Performance Trade-offs in Dynamic Backup Scheduling

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

Brandon Hoffmann

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Dr. David Lilja

May, 2015

Acknowledgements

Throughout this paper I will occasionally use the word "we" to discuss design decisions or ideas. Though this thesis and the simulator are written by myself, many of the contributions to this project were not possible without the advice, guidance and ideas of multiple additional people.

Dr. David Lilja

My adviser. Every major idea I had went through him and he himself contributed or inspired many more great ideas and project decisions. His expertise from the statistical and performance analysis side of things was fundamental to the success of this project.

Aaron Christensen of Symantec Corp

My self-proclaimed mentor. Aaron was my window into the backup world. Whenever I had questions or ideas on the backup side of things he is who I turned to. He also opened many doors for me in obtaining information, hardware, software and contact with other industry experts. His contributions can not be understated.

Dedication

To my family and friends who have helped keep me sane through this process. And to my adviser for always being helpful and understanding through my many shortcomings.

Abstract

Recently the amount of data generated and stored on computers has seen outrageous growth and the trend will only continue. With the 24 hour global business structure being the way it is now, backup windows are shrinking and/or data is expected to be available at all times. Because of this, having effective and efficient data protection has become increasingly important. It is therefore necessary to move past the outdated static backup configurations and adopt intelligent dynamic backup systems. With that in mind we introduce the Affinity dynamic backup scheduling algorithm. Using a dynamic backup simulator we examine this algorithm as well as others and examine the performance trade-offs between these algorithms. Using this algorithm we have seen incremental improvements in the three primary metrics of Storage Throughput Utilization, Storage Distribution and Backup Time Consistency. With the insight gained from our simulation we discuss the benefits and trade-offs of dynamic scheduling algorithms and also dive into ideas and changes important to the future of backup systems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Throughout the world we have seen an extraordinary increase in digital content creation and storage. Effectively and efficiently backing up the vast amounts of data stored by large enterprise companies is becoming extremely challenging yet of the utmost importance. As the systems increase in size they also inherently become more complex and challenging to manage efficiently. Traditionally this complexity was left for backup administrators to handle by designing elaborate static schedules and assigning individual storage to each backup. This has flaws however because it is extremely challenging for the administrators to design a system that is efficient and even in the case that they succeed, the system is constantly in flux and the elegant schedule drifts into chaos. It is because of these issues that it is so necessary to switch to a intelligent dynamic scheduling system that requires less complex set-up, adapts effectively to a changing system and reacts immediately to unforeseen events such as server failures.

An example of a basic backup environment is shown in Fig. 1.1. In it we have clients, media servers, a master server and the network. Clients are the computer systems that have data that needs to be backed up. A backup policy is used to indicate what files are to be backed up. We will simply refer to this policy as a backup as we assume each policy to indicate one backup event. The clients have to send their data across the network, (which for our purposes remains a black

box) over to the media servers. The media servers are where the backup data is stored. Media servers can have a wide variety of storage from direct attached hard disks to RAID or tape arrays on a Storage Area Network. The master server (for which there can be multiple) is essentially the brains of the system. It controls when backups start and holds all the backup configuration information. Each master server serves a set of media servers and clients. Larger backup systems are partitioned into groups, each with an assigned master server to control them.

In this paper we present the following contributions:

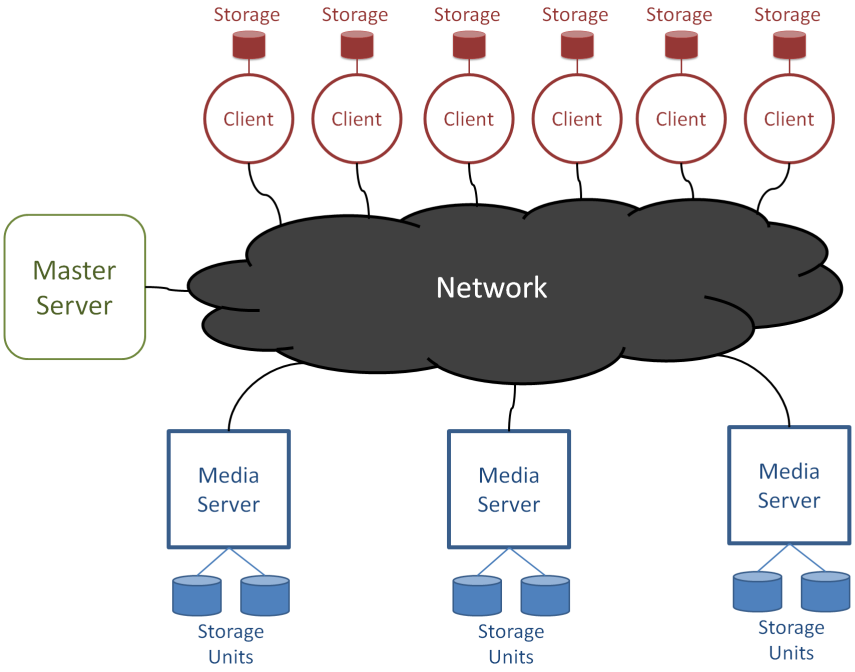


Figure 1.1: A simple diagram depicting a basic backup environment.

Additional Metrics While other work has previously focused solely on backup time, we feel that great benefit can be gained by investigating additional metrics:

Storage Throughput Utilization

The most efficient way to backup as much data as possible is to fully utilize the available storage throughput. Due to this and the fact that throughput utilization normalizes well for highly randomized systems, we feel that this is the best way to express backup efficiency. This metric is in replacement of the typically used backup time because with randomly generated systems, normalization is important. Storage throughput utilization is defined by Eq. 1.1.

$$Storage\ Throughput\ Utilization = \frac{\frac{total\ data\ size}{total\ time}}{storage\ throughput} \tag{1.1}$$

Storage Distribution

In dynamic backup systems, backups have access to multiple storage units. As a side effect of this, different incremental backups of the same backup policy tend to be spread across multiple storage devices. When it comes time to restore that data, having different incremental pieces spread across multiple storage devices can have a major impact on restore times. Therefore it is beneficial to examine the distribution of storage devices being backed up to. We examine storage distribution by looking at the distribution of the frequency that a backup-storage pair is utilized. We examine it both graphically and also look at the mean and standard deviation.

Backup Time Consistency

It is important for backups to be consistent and predictable. Having reliable performance allows companies to better predict and work around downtimes. This metric is examined by looking at the distribution of different overall backup times for different days on the same system.

Simulation Environment We introduce a new simulation environment that we use to test the scheduling algorithms. This simulation approach provides much more flexibility and depth for our tests.

Incremental Workloads A lot of previous work focuses on full backups because dynamic algorithms work very well for full backup workloads. Our work focuses on the previously neglected incremental workloads which we feel is extremely important given that the growth rate of data and migration to the cloud can make full backups less feasible.

Affinity Algorithm On top of examining other algorithms we introduce our Affinity Dynamic scheduling algorithm that creates affinities between backups and storage. These Affinities are used to slowly push the system to a steady state.

The rest of the paper will be organized as follows:

- Chapter 2 outlines some trends in the backup world at the moment and why they are good or bad as well as discuss important new trends.
- Chapter 3 discusses some related work on backup systems.
- Chapter 4 describes the importance of shifting from static to dynamic backup scheduling.
- Chapter 5 gives a detailed description of the proposed Affinity algorithm as well as the alternatives.
- Chapter 6 introduces the simulator created to test the algorithm proposed.
- Chapter 7 details the parameters and assumptions used in our testing.
- Chapter 8 presents the results of our simulations and analysis.
- Chapter 9 discusses future work that we would like to tackle.
- Chapter 10 concludes.

Chapter 2

Current Trends

Computer technology is constantly evolving and with it backup technologies and strategies also evolve. Backups started out as simple as making a second copy of a punch card and have evolved to giant software and hardware systems that are constantly working to insure no data is ever lost while also minimally impacting operations. In order to make meaningful progress it is important to understand how backups have evolved, how they are currently being used and how things will change in the future.

2.1 Storage Technologies

Storage technology in backups usually tends to lag behind the state of the art because in general all that is desired from a backup system is that it is there in case of a disaster and that it is not in the way of production. Because of this, backup environments are often designed to obtain acceptable performance for the cheapest cost which tends to be older technologies.

In the 1960s the introduction of the tape drive was extremely important to backups because with their high capacity and reliability it allowed the introduction of frequent and scheduled backups. Tape drives were revolutionary for their time but have mostly faded away except for in backup environments where they are

still commonplace. This is because tape drives still deliver excellent size for their price as well as good sequential write performance. Tape drives still offers sizes comparable to the largest disk drives but usually at a cheaper cost and since backup systems often ignore random read performance, tape drives are good on a budget.

As new storage technologies entered the market they slowly worked their way into backup environments. The introduction of floppy disks allowed for backups of small important files as well as an increase of backups by home users. After that disk drives were introduced and became commonplace in computers but still a rarity in backup systems because the cost to size ratio was way too high. Over time disk drive cost to size ratio slowly decreased to the point where disk drives were started to be used more often. This was also aided by the introduction of RAID technology which distributed data across multiple hard disks to increase the performance and reliability of relatively inexpensive disk drives. Additionally, disk drives could be used as temporary staging storage where the data is written to the disk drives at higher performance and then written on to the tape drives afterwards. In some cases this is used in order to maximize the throughput of the tape drive by ensuring steady streams of data to the tape drives. Currently there is no industry standard it is all about what the user prioritizes or just preference. You will find systems of only tape drives, only disk drives or a mixture of both.

Recently solid state drives have become the norm in a lot of computer systems. Initially used for operating systems they have now become cheap enough that it is common to find entire computers storing to a solid state drive. These solid state drives drastically outperform disk drives and tape drives in read and write performance but still are out classed in the cost to size ratio. It is because of this cost that solid state drives are rarely found in backup environments and if they are it is most likely as staging disks. As technology continues to improve, the price point of solid state drives is dropping fast and they will eventually become a staple in backup environments of the future.

2.2 Backup Networking

Networking is of huge importance in backup systems because not only are you trying to move a large amount of data across the network but in most cases you are trying to move data from many client sources and move it to significantly fewer storage servers. Luckily, network bandwidths have been steadily improving and in most cases out perform the storage devices.

In the earlier days of backups, a backup was done to an additional storage device attached to the computer because the idea of Local Area Networks (LANs) was not yet a reality. As LANs were invented it was natural to move all the backup storage to a central location separate from the backup clients. With the invention and popularization of the Internet many backups were done to off-site locations in order to provide additional security in the event of fires or other major disasters. However, the bandwidth of the Internet in most areas is much slower than storage devices and LANs decreasing the overall potential throughput. Also, backing up over the Internet is less secure and needs to be encrypted which can add to backup and restore times as well. It was not until the last few years that increased Internet bandwidth and the popularity of the "Cloud" that consumers and large enterprise companies are starting to return to the idea of backing up over the Internet.

Another important networking component of backup systems are Storage Area Networks (SANs). SANs are dedicated networks that only connect storage devices to servers that are accessing them. They are very common in large enterprise environments where the servers themselves are not large enough to store all the data and are instead attached to a SAN with disk arrays or tape libraries attached as well. SANs are traditionally set up such that multiple servers share the collection of storage available in the SAN and bypass many of the shortcomings of the traditional direct-attached storage.

2.3 Backup Configurations and Methods

Backup has evolved greatly from just making a full copy of the data you want protected. While full backups are still the safest form of backup, there are many other variations of backup being used today:

Differential

In the case of a differential backup, an initial full backup is created, then on every subsequent backup, every file that has changed since the full backup is backed up. Because this backup will get larger the longer it has been since the full backup, differential backups are most often done daily with a full backup on the weekend to reset the growth. In comparison to full backups, differential backups offer faster backup times with the trade-off of slower restore times because of the overhead of combining the differential chunk with the full backup.

Incremental

Incremental backups are very similar to a differential backup but instead of backing up everything that has changed since the last full backup, only the data that has changed since the last backup of any type is backed up. This removes the problem of the backups growing in size as you move away from the initial full backup. There are two different types of incremental backups: forward and reverse. In the case of a forward incremental, the changed data is backed up and stored separately from the full backup. In the reverse incremental the full backup data chunk is updated so that the most recent backup is up to date, but it also saves a backup of all the updated data for that backup in order to restore to in the future. The benefit of incremental backups is that the size of the backup is as small as it can be and therefore this is the fastest option for backing up. However, its trade-off is that in order to restore it must combine multiple separate chunks of data together. In terms of forward vs. reverse, forward restores faster the closer the restore date is to the full backup and reverse restores slower the farther back you

are restoring from. Reverse backups are also slower to backup because it has to update and store an additional copy every backup.

Synthetic

Synthetic backups are the artificial creation of a full backup by taking a full backup and a set of forward incremental backups and creating a new full backup by making the changes of each incremental to the original full backup. This is extremely useful because it allows you to get the backup benefits of incremental or differential backups but reduce the restore strain by adding additional full backup points via synthetic backup. While synthetic backup is usually far faster than a full backup, it should be noted that it does take time to aggregate and duplicate all the data and therefore synthetic backups are commonly done on the weekends as a replacement for weekly full backups.

2.4 Real World Systems

One of the biggest obstacles when designing a dynamic backup scheduling system is adapting to any system that it is placed on. There are no industry standards for backup systems. Everyone does it slightly different, has different needs, different schedules and are operating on different scales. The fact that there is so much diversity in system configurations is the biggest challenge of dynamic systems. With no consistency, no wide-sweeping assumptions can be made and therefore algorithms need to be either very simplistic and high level or categorized for different types of systems. Some companies such as Symantec sell all-in-one style hardware and software backup appliances [3]. These all-in-one style boxes are designed to be as easy to set-up as possible and introducing the idea of dynamic scheduling to this would increase this set-up ease.

Chapter 3

Related Work

Data protection and backup has been a major area of research for a long time and therefore there is a plethora of related work.

3.1 Cloud Backups

Cloud backup involves backing up data by sending that data over a public network to an off-site server. The main difference to traditional backup, especially in enterprise, is that potentially important proprietary information is being sent across a much less secure public network. In many cases however enterprise companies choose to limit their use of cloud backup to non-essential data. Cloud backups suffer from the same major issue of normal backups in that all the traffic is during a short period of off business hours. [4] discusses a scalable architecture for handling the cloud backup workload.

3.2 Deduplication

Data deduplication is an extremely common and important research area related to backups. In deduplication the data is broken down into chunks and when duplicate chunks are encountered, only the first is stored and the subsequent

occurrences store a pointer to the old chunk. This saves a tremendous amount of storage space because the deduplication/compression ratio can surpass 90% in some datasets [5]. Not only is deduplication useful for saving storage space it also can increase effective write bandwidth. In some systems, deduplication occurs on the client side machine which allows the data to be compressed significantly prior to being sent across the network. This idea is becoming more important as cloud backup systems with lower relative network bandwidth start to become more common.

The drawback of deduplication is that the restore performance is quite poor because the chunks are scattered or fragmented all across the storage device. This drawback is usually swept under the rug with the argument that restores are much less common than backups. While it is true that backups are much more common, increasing restore performance in deduplication environments [6] is also an important research area.

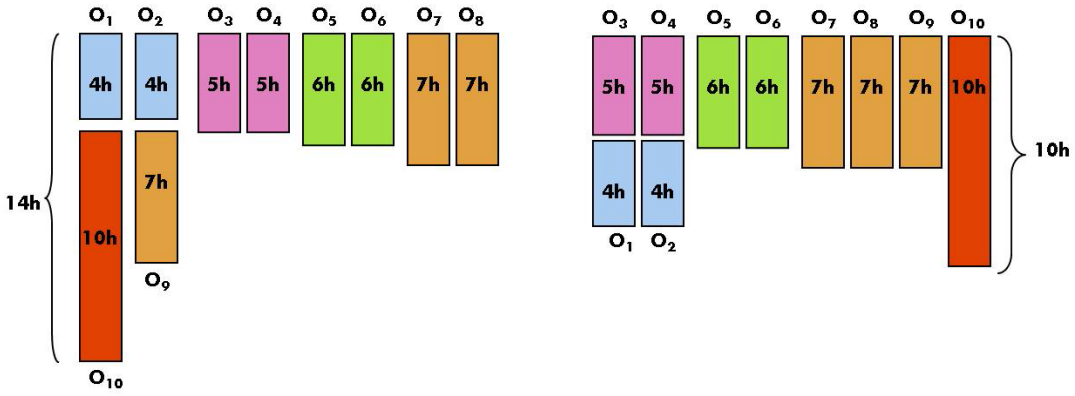
3.3 Disaster Recovery

There are many different levels of backup that cover different failure levels. For example, backing up your Laptop to a flash drive that is stored in your laptop bag will protect you from your laptop hard drive failing but not from your laptop being stolen with the bag. In a general sense, the farther removed the location of the backup destination, the more effective it is at recovering from larger disasters.

Data-centers tend to be one of the biggest examples of this being the case. It is not uncommon for entire data-centers to be wiped out via fire or natural disaster leaving some companies forced into bankruptcy. Research such as [7] attempt to minimize disaster backup windows of data-centers distributed across large areas. Doing so involves breaking the problem into directed graphs and then use greedy heuristic algorithms to attempt to find efficient backup destinations and routes.

3.4 Longest Backup First

Cherkasova, L. et al. [2] proposed the simple but effective idea of backing up the longest processing backups first to more efficiently schedule backups . While their investigation was solely focused on full backups to tape drives, the longest backup first idea is an important building block to which this work is based. That being said, they failed to see the importance of using a dynamic system on incremental backups as well and also neglected the storage distribution and consistency metrics. Fig. 3.1 is an example diagram taken from [2] that very clearly demonstrates the inefficiencies of not scheduling properly and how it can be improved.



(a) Suboptimal backup scheduling. (b) Improved backup scheduling.

Figure 3.1: Demonstration of inefficient backup scheduling from [2].

In the same paper Cherkasova, L. et al. also present very important ideas on balancing backup groups to reduce fragmentation of backups across different tape drives as well as reducing concurrent writes to the same tape in order to improve restore performance.

Cherkasova, L. et al [1] later went on to create a pre-processing bin packing scheduling algorithm that involves using historical data to determine the optimal schedule prior to any backups occurring. It is important to note that while this can create very efficient backup schedules, it suffers from the potential of extremely

long solution determination time and also the inability to adapt to server failure or restores during the backup process.

3.5 Markov Decision Process Backup Scheduling

Not only is it important to schedule which backups should backup before others but determining how often and when to backup is also a heavily researched topic. In [8] they use Markov Decision Process to create reward functions based on the priority of the data and backup resources available. Research of this type focuses on both Recovery Point Objective (RPO) and Recovery Time Objective (RTO). RPO is essentially a point in time that the backup system is expected to be able to recover to. In the context of [8] they use the number of backups that can be skipped. RTO represents the speed at which the data can be recovered. They use the number of incremental backups before a full backup must occur as a heuristic for the RTO. Then, using the Markov Decision Process, decisions are made as to what data sources are allowed to use the backup resources and whether they are incremental or full backups, based on data priorities and varying RTO and RPO values for data sources. A similar technique was also used in [9]. Techniques from this research area are very relevant when doing dynamic backup scheduling and setting priorities based on which backups are allowed to be postponed to the next backup day.

Chapter 4

Dynamic Transition

In many current enterprise backup environments the process of defining when a backup should initiate and to where it should initiate is statically defined by the backup administrator. However, this static configuration has multiple shortcomings. For one, it is very challenging to create an efficient backup configuration and manually defining a static system will lead to many inefficient systems. Additionally, even if the system is configured well, most configurations are not re-visited after initial configuration and suffer from a slow deterioration into chaos over time as backups are added, removed or modified. Lastly, while backup times can be approximated, the daily churn of the system, especially in incremental backups, can lead to varying backup times across the board. Because of this variance in backup times it is impossible to create a schedule that will be efficient everyday because it does not take into account the backup time fluctuations.

The solution to this static configuration problem is to introduce a intelligent dynamic scheduler. A dynamic scheduler gains the benefit of being able to react to the current state of system and adapt to the variation in backup times to schedule more effectively. The dynamic scheduler also has access to historical data of previous backups and is able to use that to more accurately predict the backup performance in order to efficiently allocate resources. Finally, the dynamic system also is able to adapt to unexpected occurrences such as a failed server or

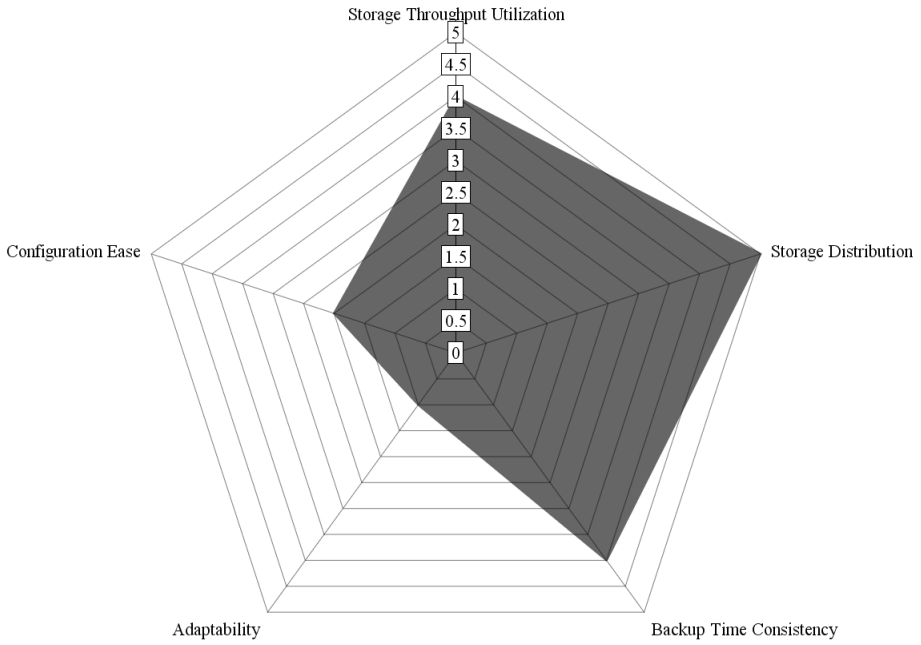


Figure 4.1: Overall performance graph of a static system. (Hypothetical)

a series of important restores being started and is able to reallocate the resources effectively.

Fig. 4.1 shows a radar graph containing the three important metrics discussed in Chapter 1 as well as two additional important characteristics to look at, Configuration Ease and Adaptability. Prior research has focused solely on backup speed or storage throughput utilization without examining the full picture. Fig. 4.1 gives a reasonable idea of what you can expect from a static scheduling algorithm. However in Fig. 4.2 we see the potential of a dynamic system. While we lose some in Storage Distribution, we more than make up for it in the other metrics.

As mentioned above, setting up a system is an extremely daunting task to the

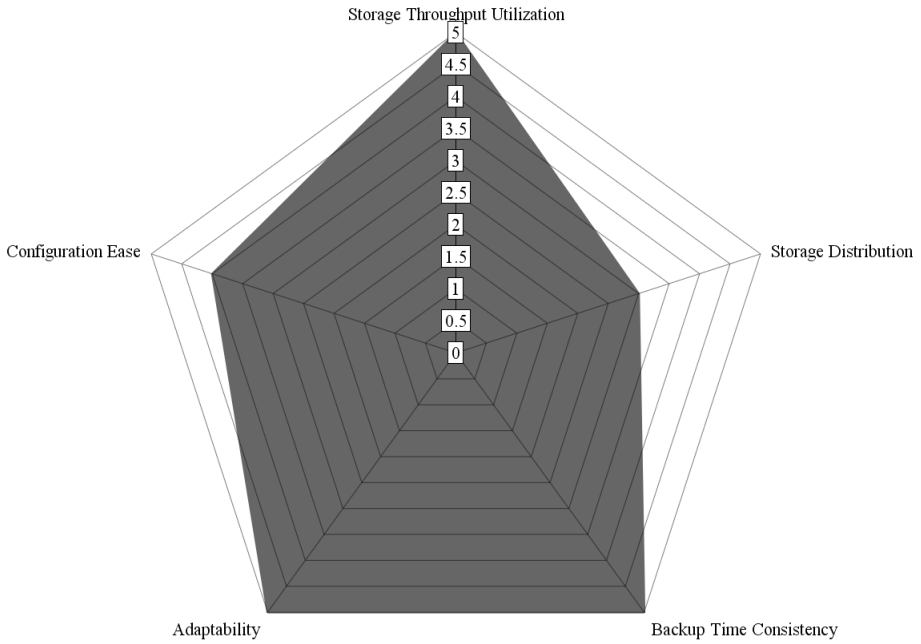


Figure 4.2: Overall performance potential graph for a dynamic system. (Hypothetical)

point of being more of an art form. Another benefit of the dynamic system is it greatly simplifies the initial configuration. With the scheduling aspect mostly removed from the equation, the administrators will just need to choose applicable storage devices for the backups. In order to allow for important backups to be backed up first, there is also priority settings that allow backups of great importance to be given priority resources and start times. The combination of all these benefits makes a dynamic system an obvious next step for backup systems.

Chapter 5

Algorithms

5.1 Affinity Dynamic Algorithm

The Affinity Dynamic Algorithm is designed to take a chaotic system and bring some consistency to it. To do this it notates the performance of backup-storage pairs on previous backup days and uses that information to create priority options for the backups. Doing this reduces the variation in storage chosen by the backups each day while also allowing additional options if the primary storage is not available and the backup is of high priority.

The algorithm was designed with simulated annealing in mind, in the sense that as the options are experimented with there eventually comes a relatively steady solution that is near optimal. In order to do that, the algorithm sometimes randomly chooses what it considers to be a sub-optimal choice. This is to avoid getting stuck in potential suboptimal configurations based on limited experimentation. The problem with incremental backup workloads is that it also needs to be adaptable and because of this the algorithm will attempt to use the priority storage when reasonable but will allow other storage devices as options when necessary. By attempting to reach a relatively steady state it will cause the storage distribution to decrease while also increasing the backup time consistency. It is also hypothesized that by using this high level approach of examining

only the performance to a given storage device, it in actuality examines the entire data path and this algorithm could help re-route backups away from storage with poor or saturated network paths. Future work needs to be done to examine this potential.

5.1.1 Historical Data

The Affinity algorithm tracks of the overall throughput of the previous backups of the same policy as well as the storage device the backup was to. In order to adapt to a changing system and also remove outliers, the historical data is set to expire after a set number of days. In the experiments performed the historical data expiration time was set to 25 days.

5.1.2 Backup Selection

In order to select the best backup to start, the following actions occur:

Backup Priority

First an ordered list of backup options is produced in descending order based on how long the backup is expected to take. This is determined by the size of the backup on the given day divided by the average of the historical throughputs for that backup. This yields a rough estimate as to how long a backup will take on this given day and allows us to prioritize longer backups first. It is very important to note here that we assume that all backups use a feature similar to Netbackup's Accelerator which keeps track of what files are changed as they are being changed as to nearly completely remove the overhead of determining changed files in incremental backups. We also assume that the size of the backup is known before being prioritized.

Storage Priority

Next, an ordered list of eligible storage units is produced where eligible is defined as having less active backups than the defined maximum. For the experiments performed this was always 5. This list was ordered in ascending order based on the expected time remaining to complete all of its active backups. This causes the storage unit with the least time remaining to be given priority for receiving the next backup.

Affinity Sorting

The top storage unit from the storage priority list is selected and presented with the top 5 (or less) backups from the backup priority list that are also eligible to backup to that storage unit. This list of 5 (or less) is then sorted in ascending order based on the Affinity with the given storage device. The Affinity is defined as the number of storage units that historically perform better than the currently chosen storage unit. By making this Affinity based on relative performance position, it allows it to adapt well to systems of any size.

Once the short list of backups is sorted, a backup is chosen based on a triangular distribution. A triangular distribution provides an easy to understand and effective way to prioritize the highest Affinity backup but also introduces some random selection into the system. A triangular distribution is also simple but adapts well when less than 5 options are available for selection which is important in scenarios with less storage options per backup. The triangular distribution used in our testing is shown in Fig. 5.1 and is produced using values:

$$a = 0 \tag{5.1}$$

$$b = 5 \tag{5.2}$$

$$c = 0 \tag{5.3}$$

The percentage that each option is chosen is outlined in Table 5.1. While the triangular distribution is used in all experiments performed, investigating other

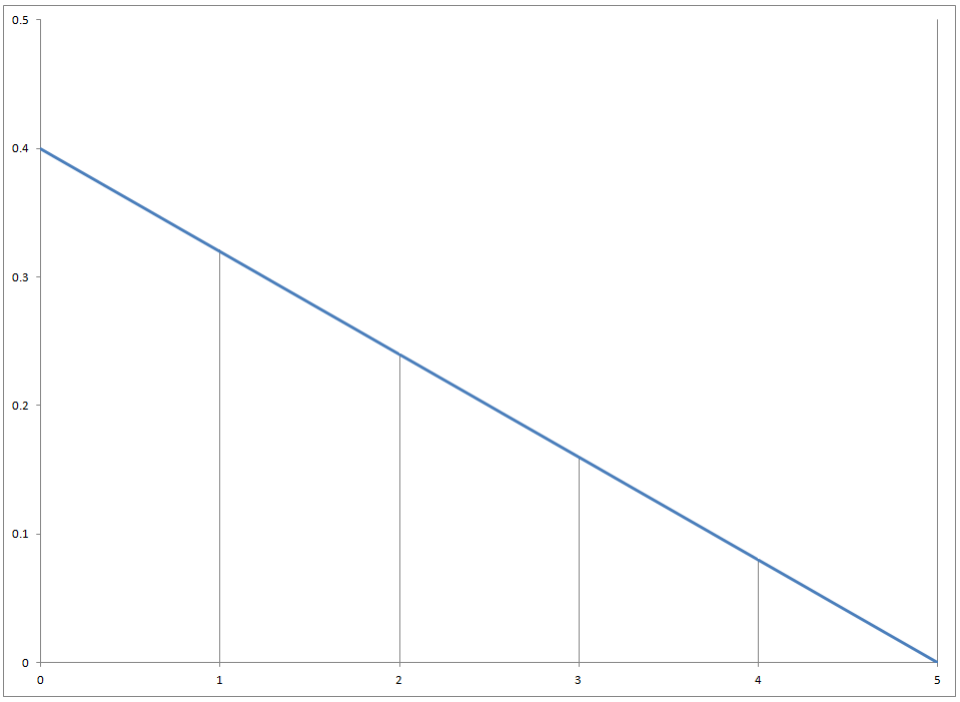


Figure 5.1: Affinity selection distribution graph

distributions is a good area of future work.

Storage Option	1	2	3	4	5
Frequeny (Percent)	36%	28%	20%	12%	4%

Table 5.1: Affinity selection numerical distribution

5.2 Alternatives

The two algorithms we choose as a comparison are the Longest Backup First algorithm and a Random algorithm.

Longest Backup First

Longest Backup First (LBF) was chosen as a good comparison is because

it is the best known algorithm that makes decisions as they appear rather than relying on a pre-processed schedule. We felt that for the incremental workload being investigated, a pre-processed schedule would take a very long time to obtain which is not viable when the size of the backups are not known well in advance. Pre-processed schedules also do not adapt well to any sort of unexpected performance or event, which we feel is vital for maximizing performance in a incremental workload.

Random

A form of Random algorithm was chosen as a second comparison to give a reasonable example of how bad the system can get. It is mostly useful for a good comparison for storage distribution in that it chooses the storage destination completely randomly.

5.2.1 Longest Backup First

The longest backup first algorithm is taken from [2] and is used as a good comparison to the Affinity Dynamic Algorithm. Because the algorithm was only examined for small sets of large full backups, we felt it was also useful to investigate the effectiveness of the algorithm for our very different incremental backup workload.

In the Longest Backup First algorithm they kept track of the total processing time or total backup time for each backup and used that as a prediction to the backup time because for Full backups, the system changes slowly and this prediction performs well. In our slightly different variation, we instead keep track of the backup historical throughputs and also allow the algorithm access to the size of the backup for that day during the prioritizing period. This essentially breaks down the algorithm to being the same as our Affinity Dynamic algorithm but without the randomization and storage affinity being applied in selection.

5.2.2 Random

For the random algorithm the goal was to give an idea of how bad the system can get. It is impossible to directly compare static and dynamic backup systems because static systems cannot be generalized like the dynamic ones. For example, it is possible for a static system to perform exactly the same as the random system if the static system was somehow scheduled before hand to magically line up with the dynamic scheduling. However, at the same time the static systems can also be poorly designed, extremely inefficient and take drastically more time to backup than the dynamic system. The random system was designed in an attempt to emulate the randomness of a static system that can sometimes perform well but also very poorly. It was also beneficial to have a baseline for the storage distribution metric with an algorithm that simply randomly chose a storage device with no intelligent reason.

The Random algorithm is quite simple, the total number of active backups in the system was capped (in our experiments this was 15) and until that cap is reached, the scheduler randomly chooses a backup and an eligible storage device (random tests still used available storage options) and started a backup. This algorithm, while still dynamic by nature, keeps no historical data and uses no intelligent decision making.

Chapter 6

Dynamic Scheduling Backup Simulator

Data protection and backup is important to every company whether it be client information or just company earnings data necessary for tax season, every company has some data they need protected. An obvious outcome of this is that backup systems vary greatly in size, complexity, usages, strategies, scheduling, etc. With systems as varied as this it is often difficult to obtain and test real systems and accurately capture this diversity in analytical modeling. Therefore we made the decision to develop a simulator to aid in the process of developing and testing different systems and algorithms. In this chapter we will dig into details about the simulator we have developed, its current functionality and structure and how it is being used, in order bring clarity to our test environment. The chapter will be structured as follows:

1. Introduction to the simulator's base design structure
2. Description of the random generators
3. Creating custom scheduling algorithms
4. Future functionality

6.1 Simulator Introduction

The simulator itself is written and java and is designed to be a simple throughput based simulator with easy set up, meaningful logging and ability to create customized algorithms for testing. In addition, side backup system generators were created in order to create input files that are random but also emulate a realistic backup system.

6.1.1 Major Classes

Each major component of the backup system is created as an object in the overall **BackupSystem** object:

- **Backup**: This keeps track of how much data is to be backed up and what client it is attached to. When the backups are initiated it keeps track of progress and where it is backing up to.
- **Client**: This is used as to keep track of the max read throughputs of the backups. It is treated as a single storage device computer.
- **MediaServer**: This holds a set of storage devices and is mainly in place for when networking is implemented.
- **StorageDevice**: This keeps track of a throughput value and when backups start to it keeps track of which backups are backing up to it.

These classes are created based on an input file and constraint file that are parsed in upon creation of the backup system. An example input file and constraint file can be found at Appendix B.1 and Appendix B.2 respectively. The overall system is made up of these objects interacting with each other and upholding the constraints imposed by the constraint file. These constraints control backup restrictions such as when backups can start, when they are expected to be completed and to which storage devices they are allowed to connect to.

6.1.2 Time-Driven

The simulator is time-driven in that there is a time counter that keeps track of what time it is and the simulator is driven by telling the system to progress the time forward by a small chunk of time. For example, the overall loop directs the system to step forward by an amount of time "t" repeatedly until the system indicates that all backups have completed. Within each time step many actions and checks occur:

First

The first thing that is checked is whether the system requires new backups to start. This is handled by the scheduler which gets defined and passed into the system. More on the scheduler in Sec. 6.3. The scheduler is given the system status and asked to return a list of backups to start and to what storage device. The backups listed are then started and all necessary flags are set.

Second

The next thing that occurs is each backup is assigned a throughput for this time step. This is done by looking at the storage device the backup is being written to and being read from. Essentially the storage device first generates its throughput for this time step based on its base throughput and its throughput variation parameters. After this number is obtained, a fraction equal to its throughput divided by the number of active backups assigned to it is determined and offered to the backup with the lowest calculated client throughput. This backup either takes all of the offered throughput or less if the client can not handle that high of throughput. The remainder is then re-calculated and redistributed to the rest of the backups, in lowest client throughput order.

Third

After all the throughput values are set, the simulator goes through to each

backup and increments its progress by the throughput multiplied by the size of the time step. During this process it checks if the backups are completed and if so makes the necessary changes to the backup and storage device classes and logs it into the output log.

Fourth

Finally, after all the backups are successfully incremented, the time is changed and the time step concludes.

6.2 Random Generators

6.2.1 Random Backup System Generator

For our testing it was important to be able to test a variety of systems. Due to lack of access to detailed enterprise backup configurations, a random generator was created in order to create realistic systems of various sizes and shapes for use with our simulator. For diversity in our testing it was also important to be able to generate multiple random systems of the same size. The generator allowed us to accomplish this diversity. The generator is by no means complex and is simply a large set of input variables used to determine distributions or number of specific objects such as backups. For a full list of input variables please see Appendix A. The most important variables are the number of storage units, clients and backups per client as those are the variables varied most often in the experiment.

Throughput values and backup sizes were randomized using distributions. For client and storage unit throughput we went with a basic Gaussian distribution. Every system will be completely different as far as variety of storage devices on clients and media server storage and therefore we felt it appropriate to give some variation but stick to a Gaussian distribution. For backup sizes we used a Pareto distribution. While obviously this does not hold for all backup environments in general we find that most backups are relatively small and close in size with a smaller subset of them being very large. It is important to keep this assumption

in mind when interpreting the results.

6.2.2 Random Constraint Generator

For our testing we also wanted to examine the effects that the number of storage options had on the backup performance. To do this we added constraint files which limited which storage devices a backup could choose from. Because constraint files require a system to be generated they are tied to a specific system file. The generator itself simply assigns a random subset of storage options to the backup, based on the inputted number of allowed storage options.

6.3 Custom Scheduling Algorithms

Arguably the most important part of the simulator is the ability to create customized algorithms and see how the same system reacts. This is accomplished by creating a template class **Scheduler** for which a subclass is created for each tested algorithm. The **Scheduler** class has a set of methods which must be fulfilled in order for the scheduler to be effective but most of the functionality is implemented in the *getNewBackups* function that returns the new backups to start given the current system conditions. The simplicity of this **Scheduler** class allows for a ton of variety in algorithms because you have complete access to everything in the backup system including allowed start times and storage constraints. This functionality allows for easy iteration of algorithms during future work.

6.4 Future Functionality

- Networking between **Media Servers** and **Clients**.
  - It should be noted that the simulator currently assumes an infinitely fast network. We realize that this is an extremely important part of backup performance but also understand that performance in cases

where the network is not bottlenecking the system is also very relevant. We therefore assume a fast dedicated network that is never the bottleneck in our test systems.

- Priority backups
- Random server failures
- Random restores during backups

Chapter 7

Experiment Set-Up

The experimentation was done using the simulator described in Chapter 6. We created fractional factorial experiment with the different primary factors and levels explained below:

Algorithms

The first thing that we will vary is the different algorithms that we are trying to compare.

Levels: Affinity Dynamic, Longest Backup First, Random

As mentioned Chapter 5, the Longest Backup First algorithm was chosen due to it being the best comparable alternative and the Random algorithm was chosen as an attempt to emulate a poor statically defined system. For more details on the algorithms chosen please refer back to Chapter 5.

Number of Backups

Modifying the number of backups is an important factor to investigate how each algorithm would react to the size of the system changing.

Levels: 100, 200, 300

These levels represent a relatively medium sized backup set and were chosen to try to focus on a middle ground between very small and very large backup environments for this experiment.

Number of Storage Units

The number of storage units lets us investigate how different algorithms react when the system becomes slightly more complex and also what effect adding more overall throughput may have.

Levels: 2, 4, 6

These levels were chosen to allow a variety in the number of storage options but also a variety in overall backup throughput. They were also chosen to create realistic backup to storage ratios.

Number of Storage Options

This is the number of storage units that are eligible to backup to for a given backup. Each backup was given a set number of options that were randomly generated for each constraint configuration.

Levels: 1, N/2, N

The mathematical nature of the levels were chosen in order to have a set number of levels while also accounting for the fact that the number of system storage devices changed. In general we feel that for a dynamic system it will often need to rely on fractions of total rather than absolute values in order to adapt to different system configurations.

Incremental Backup Sizes

Each day the size of the incremental backup was recalculated based on the total size of the given backup. The incremental size was determined by a Gaussian distribution with a mean of 15% and a standard deviation of 5%.

It needs to be noted that this test environment is far from inclusive. We have focused primarily on a medium sized system with relatively homogeneous storage performance and backups whose sizes are represented by the Pareto Distribution. Different users with different needs will have very different system configurations and this testing only covers a very small subset of the potential options. It is also important to re-iterate that no networking was taken into account during our

simulation. Therefore we assume an infinitely fast network that never delays the storage processing. We feel that this assumption is reasonable as for an environment where the network is significantly faster than the overall storage throughput the networking will have a small effect that is independent of algorithm and that the effect is negligible.

For more information about how the systems were generated please refer back to Chapter 6 and for default parameters see Appendix A.

Chapter 8

Results

The experimental results were taken from the simulator described in Chapter 6. This data was obtainable via the logging system in the simulator that logs important backup events with timestamps. With the mass amount of data and metrics collected by the simulator it is necessary to focus on a subset that is the most interesting and telling. To do so we will examine the three primary metrics one at a time and draw conclusions from the data presented. First a brief explanation of how each metric is obtained:

Storage Throughput Utilization

Each trial ran for 60 concurrent backup days and the Storage Throughput Utilization is calculated as the average across all those days. It is obtained via Eq. 8.1. Data for this metric is represented in bar graphs as a percentage of maximum. An Analysis of Variance (ANOVA) experiment was also run to determine the source of variation.

$$Storage\ Throughput\ Utilization = \frac{\frac{total\ data\ size}{total\ time}}{storage\ throughput} \tag{8.1}$$

Storage Distribution For storage distribution we examined how frequently backup-storage pairs were utilized. For a trial, we looked at each backup and gathered the frequency of all of the utilized backup-storage pairs. Then all the

numbers were used to create a histogram and this was used to visualize the storage distribution. We also examined the Mean and Standard Deviation of the values.

Another graph that is generated for storage distribution is the storage switching graph. Since the trials consist of 60 backup days, we counted how many times a backup switched from one storage device to a different one on consecutive days. This was then graphed as switches for each day.

Backup Time Consistency For backup time consistency we again used a similar histogram style as in storage distribution. Each of the 60 days had a total backup time and those times were used to create a histogram.

8.1 Storage Throughput Utilization

In order to examine the storage throughput utilization a set of ANOVA statistics were calculated for the results of the Affinity algorithm. Table 8.1 shows the results these calculations. The most interesting thing to note immediately is that the variation due to the number of backups is extremely small and statistically insignificant (with 95% confidence). However, the interaction between the number of backups and the number of storage devices is statistically significant for N/2 and N storage options, the cause of which is subtle. When examining Fig. 8.2, Fig. 8.3 and Fig. 8.4, we see that for N/2 and N storage options the effect of the number of backups seems to shift directions as the number of storage devices increases. This is because as the number of storage devices increases, this added throughput overall, with no increase to single storage throughput, causes extremely large backups to have a more negative effect on the storage throughput utilization. As the remaining smaller backups finish faster and the out-lier large backups continue processing by themselves for longer, the storage throughput utilization suffers.

Next we will examine the variation in storage throughput utilization between algorithms. Fig. 8.1 shows the performance of the Affinity, Longest Backup First,

#Options	#Storage Units	#Backups	Interaction	System Variation
1	44.2%	1.2%	3.7%	50.8%
N/2	21.8%	0.1%	10.9%	67.1%
N	30.8%	1.3%	9.3%	58.5%

Table 8.1: ANOVA was used to calculate the percentage of variation from the number of storage units, number of backups and the interaction between the two.

and Random algorithms. In it we see that overall there is very little difference in storage throughput utilization between the Affinity algorithm and Longest Backup First. Both algorithms currently suffer from being unable to handle large backup outliers which is apparent in the 6 storage unit tests (which by nature of the randomization of our study, had large outliers). This result continues to push the need for out-lier detection and handling in a dynamic system.

As identified earlier, the Affinity algorithm is not influenced much by the change in number of backups. Because of this we instead focus on the effect the number of storage options (out of N) has on the storage throughput utilization.

Fig. 8.5 displays the storage throughput utilization for varying number of storage options. In viewing the graph it becomes apparent that there is little to no benefit to increasing the number of storage options beyond N/2 (except for the case of $N = 2$ where $N/2 = 1$). This is a very interesting find because increasing the number of storage options tends to negatively impact storage distribution and therefore if there is no utilization benefit in doing so, it is more efficient to use less storage options.

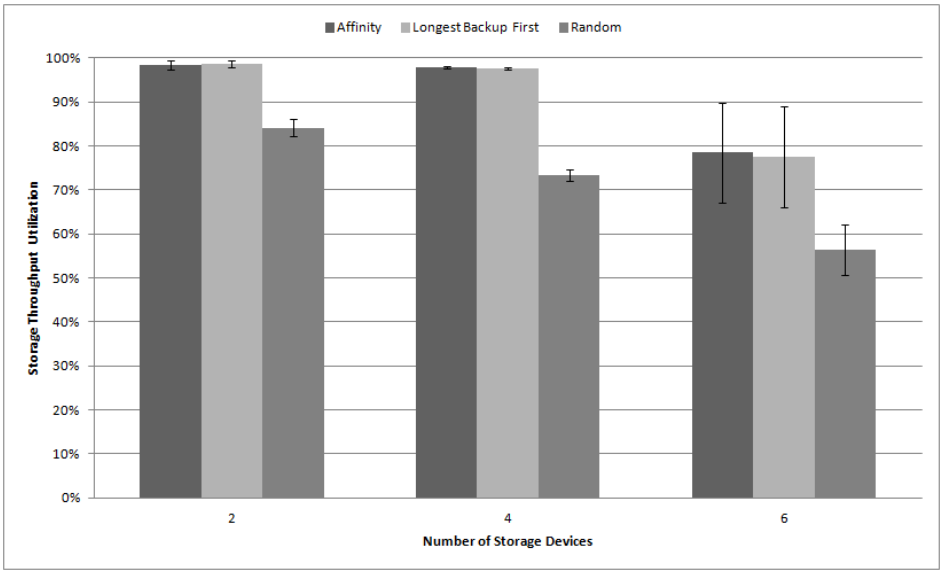


Figure 8.1: Graph showing the storage throughput utilization for different algorithms at different number of storage devices. (100 Backups)

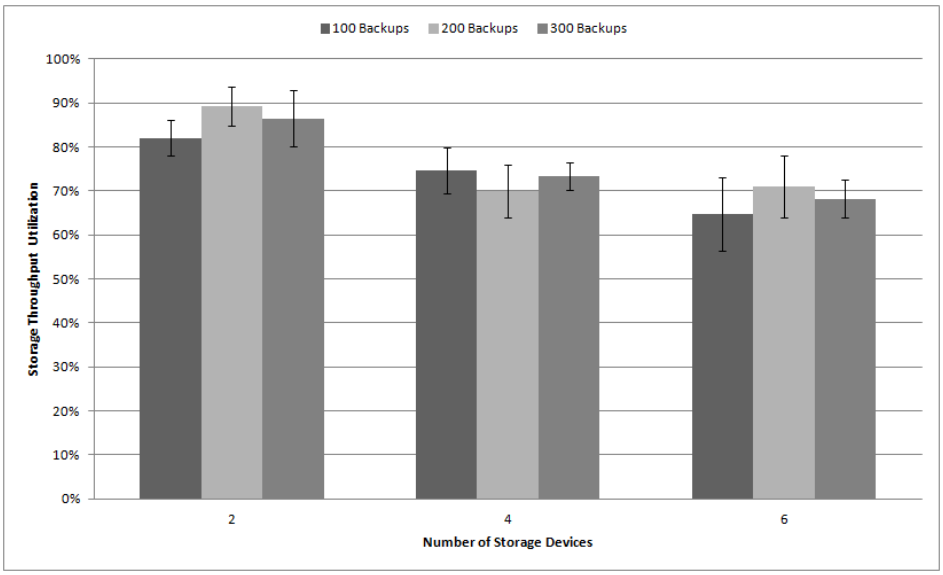


Figure 8.2: Graph showing the storage throughput utilization for the Affinity algorithm with 1 storage option per backup.

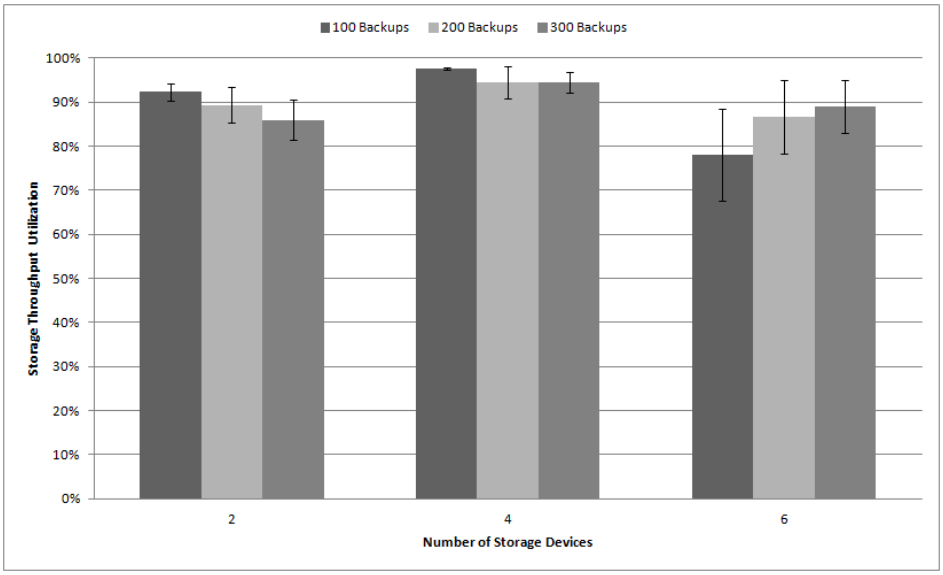


Figure 8.3: Graph showing the storage throughput utilization for the Affinity algorithm with N/2 storage options per backup. (N = number of total storage units)

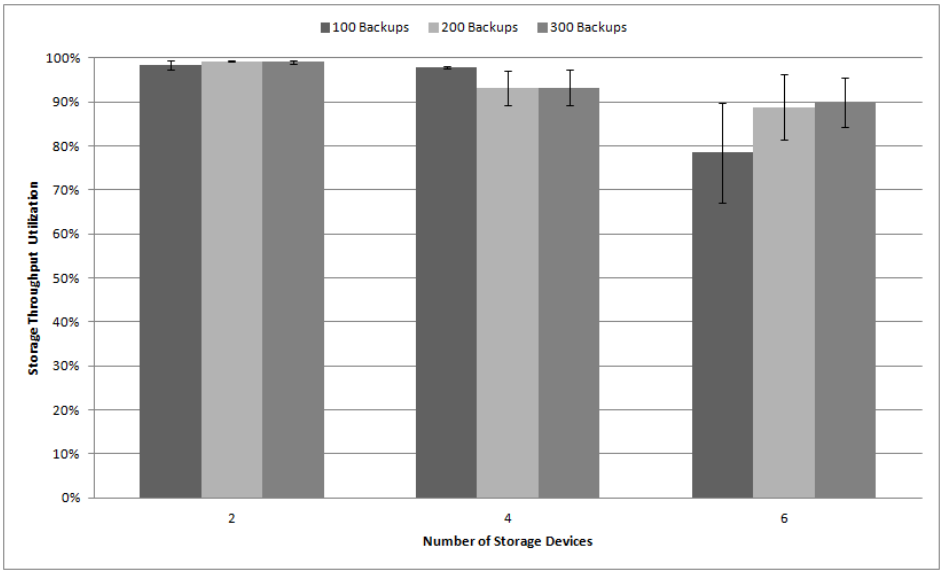


Figure 8.4: Graph showing the storage throughput utilization for the Affinity algorithm with all (or N) storage options available to all backups.

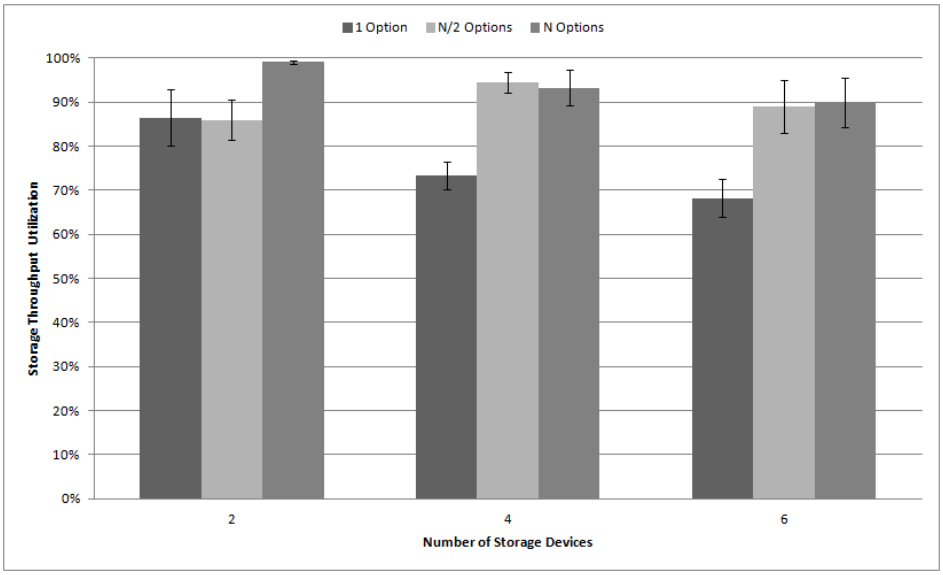


Figure 8.5: Graph showing the storage throughput utilization for the Affinity algorithm while holding number of backups constant at 100 and varying the number of storage options.

8.2 Storage Distribution

Storage distribution refers to the spread of data from the same backup policy across multiple storage devices. This is non-ideal because in an incremental environment it would be less efficient to be drawing from multiple storage devices and being pieced together. Therefore it is beneficial to examine how dynamic algorithms distribute backups to different storage options. In order to examine storage distribution a histogram graph is used. The graph, while somewhat non-intuitive allows a glimpse of the distribution of the frequency that backups choose different storage options. Fig. 8.6 shows what we think an ideal storage distribution would look like in the context of the Affinity algorithm. Based on the original idea of this algorithm, we want each backup to try all the storage options a few times but then settle on a priority option the majority of the time. This ideal distribution has two peaks, the first of which represents the randomization of the algorithm and is produced because backups try each storage device a small amount of times before determining a higher priority option. The second peak on the right represents the consistency of the algorithm that appears as time goes on. As storage determines its priority storage it chooses it more often and therefore we reduce the storage spread over time without sacrificing throughput utilization. It should be noted that this is a hypothetical ideal and realistically non-obtainable but the general shape and idea of why we are trying to obtain that shape is worth noting.

Unfortunately the results show that the algorithm is far from accomplishing the ideal goal. In a completely random scenario we would expect a Gaussian curve centered on the number of days divide by the number of storage options, however for the Affinity algorithm we would like to essentially drive a wedge in the middle and shift to large frequencies of both high and low backup-storage usage as in Fig. 8.6. Fig. 8.7 compares the distribution displayed by the Random algorithm and the Affinity algorithm. The Random algorithm behaves as expected with a Gaussian distribution centered on 10 (60 days, 6 options, $60/6 = 10$ average). The Affinity algorithm holds a similar distribution but you can see that it is starting to

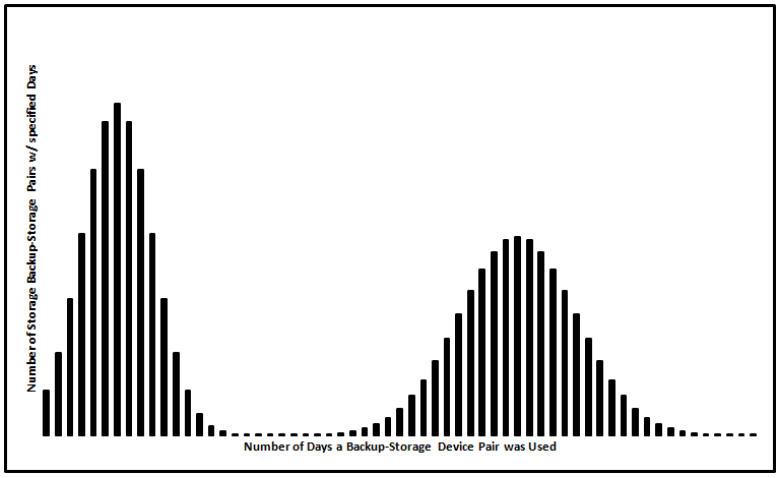


Figure 8.6: Theoretical histogram of the ideal storage distribution for what the Affinity algorithm is attempting to accomplish. It has a left peak representing the backups experimenting with storage options and a right peak which represents backups narrowing down on a single option and using it quite frequently.

be wedged down in the middle and being pushed out to the sides. This represents a step in the right direction but more tuning is needed to continue to wedge the graph and form two peaks. Fig. 8.8 compares the Affinity algorithm against the Longest Backup First algorithm. As with storage utilization we see a very slight improvement as the Affinity algorithm has shallower middle and wider base, however both still appear centered near 10. The distribution of Longest Backup First is somewhat unexpected because despite not favoring specific storage units there is still an appearance of favoritism. Table 8.2 lists the Means and Standard Deviations of the different algorithms for the given dataset. This shows a slightly higher standard deviation for Affinity which is good in this case.

Overall we feel like in order to move closer to our theoretical ideal distribution we need to change the probability function used to choose backup options (refer to Sec. 5.1.2). The function is currently too lenient and allows too much randomization and not enough consistency in selecting the highest priority. Adjusting

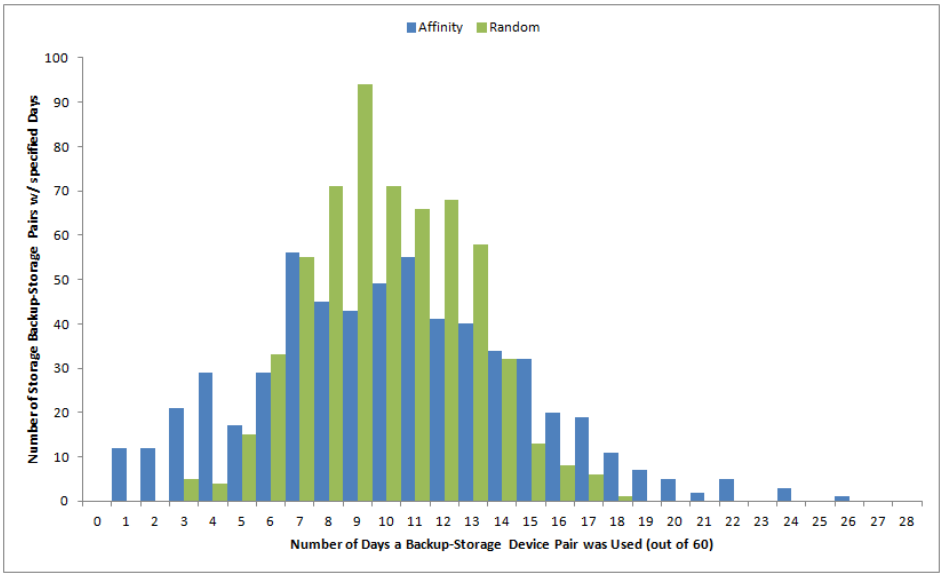


Figure 8.7: Histogram of Affinity and Random algorithms side by side for comparison.

the number of presented options or making it scale with system size is another option to potentially improve this graph. It is possible that throughput utilization will suffer a small amount in changing these things but the exact trade-offs is an interesting prospect of future work.

Algorithm	Mean	Standard Deviation
Random	10.0	2.7
Affinity	10.2	4.6
LBF	10.0	4.0

Table 8.2: Storage Distribution Statistics for different algorithms.

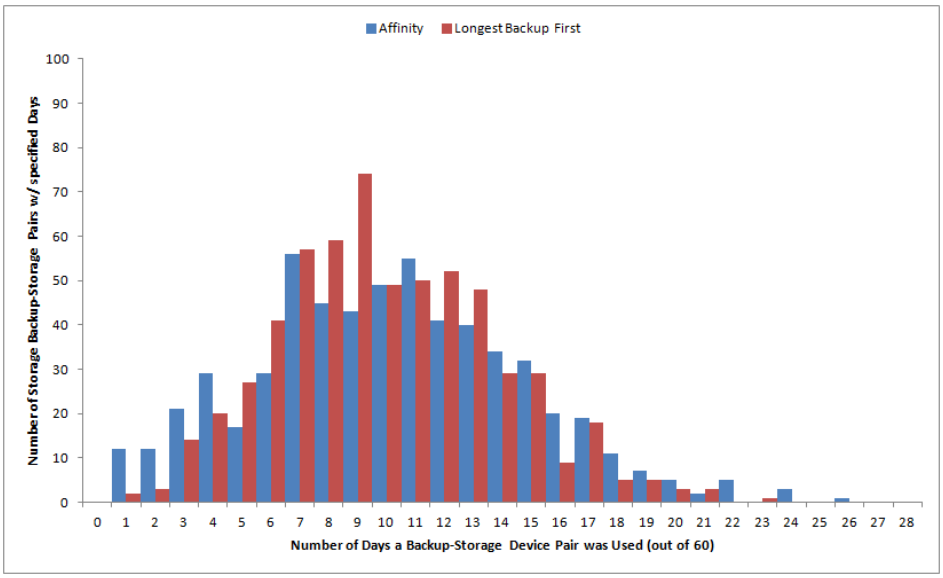


Figure 8.8: Histogram of Affinity and LBF algorithms side by side for comparison.

8.2.1 Storage Switching

Another way to examine how backups are distributing to each storage device is to examine how often they switch from one day to the previous. Ideally at the beginning it switches a lot but then over-time this decreases dramatically. Fig. 8.9 shows the percentage of backups that switch storage devices for the Affinity, Longest Backup First and Random. This graph makes it clear why the previous distributions fall short of the goal distribution. The graph is very messy but essentially all 3 algorithms behave close to equally with Affinity slightly lower in a lot of cases. This tells us that the algorithm is not forcing backups to the priority storage device strongly enough. This also brings up the possibility of a future mechanic that gives the storage used on the previous day higher priority in some form.

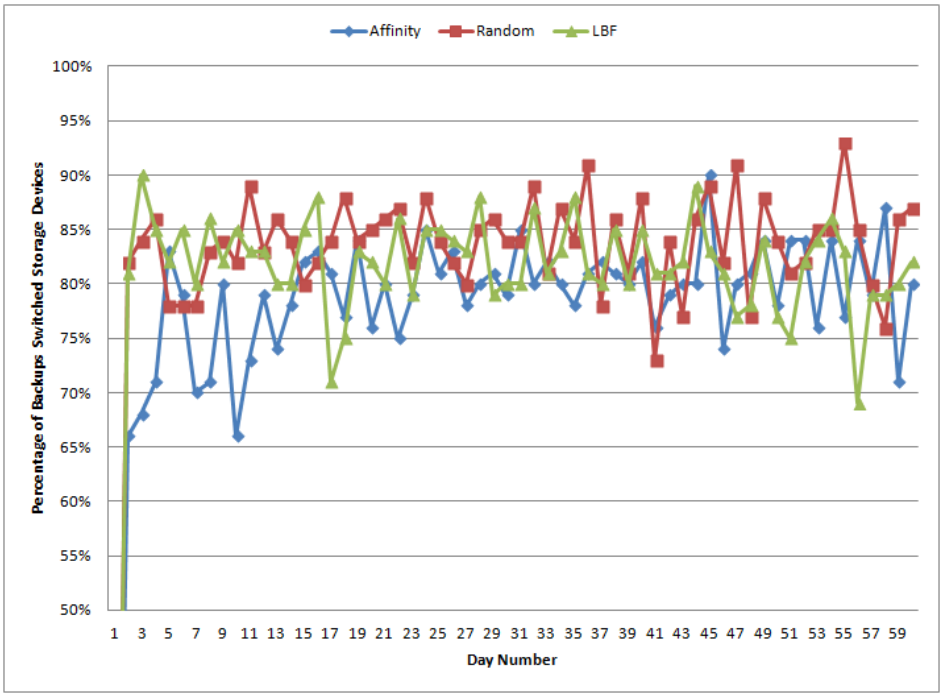


Figure 8.9: Graph showing daily storage switching percentage for a system with 100 backups and 6 storage options all available to each backup.

8.3 Backup Time Consistency

The last of the priority metrics is Backup Time Consistency. Backing up data is a necessary evil in the enterprise world and planning around the time required to backup data and allow the servers to return to operation would be greatly beneficial. In that sense the more consistent the backup time the better the result. Fig. 8.10 shows a distribution of backup times for both the Affinity and Longest Backup First algorithms. In this case the performance of the Affinity algorithm is slightly better but most importantly it has a taller and tighter distribution. Due to the slow convergence of the Affinity algorithm this trend is common and shows an improvement in consistency over Longest Backup First which is a wider Gaussian distribution.

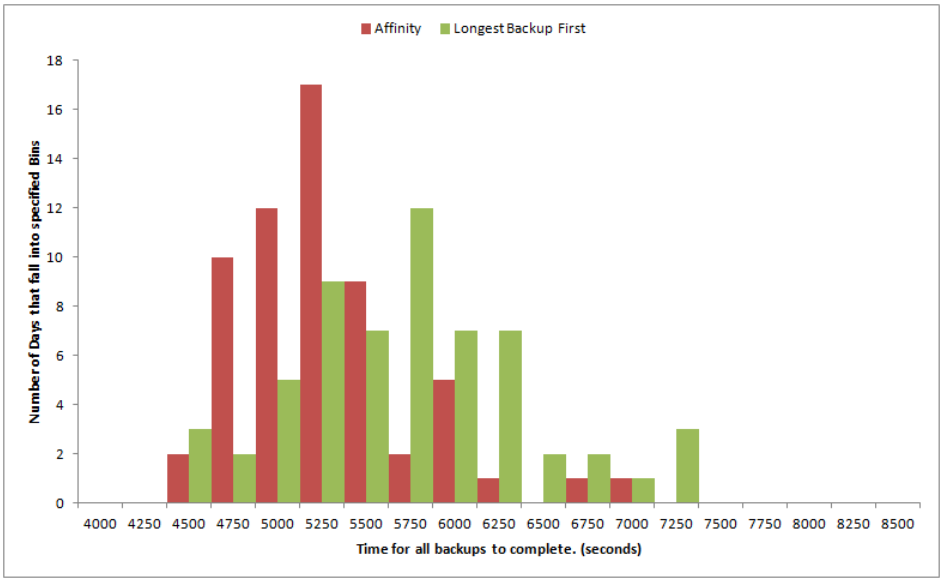


Figure 8.10: Histogram of backup times for a single system over 60 days. Affinity and LBF shown. (100 backups, 6 storage devices, N options)

Chapter 9

Future Work

For the future of the project it is first important to tackle the shortcomings of the current algorithm:

- **Storage Switching:** As it stands right now, backups are not as inclined to continue backing up to the priority storage unit as we would ideally want. There is room for exploration with the Affinity selection distribution as well as introducing a new mechanic that encourages backups to backup to the same storage as the previous day.
- **Outlier Backups:** Backups that are drastically larger than others in the same system need to be properly identified and given less concurrent backups to its storage destination. This could be handled dynamically by flagging backups that remain operating significantly longer than most of the others.

Other areas of future work include:

- **Networking:** The simulator would benefit greatly from the introduction of networking between devices. This would allow us to test algorithms ability to detect bad routes between clients and storage devices and also auto alleviate network bottlenecks.

- **Other Algorithms:** It goes without saying that there are other potentially better algorithms out there waiting to be studied.

- **Scale Up Systems:** In this set of experiments all the systems were randomized. However it would be interesting to do similar studies where smaller systems are subsets of the larger systems so there is more similarity between the systems.

- **Client initiated backup systems:** In current backup environments, all the decision making is handled in the central master server that decides when backups initiate. This would be considered server-centric. There is potential for a client-centric system where the backup clients themselves request storage and upon refusal potentially alter its request and/or request it later.

Chapter 10

Conclusion

As computer systems start pushing the petabyte and exabyte levels, data protection systems are forced to adapt their capabilities to handle the exponentially growing amount of data stored. In this paper we have examined the inefficiencies of the standard static backup scheduling system which suffers from poor performance, complicated configuration and non-existent adaptability. Dynamic backup scheduling systems is the obvious future of backup systems due to its superior throughput utilization, adaptability and ease of configuration.

We have introduced the Affinity dynamic scheduling algorithm, designed to have backups identify optimal storage device routes and slowly identify the optimal steady solution. We have built a new simulator which can simulate dynamic backup systems and create custom scheduling algorithms at will. We have used that simulator to examine the performance of the Affinity algorithm vs. Longest Backup First [2] and have shown that in all three primary metrics (Storage Throughput Utilization, Storage Distribution, Backup Time Consistency) the Affinity algorithm performs to par or incrementally improves upon Longest Backup First. In some cases we see up to 99% Storage Throughput Utilization for the Affinity algorithm. We have also however seen the current drawbacks of the Affinity algorithm and have proposed avenues for future ways to improve the

efficiency of said algorithm. This paper also examines incremental backup workloads that have the potential to vary drastically, a workload that has not been examined sufficiently for dynamic algorithms. We have seen that in incremental workloads, out-lier backups are costly and must be handled well. We have seen that the Affinity algorithm, while making incremental improvements, still has multiple weaknesses that need to be addressed. Future work aims to address those weaknesses as well as improve the overall performance and consistency of the algorithm overall.

References

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

[2] L. Cherkasova, R. Lau, H. Burose, and B. 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, Sept 2009.

[3] Netbackup appliances:. `http://www.symantec.com/backup-appliance/`.

[4] B.I. Ismail, M.N. Mohd Mydin, and M.F. Khalid. Architecture of scalable backup service for private cloud. In *Open Systems (ICOS), 2013 IEEE Conference on*, pages 174–179, Dec 2013.

[5] Nohhyun Park and D.J. Lilja. Characterizing datasets for data deduplication in backup applications. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1–10, Dec 2010.

[6] Young Jin Nam, Dongchul Park, and D.H.C. Du. Assuring demanded read performance of data deduplication storage with backup datasets. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 201–208, Aug 2012.

[7] Jingjing Yao, Ping Lu, and Zuqing Zhu. Minimizing disaster backup window for geo-distributed multi-datacenter cloud systems. In *Communications (ICC), 2014 IEEE International Conference on*, pages 3631–3635, June 2014.

[8] Ruofan Xia, F. Machida, and K. Trivedi. A markov decision process approach for optimal data backup scheduling. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 660–665, June 2014.

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

Appendix A

Simulator Variables and Options

Table A.1: BackupSystem.java Variables

Variable	Default Value	Comments
isIncremental	true	This sets whether the simulator is in incremental backup or full backup mode.
backupStart	1000	This value determines how often (ms) backups are looked to start. It defaults to 1s because that is what Symantec's Netbackup does.

Table A.2: DynamicBackupSimulator.java Variables

Variable	Default Value	Comments
stepSize	100	This is how much time (ms) passes between each timestep in the simulator.
iterations	60	This is how many days or iterations each backup system simulates. After each iteration the system resets and goes again after tabulating historic data
scheduler	—	This is what algorithm is used in this test.
systemConfigFile	—	This is the system configuration file to be parsed in.
systemConstraintFile	—	This is the constraint configuration file to be parsed in.
dataLogFile	—	This is the file to print the end of test output.
windowSizeMultiplier	-1	This is a feature not used in our testing. It gives backups a backup window proportionate to their amount of data. -1 turns this off.
overallBackupWindow	-1	Similar to the previous variable, this is used to put an overall backup window on the entire backup set. Not used in our experiments so turned off with -1.

Table A.3: SystemRandomizer.java Variables

Variable	Default Value	Comments
outputFileName	–	This is the name of the file written to
numServers	3	This is the number of media servers generated.
numStorageUnits	–	Varied in experiments.
stuMeanThroughput	100	Mean throughput of backup storage units used for Gaussian distribution.
stuThroughputDeviation	20	Standard Deviation also used for storage unit throughput generation.
stuMaxData	10,000	Max data allowed in storage units. This value currenlty does not mean anything.
numClients	–	Number of backup Clients. Varied in experiments.
clientsMeanThroughput	75	Mean throughput of client storage used for Gaussian distribution.
clientsThroughputDeviation	10	Standard Deviation also used for client throughput generation.
backupsPerClient	1	How many backup policies per client.
backupsScale	100,0000	Scale for the Pareto distribution used for determining backup sizes.
backupsShape	2	Shape of Pareto distribution used for determining backup sizes.
throughputVariance	.05	The variation in throughput of storage in the experiments (on a step by step basis).

Table A.4: ConstraintsRandomizer.java Variables

Variable	Default Value	Comments
inputFileName	–	The associated systemFile to be used for constraints.
randomizeStorage	True	Whether to randomize storage options.
numRandomStorageConstraints	–	Number of storage options, varied in experiment.
randomizeServer	False	Whether to randomize server options.
randomizeStartTime	False	Whether to randomize start time.
randomizeEndTime	False	Whether to randomize end time.

Appendix B

Sample Input Files

B.1 System Input File

```
#SampleInputFile.txt
#Media Servers
#media_server(*name*)
media_server(server1)
media_server(server2)

#Storage Devices
#stu(*name*,*media_server_name*, *throughput(MB/s)*,
# *throughput_variance(%)*, *max_data(MB)*, *current_data(MB)*)
stu(storage1, server1, 100, .05, 1048576, 0)
stu(storage2, server2, 100, .05, 1048576, 0)

#Clients
#client(*name*, *throughput(MB/s)*, *throughput_variance(%)*)
client(client1, 100, .05)
client(client2, 100, .05)
client(client3, 100, .05)
```

```
client(client4, 100, .05)
client(client5, 100, .05)
client(client6, 100, .05)

#Backups
#backup(*name*, *client*, *data_size(MB)*)
backup(backup1, client1, 10000)
backup(backup2, client2, 10000)
backup(backup3, client3, 10000)
backup(backup4, client4, 10000)
backup(backup5, client5, 10000)
backup(backup6, client6, 10000)
```

B.2 Constraint File

```
#SampleConstraintsFile.txt
#Tied to SampleInputFile.txt
#*backupName*(*storageConstraints*, *serverConstraints*,
# *startTimeConstraint*, *endTimeConstraint*)
# a * indicates no constraints
backup1({storage1, storage2}, *, *, *)
backup2(storage2, *, *, *)
backup3({storage1, storage2}, *, *, *)
#Note, not all backups need constraints. If they are not
# on the list, they are given empty constraints.
backup5(storage1, *, *, *)
backup6({storage1, storage2}, *, *, *)
```