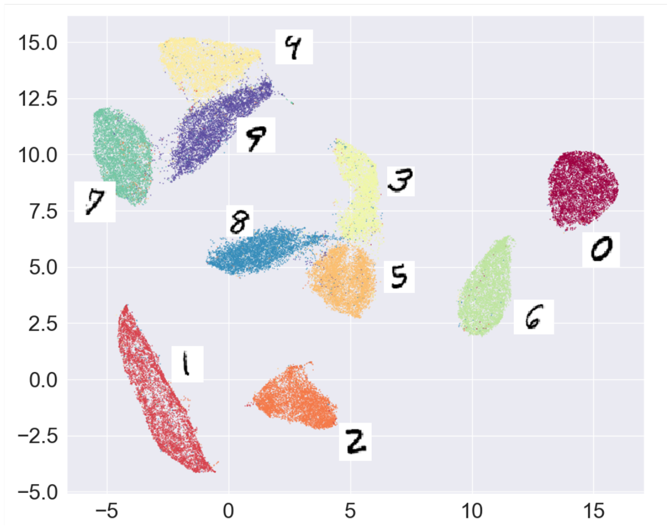


Figure 6.7: The MNIST dataset is encoded as the coordinates in the 2D-UMAP embedding space.

but also preserves the global structure of the data.

During the backup, a backup, bk_i, with N pieces of data (0,1, ... n, ... N-1) are generated on day i, and the unique data, $uniq_i$ with $M(\leq N)$ pieces of data, are identified by the deduplication engine and saved into backup storage, as shown in Fig. 6.8. To achieve better performance on encoding and interpreting the data, the AdaEmb-encoder and the data clustering model (DBSCAN) are updated offline using the existing data $uniq_{i-1}$, which is shown in ⓪. Given a new piece of data $bk_i(n)$, the coordinates of the data are computed by the AdaEmb-encoder and then passed into the DBSCAN model. The data are assigned to the nearest group in terms of their location in the embedding space, which is shown in ①.

Redundancy measurement

It is extremely challenging to measure redundancy among high-dimensional data due to the curse of dimensionality. The UMAP-based AdaEmb-encoder achieves good performance on explaining the data in the embedding space, and therefore the redundancy level of the data can be measured by calculating the distances among the data in the

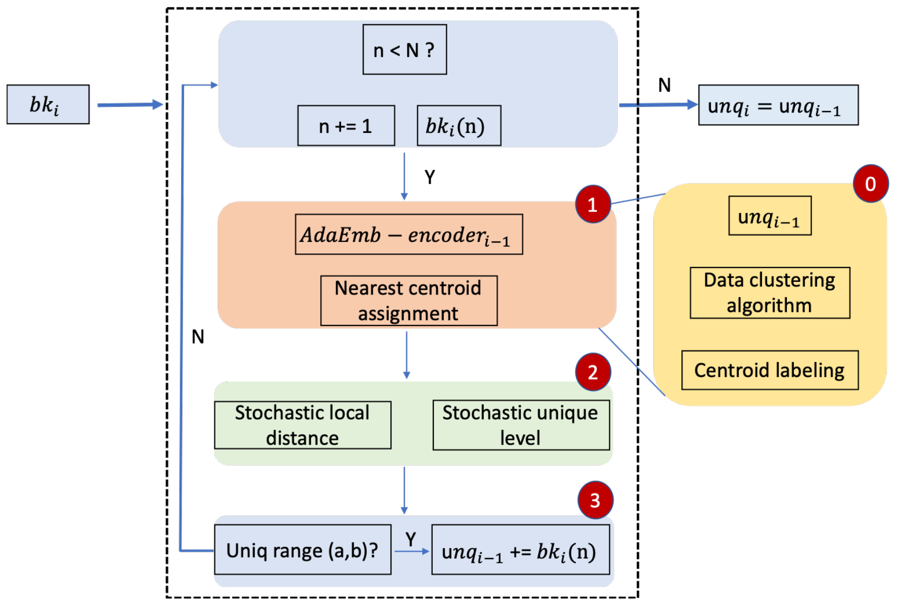


Figure 6.8: The deduplication process for training data.

learned embedding space. The data redundancy measurement approach is illustrated in ② and ③ of Fig. 6.8. More details are introduced below.

Stochastic local distance. To measure the redundancy of a piece of new incoming data $bk_i(n)$, the local distance of the existing data that are in the same group as $bk_i(n)$ is calculated to serve as a reference baseline, as shown in Fig. 6.9. The local distance is quantified as the mean value of the Euclidean distances between the existing data in the same group and the group centroid. To reduce computing overhead, the local distance is calculated stochastically as follows:

1) Randomly sample p data points from the data in the same group.
2) Calculate the Euclidean distances between each sampled data point and the centroid.
3) Repeat the above two steps q times.
4) The stochastic local distance is calculated as the mean of the pq distances.

Stochastic unique level. The unique level of $bk_i(n)$ is defined as the minimum

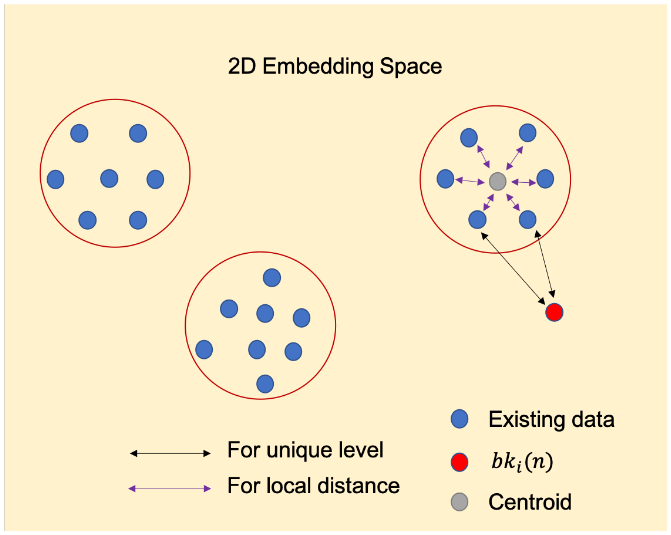


Figure 6.9: Local distance and unique level in the 2D embedding space.

distance to an existing data point in the same group, as shown in Fig. 6.9. Similarly, to reduce computing overhead, the unique level is calculated stochastically as follows:

1) Randomly sample p data points from the data in the same group.

2) Calculate the minimum Euclidean distance between $bk_i(n)$ and the sampled data.

3) Repeat the above two steps q times.

4) The stochastic unique level of $bk_i(n)$ is calculated as the mean value of the q distances.

Based on the authors' observations, the performance of stochastic local distance and stochastic unique level is robust to a wide range of p and q parameters. The data $bk_i(n)$ will be identified as unique if their unique level lies in the *unique range* defined by [a,b]. The lower and upper bounds of the unique range are quantified by the proportion of the local distance (LD), [a*LD, b*LD] (simply to be denoted as [a,b]), where $0 \leq a < b \leq 1$. The data $bk_i(n)$ will be classified as redundant if their unique level is less than a. To avoid destroying the data structure, $bk_i(n)$ will be classified as an outlier if their unique level is greater than b. Based on this setting, the unique range is adaptively adjusted to the change in LD as each unique data point is identified. After going through all backup data points bk_i, the data after deduplication, $uniq_i$, are eventually saved into backup storage.

6.4 Experiment results

To evaluate the performance of the AdaEmb-encoder, experiments were carried out to examine some important design parameters to provide significant insights. The experiments were driven by a commonly used high-dimensional multi-class dataset, MINST Handwritten Digits. The dataset consists of 60,000 training data points and 10,000 test data point, where each data point has 784 dimensions. Some important libraries used in efficient deduplication are:

- sklearn.cluster: Used to implement DBSCAN to cluster the data based on density.
- UMAP: Used in the AdaEmb-encoder to encode high-dimensional data as their coordinates in the embedding space.
- Keras: Used to construct CNNs to test and evaluate the tradeoffs of the proposed backup strategy.

To simulate the backup process, the dataset was divided into ten groups, and each group was backed up daily. In the real case, data generated by clients are not guaranteed to follow the same distribution on different days, which is also considered in the design of the AdaEmb-encoder. Therefore, the ten daily backups had different distributions created by forcing them to bias to different classes. For example, the backup on day 1 was biased to the digit '0', and the backup on day 10 was biased to the digit '9'. During backup, the backup data redundancy was reduced using the 2D AdaEmb-encoder deduplication engine. Only the data identified as lying in the defined *unique range* were saved into backup storage.

First, this section presents how strongly the *unique range* affects the redundancy in the data that are backed up. In the experiment, four kinds of redundancy levels (denoted as high, high medium, low medium, and low) were tested by varying the *unique range* as [0.1, 0.2], [0.3, 0.4], [0.7, 0.8], and [0.9, 1.0] respectively. Fig. 6.13 shows examples of backup data with these four kinds of redundancy. The existing data were used to help the AdaEmb-encoder learn, whereas the unique data were identified as lying within the *unique range*. The results show that the high-redundancy unique data (*unique range*: [0.1, 0.2]) are centralized while they are close to the existing data, as shown in Fig. 6.13a. As the unique level increased, low-redundancy unique data (*unique range*: [0.9, 1.0]) became distributed around the existing data, as shown in Fig. 6.13d.

Second, the CNN training performance was tested using backup data with different redundancy levels. A regular CNN was constructed for all following experiments. The backup process was simulated as introduced in Section 6.3 and proceeded for ten simulated days. Fig. 6.10 shows the training performance of the CNN on day 10. The CNN was trained using the backup data (i.e., $uniq_{10}$) with the same amount of data, but different redundancy levels, and the classification accuracy was evaluated using the test data. The results reveal that lower-redundancy training data yielded more stable and consistently better classification performance than higher-redundancy training data during an entire training process. This result also verifies that lower data redundancy provides more valuable information for training the network. Furthermore, a greater improvement in classification accuracy is gained by using low-redundancy training data at the beginning of the training process.

Third, the classification accuracy of the CNNs trained by the backup data was evaluated on different simulated days (i.e., $uniq_1$, $uniq_1$, ..., $uniq_{10}$), as shown in Fig. 6.11. As the backup proceeded, the classification accuracy of the CNNs trained by backups with different redundancy levels increased from day 1 to day 10 because more data were collected. Moreover, the CNN trained by lower-redundancy backup data exhibited consistently better classification accuracy than the CNN trained by higher-redundancy data from day 3 to day 10. The first two days were an exception due to the limited amount of backup data available for training a good deduplication engine. The size of the backup on day 10, $uniq_{10}$ with low redundancy, was 3,966, and the CNN trained by those data achieved 97.8% classification accuracy, which was only 1.3% less than the CNN trained using 60,000 training data points.

Fourth, the dimensions of the embedding space used to encode the backup data were increased. Fig. 6.12 shows the performance behavior of embedding spaces of various dimensions. The results showed that the AdaEmb-encoder-based deduplication engine

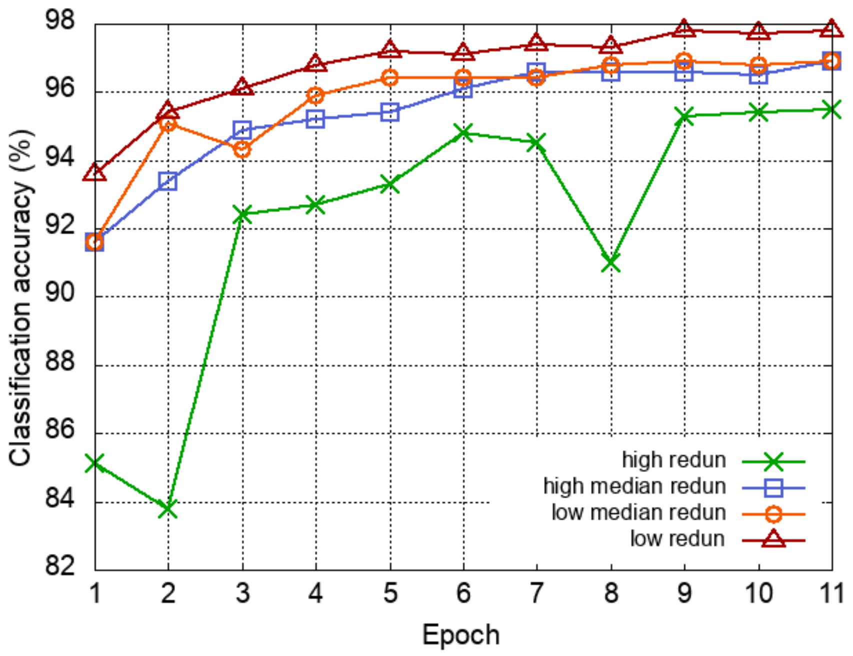


Figure 6.10: Training performance on backup data with different redundancy levels.

exhibited robust performance in reducing data redundancy in a low-dimensional embedding space. The classification accuracy improved by 0.2% when the embedding space was increased from 2D to 4D. However, the classification performance then decreased from 98.0% to 97.2% when the embedding space was increased from 4D to 6D. The curse of dimensionality resulted in degrading the performance of the AdaEmb-encoder on measuring redundancy.

Fifth, the upper bound of the *unique range* was maintained at 1. The lower bound was decreased from 0.9 to 0.7, 0.5, and 0.1 to reduce deduplication power, or in other words, more data with lower redundancy were backed up for training. Fig. 6.14 shows the classification accuracy and training time of the CNNs trained by different amounts of data in backup storage (i.e., $uniq_{10}$). Table 6.1 gives information about the system used to test the training time. The horizontal axis presents the amount of data after deduplication as a proportion of the original 60,000 training data points. A proportion of 1 means that no data were deduplicated and that all data in backup storage were used for training. As the proportion increased from 0.07 to 1.0, the classification accuracy slightly

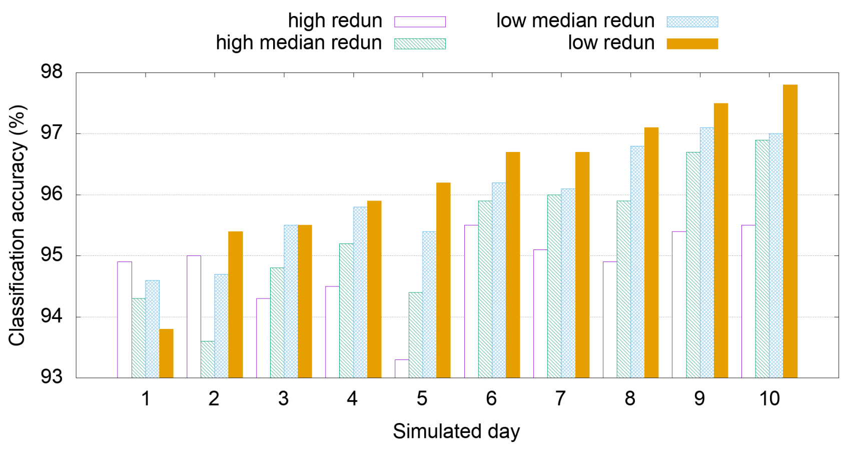


Figure 6.11: CNN classification accuracy for different simulated days.

Model name	Quad-Core Intel Core i7
CPU(GHz)	2.9
CPU cores	4
Cache size	8 MB

Table 6.1: System used to run the experiments.

improved from 97.8% to 99.1%, but the training time significantly increased from 108 seconds to 1092 seconds. From the evaluation just described, AdaEmb-encoder-based deduplication achieved good performance on reducing redundancy while preserving data structure. Therefore, compared to the CNN trained using all data, AdaEmb-encoder-based deduplication saved 93% of backup storage space and reduced training time by a factor of 10 while incurring only 1.3% loss of classification accuracy.

6.5 Conclusion

Data deduplication is a core component in backup systems to reduce the number of duplicates in the backup data. However, traditional deduplication techniques are ineffective for the data used to train a neural network classifier, particularly for image recognition tasks. This paper has proposed a novel deduplication strategy called the AdaEmb-encoder. Instead of using a hashing algorithm, each data point is encoded as

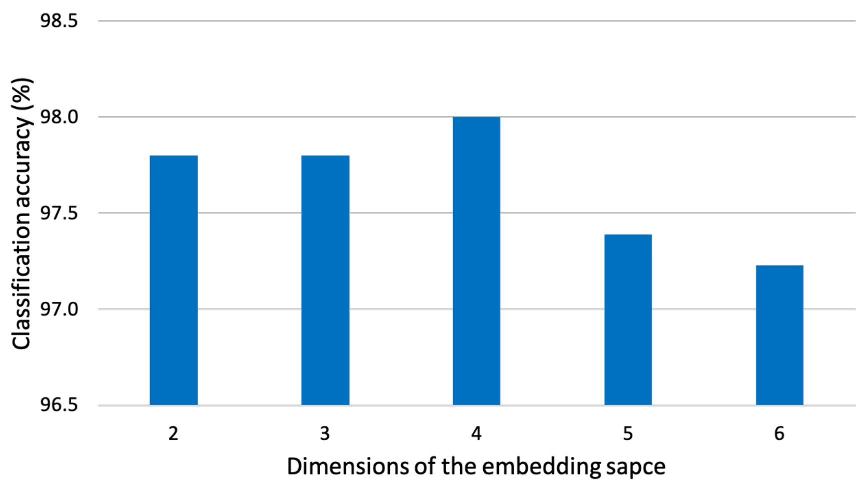
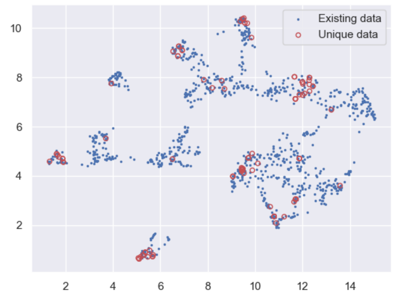
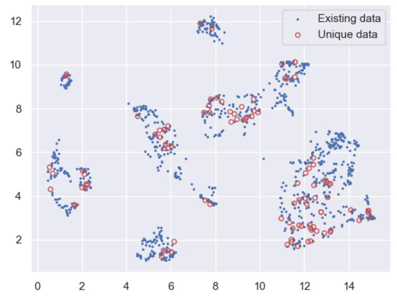


Figure 6.12: Classification accuracy of embedding spaces of various dimensions used for encoding data.

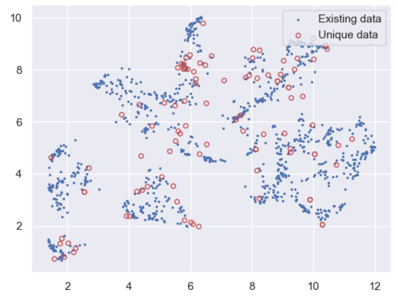
its location in the embedding space using the UMAP encoder. During the backup, the AdaEmb-encoder deduplication engine adaptively adjusts the *unique range* to preserve data structure. Based on the authors' observations, it exhibits good performance on identifying critical data in the embedding space. Experimental results showed that the AdaEmb-encoder effectively deduplicated 93% of the training data. The CNN trained by the identified unique data reduced training time by a factor of 10 while incurring only 1.3% loss of classification accuracy.



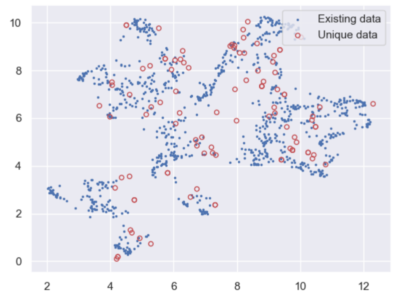
(a) high redundancy



(b) high median redundancy



(c) low median redundancy



(d) low redundancy

Figure 6.13: Data with different redundancy levels are backed up.

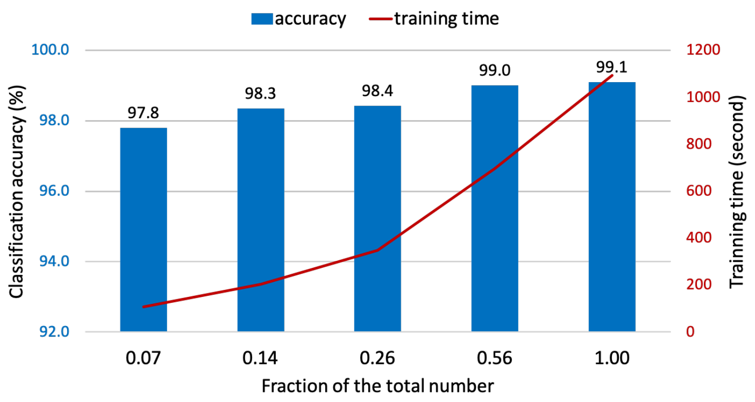


Figure 6.14: Classification accuracy of CNNs trained by different amounts of low-redundancy data.

Chapter 7

Conclusion

With the increase in the scale of backup systems and the exponential growth of the data generated by enterprises, static backup systems are having challenges satisfying the higher service level required by backup users. Given the success of the machine or deep learning techniques in data clustering, feature and pattern generalization, and similar applications, we designed the intelligent agent HyperProtect, which aims to leverage the efficiency and reliability of large-scale enterprise backup systems.

In Chapter 2, HyperProtect scheduling is proposed to enhance the stability and predictability of backup systems. Previous works mainly focused on the reduction of the time required to complete the backup process but ignored the recovery performance and stability of systems. Hence, HyperProtect scheduling is proposed. According to our testing, it successfully improved the backup time consistency and reduced storage switching, while maintaining high backup efficiency.

In Chapter 3, the high-efficiency deduplication is introduced to improve deduplication performance while significantly reducing redundancy for existing clients. Combined with HyperProtect scheduling (introduced in Chapter 2), the developed high-efficiency deduplication offers a more stable and predictable environment for backup systems.

In Chapter 4, a novel structure is presented to forecast the backup storage usage and predict the time that the backup storage will reach this capacity by using machine learning techniques. In comparison with a prevalent backup system forecasting tool, our structure provides better, more robust predictions of backup storage capacity and the size of data generated by a given number of clients and the associated deduplication ratios.

As discussed in Chapter 5, there is no historical data to reference for deduplication for newly created clients. We explored the backup content correlation existing in the different clients protected in the same backup system. Hence, we propose a fingerprint prefetching algorithm, called prefetching backup content correlated fingerprint (PBCCF), to improve the prefetching performance. Lightweight machine learning and statistical techniques are applied to discover the backup patterns and generalize their features using only high-level meta data.

As shown in Chapter 6, the traditional deduplication technique failed to effectively reduce the redundancy of the training data in terms of learning models. Hence, we present a novel and effective deduplication engine, AdaEmb-Encoder, for the new data type (deep learning classifier training data). In our test, AdaEmb-Encoder significantly decreases the storage space and training time, while maintaining good classification accuracy.

In summary, this thesis mainly focuses on the design of an intelligent agent, HyperProtect, that is capable of optimizing large-scale backup storage systems by adapting to the dynamic backup environment. HyperProtect includes smart scheduling and high-efficiency deduplication for leveraging backup efficiency, backup stability, and backup predictability; the novel forecasting structure for improving system reliability; the PBCCF deduplication strategy for accelerating the backup process and reducing the storage overhead; and the AdaEmb-Encoder deduplication engine for effectively reducing data redundancy in terms of learning classifiers while backing up the training data.

References

[1] George Amvrosiadis and Medha Bhadkamkar. Identifying trends in enterprise data protection systems. In *USENIX Annual Technical Conference*, pages 151–164, 2015.

[2] Rob Kolstad. A next step in backup and restore technology. In *Proceedings USENIX LISA V*, pages 73–79, 1991.

[3] Steve M Romig. Backup at ohio state, take 2. In *Proceedings of the Fourth Large Installation System Administrator's Conference (LISA IV)(USENIX Association: Berkeley, CA)*, page 137, 1990.

[4] Elizabeth D Zwicky. Torture-testing backup and archive programs: Things you ought to know but probably would rather not. In *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)(USENIX Association: Berkeley, CA)*, page 181, 1991.

[5] Suparna Bhattacharya, C Mohan, Karen W Brannon, Inderpal Narang, Hui-I Hsiao, and Mahadevan Subramanian. Coordinating backup/recovery and data consistency between database and file systems. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 500–511. ACM, 2002.

[6] Ann Chervenak, Vivekenand Vellanki, and Zachary Kurmas. Protecting file systems: A survey of backup techniques. In *Joint NASA and IEEE Mass Storage Conference*, volume 99, 1998.

[7] James Da Silva, Ólafur Gudmundsson, and Daniel Mossé. Performance of a parallel network backup manager. In *USENIX Summer*, 1992.

[8] James Da Silva and Olafur Gudmundsson. The amanda network backup manager. In *LISA*, 1993.

[9] Fanglu Guo and Petros Efstathopoulos. Building a high-performance deduplication system. In *USENIX annual technical conference*, 2011.

[10] ARCSERVE. Unified Data Protection. `https://www.arcserve.com/`, 2018.

[11] BACULA SYSTEMS. Bacula 9.2.1. `http://blog.bacula.org/`, 2019.

[12] COMMVAULT SYSTEMS INC. CommVault. `https://www.commvault.com/`, 2020.

[13] EMC CORPORATION. EMC NetWorker. `https://www.emc.com`, 2020.

[14] HEWLETT-PACKARD COMPANY. HP Data Protector. `https://support.hpe.com/`, 2019.

[15] Veritas. NetBackup 8.1.2. `https://www.veritas.com/`, 2019.

[16] Yaobin Qin, Brandon Hoffmann, and David J Lilja. Hyperprotect: Enhancing the performance of a dynamic backup system using intelligent scheduling. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2018.

[17] Eitan Bachmat and Jiri Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *ACM SIGMETRICS Performance Evaluation Review*, volume 30, pages 55–65. ACM, 2002.

[18] Richard Golding, Peter Bosch, John Wilkes, et al. Idleness is not sloth. In *USENIX*, pages 201–212, 1995.

[19] Ningfang Mi, Alma Riska, Xin Li, Evgenia Smirni, and Erik Riedel. Restrained utilization of idleness for transparent scheduling of background tasks. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 205–216. ACM, 2009.

[20] Avichai Giat, Dan Pelleg, Eran Raichstein, and Amir Ronen. Using machine learning techniques to enhance the performance of an automatic backup and recovery system. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, page 1. ACM, 2010.

[21] Peter M van de Ven, Bo Zhang, and Angela Schorgendorfer. Distributed backup scheduling: Modeling and optimization. In *INFOCOM, 2014 Proceedings IEEE*, pages 1644–1652. IEEE, 2014.

[22] Mark Chamness. Capacity forecasting in a backup storage environment. In *Proceedings of USENIX Large Installation System Administration Conference (LISA)*, volume 4, 2011.

[23] Ludmila Cherkasova, Roger Lau, Harald Burose, and Bernhard Kappler. Enhancing and optimizing a data protection solution. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*, pages 1–10. IEEE, 2009.

[24] Ludmila Cherkasova, Alex Zhang, and Xiaozhou Li. Dp+ ip= design of efficient backup scheduling. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 118–125. IEEE, 2010.

[25] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *FAST*, volume 2, pages 89–101, 2002.

[26] Benjamin Zhu, Kai Li, and R Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Fast*, volume 8, pages 1–14, 2008.

[27] Bin Lin, Shanshan Li, Xiangke Liao, Jing Zhang, and Xiaodong Liu. Leach: an automatic learning cache for inline primary deduplication system. *Frontiers of Computer Science*, 8(2):175–183, 2014.

[28] Sonam Mandal, Geoff Kuenning, Dongju Ok, Varun Shastry, Philip Shilane, Sun Zhen, Vasily Tarasov, and Erez Zadok. Using hints to improve inline block-layer deduplication. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pages 315–322, 2016.

[29] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 174–187. ACM, 2001.

[30] Phlip Shilane, Mark Huang, Grant Wallace, and Windsor Hsu. Wan-optimized replication of backup datasets using stream-informed delta compression. *ACM Transactions on Storage (TOS)*, 8(4):13, 2012.

[31] Yu Hua and Xue Liu. Scheduling heterogeneous flows with delay-aware deduplication for avionics applications. *IEEE Transactions on Parallel and Distributed Systems*, 23(9):1790–1802, 2012.

[32] Jianhua Sun, Hao Chen, Ligang He, and Huailiang Tan. Redundant network traffic elimination with gpu accelerated rabin fingerprinting. *IEEE Transactions on Parallel and Distributed Systems*, 27(7):2130–2142, 2015.

[33] Rubrik. Rubrik. `https://www.rubrik.com//`, 2020.

[34] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezis, and Peter Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Fast*, volume 9, pages 111–123, 2009.

[35] Wen Xia, Hong Jiang, Dan Feng, and Yu Hua. Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput. In *USENIX annual technical conference*, pages 26–30, 2011.

[36] Yaobin Qin, Brandon Hoffmann, Yuwei Wang, and David J Lilja. Exploring a forecasting structure for the capacity usage in backup storage systems. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0126–0134. IEEE, 2019.

[37] Christy Vaughn, Caleb Miller, Onyebuchi Ekenta, Hongtao Sun, Medha Bhadkamkar, Petros Efstathopoulos, and Erim Kardes. Soothsayer: Predicting capacity usage in backup storage systems. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pages 208–217. IEEE, 2015.

[38] Grant Wallace, Fred Douglis, Hangwei Qian, Philip Shilane, Stephen Smaldone, Mark Chamness, and Windsor Hsu. Characteristics of backup workloads in production systems. In *FAST*, volume 12, pages 4–4, 2012.

[39] Mike Dutch. Understanding data deduplication ratios. In *SNIA Data Management Forum*, page 7, 2008.

[40] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[41] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[42] Paul W Holland and Roy E Welsch. Robust regression using iteratively reweighted least-squares. *Communications in Statistics-theory and Methods*, 6(9):813–827, 1977.

[43] Yaobin Qin, Xianbo Zhang, and David J Lilja. Pbccf: Accelerated deduplication by prefetching backup content correlated fingerprints. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 146–154. IEEE, 2020.

[44] Min Fu, Dan Feng, Yu Hua, Xubin He, Zuoning Chen, Wen Xia, Yucheng Zhang, and Yujuan Tan. Design tradeoffs for data deduplication performance in backup workloads. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*, pages 331–344, 2015.

[45] Deepavali Bhagwat, Kave Eshghi, Darrell DE Long, and Mark Lillibridge. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 1–9. IEEE, 2009.

[46] Wei Zhang, Tao Yang, Gautham Narayanasamy, and Hong Tang. Low-cost data deduplication for virtual machine backup in cloud storage. In *Presented as part of the 5th {USENIX} Workshop on Hot Topics in Storage and File Systems*, 2013.

[47] Jaehong Min, Daeyoung Yoon, and Youjip Won. Efficient deduplication techniques for modern backup operation. *IEEE Transactions on Computers*, 60(6):824–840, 2010.

[48] Cezary Dubnicki, Leszek Gryz, Lukasz Heldt, Michal Kaczmarczyk, Wojciech Kilian, Przemyslaw Strzelczak, Jerzy Szczepkowski, Cristian Ungureanu, and Michal Welnicki. Hydrastor: A scalable secondary storage. In *FAST*, volume 9, pages 197–210, 2009.

[49] Biplob K Debnath, Sudipta Sengupta, and Jin Li. Chunkstash: Speeding up inline storage deduplication using flash memory. In *USENIX annual technical conference*, pages 1–16, 2010.

[50] Kiran Srinivasan, Timothy Bisson, Garth R Goodson, and Kaladhar Voruganti. idedup: latency-aware, inline data deduplication for primary storage. In *Fast*, volume 12, pages 1–14, 2012.

[51] Wenji Li, Gregory Jean-Baptise, Juan Riveros, Giri Narasimhan, Tony Zhang, and Ming Zhao. Cachededup: In-line deduplication for flash caching. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pages 301–314, 2016.

[52] Yoshihiro Tsuchiya and Takashi Watanabe. Dblk: Deduplication for primary block storage. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–5. IEEE, 2011.

[53] Vasily Tarasov, Deepak Jain, Geoff Kuenning, Sonam Mandal, Karthikeyani Palanisami, Philip Shilane, Sagar Trehan, and Erez Zadok. Dmdedup: Device mapper target for data deduplication. In *2014 Ottawa Linux Symposium*, 2014.

[54] Zhuan Chen and Kai Shen. Ordermergededup: Efficient, failure-consistent deduplication on flash. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pages 291–299, 2016.

[55] Ahmed El-Shimi, Ran Kalach, Ankit Kumar, Adi Ottean, Jin Li, and Sudipta Sengupta. Primary data deduplication—large scale study and system design. In *Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12)*, pages 285–296, 2012.

[56] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (TOS)*, 7(4):14, 2012.

[57] Avani Wildani, Ethan L Miller, and Ohad Rodeh. Hands: A heuristically arranged non-backup in-line deduplication system. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 446–457. IEEE, 2013.

[58] Dirk Meister, Jürgen Kaiser, Andre Brinkmann, Toni Cortes, Michael Kuhn, and Julian Kunkel. A study on data deduplication in hpc storage systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 7. IEEE Computer Society Press, 2012.

[59] Yaobin Qin and David J Lilja. Adaemb-encoder: Adaptive embedding spatial encoder-based deduplication for backing up classifier training data. In *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2020.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[61] Junwei Han, Dingwen Zhang, Gong Cheng, Nian Liu, and Dong Xu. Advanced deep-learning techniques for salient and category-specific object detection: a survey. *IEEE Signal Processing Magazine*, 35(1):84–100, 2018.

[62] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[63] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[64] Sarah M Erfani, Sutharshan Rajasegarar, Shanika Karunasekera, and Christopher Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58:121–134, 2016.

[65] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[66] Guangping Xu, Bo Tang, Hongli Lu, Quan Yu, and Chi Wan Sung. Lipa: A learning-based indexing and prefetching approach for data deduplication. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 299–310. IEEE, 2019.

[67] Yaobin Qin, Bingzhe Li, and David J Lilja. Enhancing the ensemble of exemplar-svms for binary classification using concurrent selection and ensemble learning. In *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 673–682. IEEE, 2018.

[68] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[69] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[70] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[71] Ronald Rivest and S Dusse. The md5 message-digest algorithm, 1992.