Improved Performability of Disk Arrays by the Use of Fuzzy Control

A Dissertation

Presented in Partial Fulfillment of the Requirements for the

Degree of Doctorate of Philosophy

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Guillermo Navarro

Major Professor: Gregory Donohoe, Ph.D.

Committee Members: Milos Manic, Ph.D.; Axel Krings, Ph.D.; David Umberger, Ph.D.

Department Administrator: Frederick Sheldon, Ph.D.

November 2015

AUTHORIZATION TO SUBMIT DISSERTATION

This dissertation of Guillermo Navarro, submitted for the degree of Doctor of Philosophy (Ph.D.) with a Major in Computer Science and titled "Improved Performability of Disk Arrays by the Use of Fuzzy Control," has been reviewed in final form. Permission, as indicated by the signatures and dates below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor: _____ Date: _____

Gregory Donohoe, Ph.D.

Committee Members: _____ Date: _____

Axel Krings, Ph.D.

_____ Date: _____

Milos Manic, Ph.D.

_____ Date: _____

David Umberger, Ph.D.

Department Administrator: _____ Date: _____

Frederick Sheldon, Ph.D.

ABSTRACT

Performability is the composite measure of performance and reliability. This measure is a vital evaluation method for fault-tolerant systems that can undergo a graceful degradation of performance in the presence of faults, allowing continued "normal" operation. Performability analysis is the study of the performance of systems under non-optimal conditions. The non-optimal conditions can be degraded, such as drive failure or with background tasks, such as background logical volume copy. The performability study of disk arrays is the study of a competitive challenge imposed to disk arrays. Now disk arrays are expected to guarantee low user latencies even under self-repairable failure conditions such as a disk failure and/or in the presence of background tasks such as data replication. Besides that expectation, the disk arrays are also expected to repair themselves and finish background tasks as quickly as possible. The two goals are opposing in nature. If the disk array allocates more of its resources to serve user requests, the self-repair and the background tasks take longer to be completed. But if the disk array allocates more of its resources to self-repair or the background tasks, the user requests will suffer a performance impact in terms of higher latencies or lower throughputs. This is a challenge that disk arrays have to meet in order to meet user expectations better. There is no perfect response to this challenge. The solution is to propose responses that optimize the use of the internal resources of a disk array. The problem of achieving the opposing goals is posed as a control problem and that is tackled by applying fuzzy logic and control. This dissertation makes two major contributions:

1) Performability analysis of disk arrays using fuzzy logic that provides us with a practical, easy-to-use, numerical algorithm to achieve consistently high performability based on the reliability metrics of a RAID disk group.

2) Fuzzy control approach to improve disk array performability that gives us a practical, effective, and easily-updated means to schedule the execution of customer requests and concurrent data protection tasks. This approach overcomes the lack of internal information of components by using a rule-based approach instead of a detailed control model.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

CHAPTER 1: INTRODUCTION

1.1 PERFORMABILITY OF DISK ARRAYS

Fault-tolerant systems are expected to serve their purpose even in the presence of failures. Reliability analysis is the study of the estimation of how likely is a failure to occur in a fault-tolerant system. Performability was proposed over three decades ago as an answer to the question "what is the level of effectiveness of a system considering the likeliness of failures?' In other words, "what is the performance of a system based on its reliability". Performability answers this question by the unification of performance and reliability analysis applied to fault-tolerant systems.

Performability was proposed and defined by Meyer [Meyer 78a] as "the unification of performance and reliability." Meyer presented a performability evaluation of fault-tolerant computers for aircraft control [Meyer 80a] as one the first examples of performability evaluations published. Based on Meyer's definition of performability, other authors published performability analysis of different fault-tolerant systems such as disk arrays [Islam 93a][Barnett 98a].

The concept of performability was extended by Zhang et al. [Zhang 06a] to consider background tasks in addition to failure conditions. Zhang in [Zhang 06a] presented a performability evaluation of a disk array under the presence of *background jobs*, i.e., tasks the disk array executes with no user intervention and have a soft deadline. The background jobs are independent of the user jobs, i.e., user reads and writes.

Redundant arrays of inexpensive disk (RAID) [Patterson 88a] systems were proposed with the goal of avoiding the loss of data stored on disks and increasing the throughput of a group of disks. The research in RAID systems was jumpstarted by the seminal paper by Patterson in 1988.

Disk arrays are fundamentally RAID systems but with an advanced set of features added over the years since the late 1980s. One example of such features is local and remote logical volume copy between disk arrays. Another example is intelligent data caching schemes between different storage media such as Solid State Drives (SSD) and magnetic Hard Disk Drives (HDD). Disk arrays are fault-tolerant systems that are expected to deliver the storage and retrieval of data even under the presence of failures or background jobs.

The presence of failures or background jobs can be considered a non-optimal state of a disk array from the point of view of a user issuing read/write requests. Both failures and background jobs can be then considered *non-optimal conditions*. The *optimal state* of a disk array can be defined as the state in which the disk array has no failure and background jobs and can dedicate the resources (CPU, memory, IO ports, Hard Disk Drives) exclusively to the service of user reads and writes. Based on Meyer's original definition of performability and Zhang's extension of Meyer's definition, we will define performability of disk arrays as *a measure of the probability of user requests to achieve a performance level under non-optimal conditions*. A more precise definition of performability is presented in section 2.2.

1.2 Objective of this Dissertation

As part of the progress made in the development of new disk array features, the challenges imposed to disk arrays are greater. Now disk arrays are expected to guarantee low latencies (response times) and high throughputs (in I/O requests per second) for user requests (read/writes) even under self-repairable failure conditions such as a disk failure and/or in the presence of background tasks such as data replication. Besides that expectation, the disk arrays are also expected to repair themselves and finish background tasks as quickly as possible. The two goals are opposing in nature. If the disk array allocates more of its resources to serve user requests, the self-repair and the background tasks take longer to be completed. But if the disk array allocates more of its resources to self-repair or the background tasks, the user requests will suffer a performance impact in terms of higher latencies or lower throughputs. Clearly, there is a challenge for disk arrays to provide the best performability.

The research reported in this dissertation seeks to improve the performability of disk arrays by:

1) Proposing an algorithm for estimating the performability of disk arrays considering failure rates. The algorithm makes use of fuzzy logic to deal with the uncertainty of some of the parameters.

2) Proposing control strategies based on the use of fuzzy logic and control. The fuzzy control will be used to control the execution of background tasks based on external requirements such as desired latency of user requests and time to complete background tasks.

The purpose of the fuzzy logic as well as fuzzy control solutions is to minimize the performance impact of the background jobs on the user request latency and throughput. The fuzzy logic and fuzzy control approach was chosen because it overcomes the limitation of the lack of internal information of components such as disk drives.

There is no perfect response to the disk array performability challenge. The solution is to propose responses that optimize the use of the internal resources of a disk array. The optimization is understood in this dissertation as the balancing of the use of resources such as Hard Disk Drives (HDDs), to achieve the goal of accomplishing mutually exclusive goals. The balancing of the resources to provide an optimal response to competing processes requires considering a number of parameters such as disk latencies, disk array controller latencies, bandwidth of the communication ports, data transfer sizes, memory caching algorithms, RAID levels, data access patterns such as random or sequential, type of data accesses (read or writes), and queue lengths. This list of parameters is not a comprehensive list of all the parameters to consider for the optimization problem, but it gives an idea that the problem can have a dimensionality that makes it complex. This is the challenge that this dissertation approaches by proposing fuzzy control schemes for disk array performability.

1.3 Assumptions of the Dissertation

Certain assumptions are made throughout this dissertation. The first is that no cache memory is considered when proposing the performability solutions for the disk array. The disk array will be considered to be in write-through mode, i.e., the user writes will go directly to the disks. The second assumption is that the bottlenecks can be the hard disks or the disk array controller. Other components, such as front-end and back-end communication links (Fibre Channel, Serial Attached SCSI (SAS), and Ethernet) are not considered a bottleneck for the purposes of this study.

There is no intention in this dissertation to do an exhaustive modeling of disk array components. In other words, it is not the intention to present comprehensive analysis of disk drive behavior, the disk array controller board or the communication links. There is no intention to study the different kinds and patterns of user data workloads applied to disk arrays. The type of user data workload used for this study will be the On-line Transaction Processing (OLTP) workloads, the kind of workload produced by databases such Oracle ™.

The OLTP workload is dominated by small transfer sizes (8KiB or 16KiB) and random accesses over virtual disks (VDs), which are also referred to as logical volumes (LVs).

1.4 Contributions of this Dissertation

This dissertation presents two major contributions:

1) First, this dissertation contributes in the performability analysis of disk arrays using fuzzy logic that provides us with a practical, easy-to-use, numerical algorithm to achieve consistently high performability based on the reliability metrics of a RAID disk group.

2) Second, this dissertation contributes by proposing a fuzzy control approach to improve disk array performability that gives us a practical, effective, and easily-updated means to schedule the execution of customer requests and concurrent data protection tasks. This approach overcomes the lack of internal information of components such as disk drives by using a rule-based approach instead of a detailed control model. The fuzzy control schemes presented in this dissertation have resulted in patents awarded by the United States Patent and Trademark Office (USPTO) [US Patents 8,201,018, 8,650,145, and 9,063,835].

1.5 Organization of this Dissertation

Chapter 2 is a background on disk arrays, performability and fuzzy control. The description of a generic disk array and the current technologies used for disk arrays are provided as well as an introduction to the data protection policies used by disk arrays. Also, the fundamental concept of virtualization as understood for disk arrays is explained. The performability concept is explained in detail and an example is provided. Finally, fundamentals of fuzzy control theory are provided for the reader.

Chapter 3 tackles the disk array performability analysis problem by the use of fuzzy performability applied to an e-mail server. Chapter 3 presents contribution 1) mentioned in the previous section, i.e., the estimation of performability of disk arrays by using a fuzzy numerical method. The application of this algorithm in the sizing of an e-mail server shows how this numerical method to estimate performability can be applied to size IT services such as Email.

Chapter 4 approaches the problem applied to the sparing (rebuild) and a solution using fuzzy logic control and neural-fuzzy control. This chapter makes the contribution of new patented fuzzy control schemes that provide better performability of disk arrays when

reconstructing data redundancy (sparing) due to a disk failure. This performability is improved by reducing the sparing time by half while at the same time ensuring a proper latency of user requests (reads and writes) under the presence of the background sparing process.

Chapter 5 tackles the problem of point-in-time copy of logical volume (LV) snapshots solved using fuzzy logic. This problem is addressed by proposing a novel patented scheme to deal with the Copy-On-Write problem along with a novel fuzzy control scheme that ensures that the latency of user requests will not be as impacted by the snapshot copy of LVs and the same time it guarantees the progress of the LV snapshots.

Chapter 6 proposes a solution to the problem of point-in-time copy of LV cloning replication using fuzzy logic. This problem is managed by proposing a new patented scheme that throttles the rate of cloning replication when the user latency is high but speeds up the rate of replication when the user latency is low. This balance between goals is achieved by using a fuzzy controller scheme that balances the need of a low latency of user requests vs. a quick LV cloning replication.

Chapters 4, 5 and 6 present the fuzzy control approaches related to contribution 2) mentioned in the previous section. These three chapters show how the fuzzy control approach can be practically applied overcoming the lack of internal information of components such as disk drives. The purpose of the fuzzy control schemes in these three chapters is to improve the latency of the user requests (reads and writes) in the presence of a background job.

CHAPTER 2: BACKGROUND ON DISK ARRAYS, PERFORMABILITY AND FUZZY CONTROL

This chapter provides an introduction to the three areas of knowledge that compose this dissertation: disk array technology, performability, and fuzzy control.

2.1 DISK ARRAYS

Disk array is the term used for the Redundant Array of Independent Disks (RAID) with additional features that have been added to the original RAID concept. The concept of RAID is first patented by N. K. Ouchi in 1978 (US Patent 4,092,732). Disk arrays are now an essential part of the IT centers. The Storage Networking Industry Association (SNIA) http://www.snia.org is an organization of member companies with the mission to promote standards, technologies and educational services related to storage technologies. The SNIA defines a disk array as *a set of disks from one or more commonly accessible disk subsystems, combined with a body of control software. The control software presents the disks' storage capacity to hosts as one or more virtual disks.* The term virtual disk is defined as *the disk array object that most closely resembles a physical disk from the operating environment's point of view* [SNIA 13a]. The term *logical volume* is also used as a synonym of virtual disk.

(a) HP 3PAR 7400 (b) Ace Powerworks 466

Fig. 2.1: Examples of modern disk arrays

Disk arrays are fault tolerant since they can continue to operate under the failure of a drive or a controller. Fig. 2.1 shows two pictures of modern disk arrays. Disk arrays provide not only the means to store huge amounts of data, but means to ensure the survival of the data in case of failures or catastrophes. The disk array business market share was US $22.3 billion in 2012 [Gartner 13a]. For example, Facebook, the social networking service, use disk arrays to store user profiles. The user profiles contain text, picture, audio and video. As of the end of

2013, Facebook had over 240 billion photos in disk arrays and 350 million photos were added per day. This translated into 7 Petabytes of new storage per day.

The concept of redundant arrays of inexpensive disks (RAID) was first published in a peer-reviewed journal by Patterson [Patterson 88a] to improve the dependability and performance of storage systems [Patterson 94a]. The RAID systems have fault tolerance to disk failures by storing redundant copies of the user data (RAID1) or by using parity as a means to rebuild the data in case of disk failure. When a disk fails, the disk array loses the data redundancy of the data on the failed disk. The process of reconstructing the data redundancy is known as rebuild [Menon 94a] or sparing [Thomasian 97a]. Issues related to reliability have been researched before [Schulze 89a], [Burkhard 93a], [Ganger 94a]. The performance under optimal conditions has been studied before [Lee 93a], [Catania 95a], [Schwarz 92a], [Varki 03a]. The performance under degraded conditions and performability estimation has been researched before as well [Islam 93a], [Muntz 90a], [Reddy 91a], [Thomasian 97a], [Barnett 98a].

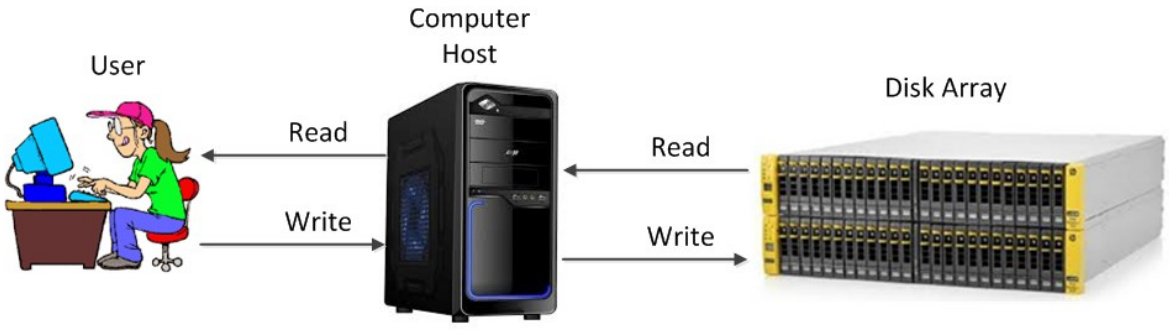

Fig. 2.2: Storage as a service: typical scenario

Disk arrays provide storage service by two basic data transfer operations: user reads and user writes. Users send read or writes requests through the computer host, as shown in Fig. 2.2. The computer host relays the requests for data (read) and to save data (write). The two most important performance metrics of the Reads/Writes are throughput and latency. The reads and writes have other attributes such as data transfer size. The number of data transfer operations (reads or writes) per unit of time is the throughput, usually measured in requests/second. The definition of throughput by the SNIA is *the number of I/O requests satisfied per unit time. The throughput is expressed in I/O requests/second (IO/s), where a request is an application request to a storage subsystem to perform a read or write operation*.

The time it takes for a request to be satisfied is the latency, usually measured in milliseconds. The definition of latency proposed by the SNIA is *synonym for I/O request execution time, the time between the making of an I/O request and completion of the request's execution.* The term *response time* is used in the Storage community as a synonym for *latency*. For this dissertation, both terms will be used. Another attribute of a user request is the transfer size, which is measured in Kibibytes (KiB), 1024 bytes.

2.1.1 Disk Array Architecture

A modern disk array is basically composed of two main sections: the controllers and the array of disks. The disks used by disk arrays are most commonly hard disk drives (HDDs) or solid state drives (SSDs). Disk arrays achieve fault tolerant capability by the use of redundancy. The number of controllers of a fault tolerant disk array is at least two. The minimum number of drives varies from product to product but is usually at least eight HDDs. Fig. 2.3 shows the basic block diagram of a disk array.

The disks store the user data using the controller as the link between the storage provided by the drives and the users of the storage space. The disks are installed in specially designed enclosures (disk enclosures) that hold a number of disks, e.g. 20, that have connectors and electronic circuitry to allow all disks in the disk enclosure to be "visible" (accessible) and communicate with the disk array controllers. In modern disk arrays, the disk enclosures connect to the disk array controllers usually through Fibre Channel (FC) or Serial-Attached SCSI (SAS) interfaces.

The controllers provide three essential functions: 1) provide virtual storage capacity to computer systems; 2) interface with computer systems and 3) provide data redundancy so data can be recovered in the event of a disk failure. The first essential function of presenting the virtual storage capacity means that the capacity of all disks combined is presented as one single big capacity. In other words, the disk array controllers abstract out all the *physical* details of the disk configuration such as number and storage capacity of the disks, and present a *logical* combined storage capacity of all disks. For example, if a user has a disk array with 20 disks with 300GB of capacity each, the disk array controller may present a single 20 x 300GB = 6,000GB capacity to the users. This allows users of the disk array to allocate capacity easily by leaving all decisions about the physical details (which disks and sectors

within the drives to use) up to the controllers. The virtual storage capacity depends on the RAID level. This will be explained in subsection 2.1.2.

The disk array controllers also perform the essential function of interfacing with the computer systems, e.g., Windows or Linux, that make use of the virtual storage capacity provided by the disk array. The most common communication interfaces used by the controllers are FC and Internet SCSI (iSCSI). The communication ports used by the

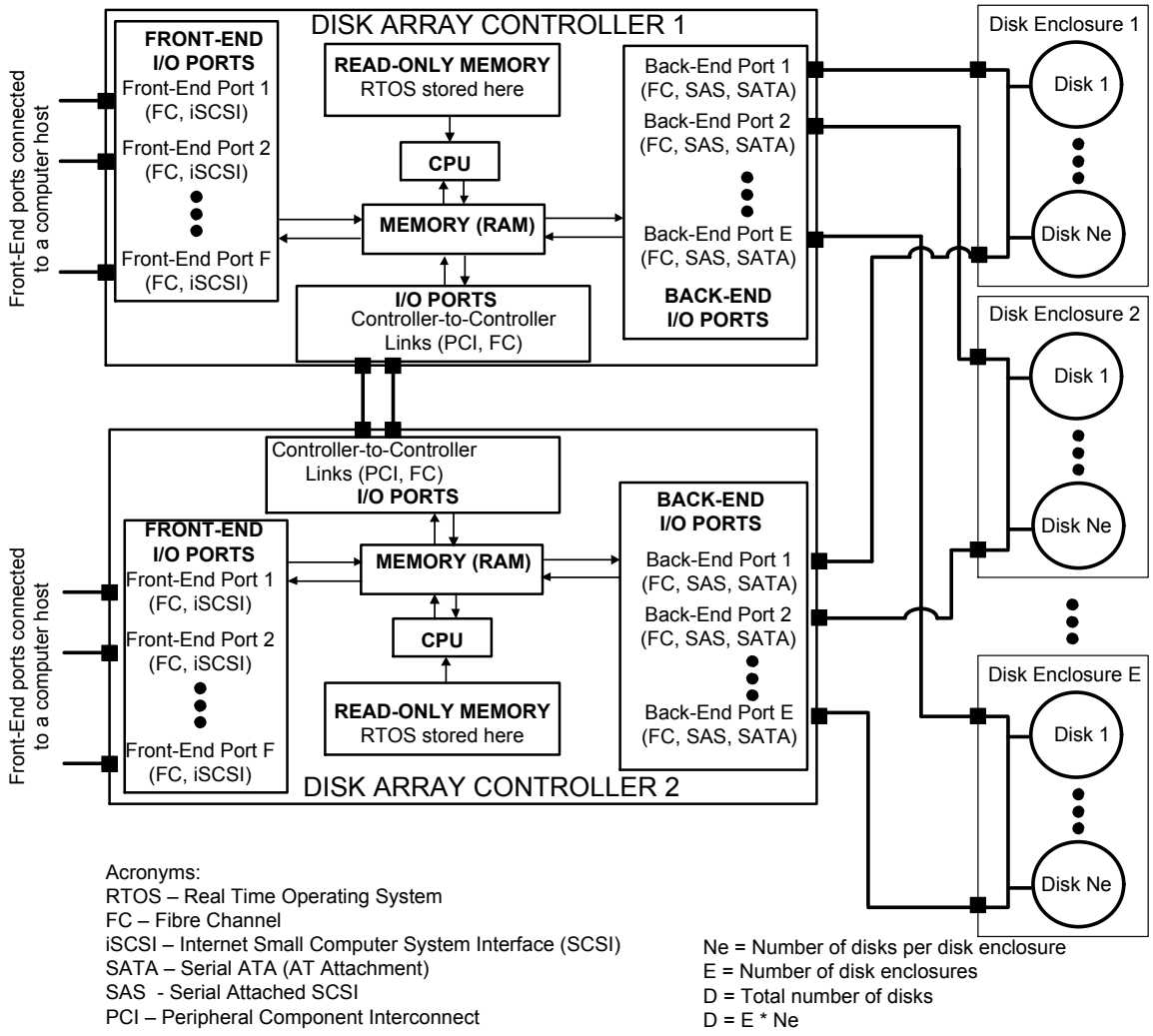

Fig. 2.3: Block Diagram of a Modern Disk Array

controllers to interface with the computer systems are referred to as the *front-end I/O ports*. The ports used to communicate with the disks (through the disk enclosures) are referred to as *back-end I/O ports*. The controllers communicate with each other most commonly using FC or Peripheral Component Interconnect (PCI) interfaces.

The disk array controllers contain central processing unit (CPU) along with random access memory (RAM) and read-only memory (ROM) to implement and execute the algorithms that carry out the essential functions of the controllers and more features, such as local replication.

2.1.2 RAID LEVELS

RAID systems make use of two orthogonal concepts: data striping across disks for improved performance, and redundancy for improved reliability. Data striping allocates data over multiple disks to make them appear as one single, large, fast disk. This allows multiple I/Os to be serviced in parallel. Most of the redundant disk array organizations can be distinguished based on two features: 1) the granularity of the data interleaving and 2) the method and pattern in which the redundant information is computed and distributed across the disk array [Patterson 94a].

The basic RAID levels that were introduced by Patterson, Gibson and Katz in [Patterson 88a] are RAID1 through RAID5. The term *level* is used to denote the method and pattern used to maintain the redundancy of the data. There are very complete descriptions of the RAID levels in [Shooman 03a] and [Patterson 07a]. In this section a basic presentation of the RAID levels is given.

1. RAID0 – This level has no redundancy. The data is striped across the disks. This level is not as used in practice.
2. RAID1 – This level implements redundancy by copying or *mirroring* data across drives. The most common number of copies is two. This means that data is written to two disks. When data is read, then either disk can be picked to provide the data. This RAID level is used a lot in practice because it is simple and does not require any special parity computation. The drawback of this RAID level is cost in terms of space efficiency; if two copies of the data are stored, that reduces in half the available storage capacity for the users to store data.
3. RAID2 – This level implements memory-style error correcting code. This RAID level is practically not used in commercial disk arrays. It is mentioned here for completeness.
4. RAID3 – This is a bit-interleaved parity level. In other words, the parity is computed at the bit level. Levels 3, 4 and 5 make use of the XOR function to compute parity [Shooman

03a]. The data is striped across a group of *N* disks including a parity disk. When reading or writing, all *N* disks have to be read or written. This level has been used rarely in practice

5. RAID4 – This is a block-interleaved parity level. The parity-bit code is applied at a block level, e.g., 512 or 2048 byte-blocks. The parity bits are stored on a dedicated parity disk. The fundamental difference between level 3 and 4 is that the data is interleaved between disks at the sector level in 4 and at the bit level in 3.
6. RAID5 – This is a block-interleaved parity level, like level 4, but the parity blocks are distributed across the disks. This level is widely used in practice by the disk array companies such as EMC, IBM and Hewlett Packard.

Table 2.1: RAID Levels and Number of disk failures tolerated

RAID Level		Disk failures tolerated and parity overhead for data striped across *G* disks
0	Non-redundant striped	0 failures and 0 parity disks (no overhead)
1	Mirrored	1 failure and *G*/2 disks
2	Memory-style ECC	1 failure and *G*/2 disks
3	Bit-interleaved parity	1 failure and 1 parity disk
4	Block-interleaved parity	1 failure and 1 parity disk
5	Block-interleaved parity	1 failure and 1 parity disk
6	Block-interleaved parity computed in two ways	2 failures and 2 parity disks

7. RAID6 – This is a block interleaved parity level, like level 5, but the two parity blocks are computed instead of one as in level 5. The computation of two different independent parity blocks allows the disk array to recover from two disk failures. This level is becoming very widely used in practice as the size of disks increases.

Since the introduction of RAID systems in Patterson's seminal paper in the late eighties [Patterson 88a], disk arrays have been an active area of research. The analysis of the

reliability and the performance of the RAID systems and the different RAID levels have been studied since then [Schulze 89a],[Patterson 94a],[Burkhard 93a],[Ganger 94a],[Patterson 07a],[Rezaul 93a],Barnett 98a].

2.1.3 STORAGE VIRTUALIZATION

This dissertation makes use of the *virtual disk* or *logical volume* concept, explained in this section. The SNIA defines a virtual disk as *a set of disk blocks presented to an operating environment as a range of consecutively numbered logical blocks with disk-like storage and I/O semantics. The virtual disk is the disk array object that most closely resembles a physical disk from the operating environment's viewpoint.* For this dissertation, the term *logical volume* will be used as a synonym of virtual disk.

Disk arrays store user data using a technique named *storage virtualization*. In order to explain what storage virtualization is, we need to explain what a logical volume is in the context of disk array technology.

The disk arrays combine the storage capacity of all the disks connected to the array in one single capacity that is referred to as *total capacity*. The total capacity is usually in the order of Terabytes (TB) or Petabytes (PB). Disk arrays are designed to share the total capacity among different users and to allow the allocation of capacity in stages. The way these two goals are accomplished is by the use of *logical volumes*. Logical volumes are partitions of the total storage capacity offered by the disk array. For example, if the total storage capacity offered by a disk array is 10 TB, a user may allocate only 1 TB for a logical volume and leave all the other 9 TB available for some other time. This allows customers to save time because the disk array only has to be create tables in memory for the actually allocated space, e.g. 1TB, but at the same time the disk array is prepared to grow those tables easily when more space demanded, e.g. 2TB more. A logical volume is presented to a user as a set of consecutive and individually addressable bytes. The number of bytes in a logical volume depends on the size that the user allocated. Following the example of the 1 TB logical volume, the disk array would allocate 2^{40} bytes for the user presented as one set of consecutive bytes encapsulated in the 1 TB logical volume. This is known as the *virtual disk* or *logical volume presentation*. In set form:

$$LV_{ID} = \{b_1, b_2, ..., b_S\} \quad (2.1)$$

where b is a byte and S is the total number of bytes allocated to the logical volume. The *ID* is a unique identifier assigned to a logical volume. The *ID* can numerical or alpha-numeric. Fig. 2.4 shows an example of the physical implementation of a logical volume using RAID1 (R1).

The disk array presents the logical volume in a logical form, but the physical allocation is different and depends on factors such as the RAID level to use, the number of disks on which the logical volume will be stored, and the granularity of the physical allocation.

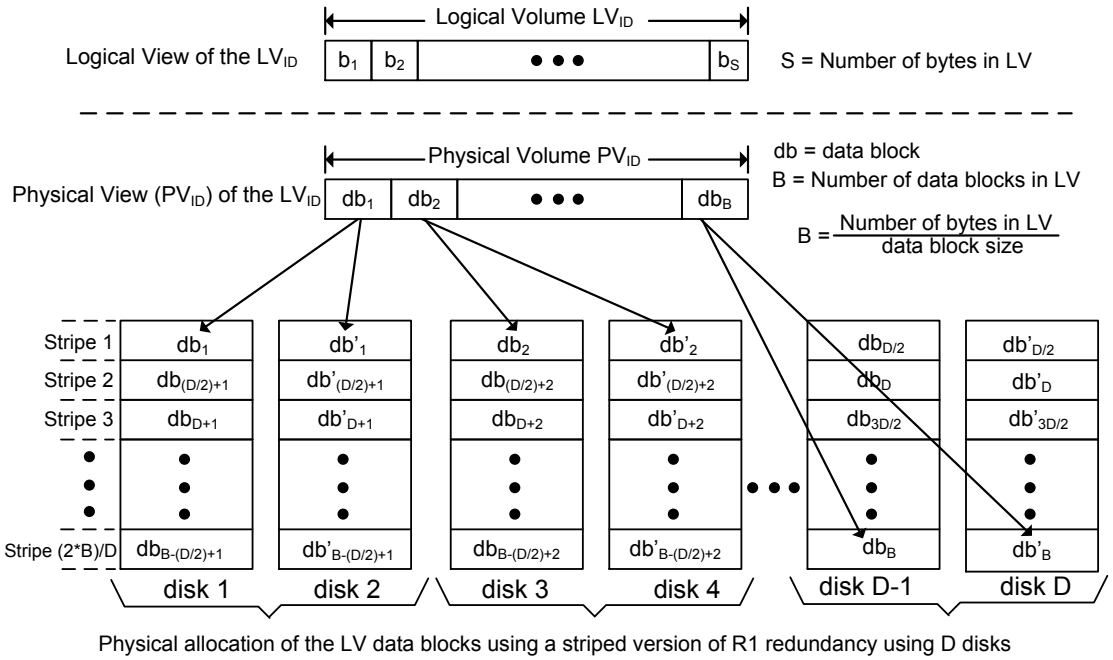

Fig. 2.4: Storage Virtualization: LV logical and physical implementation

The disk array keeps track of the allocated space and whether or not it has been written to or moved from one disk to another. The tracking of this information is kept in tables known as *metadata*. It is impractical to have the metadata keep track of the activity at the byte level. As a consequence, the disk arrays keep data for logical volumes using a minimum unit of allocation much bigger than a byte. This minimum unit of allocation will be referred to as *data block* and its size varies in practice for different disk arrays. Typical data block sizes in practice are 128KiB, 256KiB, 512KiB, 1MiB and sometimes bigger. The

number of data blocks B, to allocate to a logical volume depends on the size of the logical volume, S, and the data block size, S_b.

$$B = S / S_b \tag{2.2}$$

The logical volume is allocated on the disks as a *physical volume*. The physical volume contains the data blocks and the mirrored copies in the case of RAID1 or the parity blocks in the case of RAID3, RAID4 and RAID5. The transformation from logical to physical volume depends on logical volume *ID*, the size of the data block S_b, the RAID level R_L, and the number of disks in the RAID group or *group size*, G, e.g., two disks for RAID1.The PV_{ID} is a function that maps a logical volume into a list of data blocks.

$$PV_{ID} = (LV_{ID}, S_b, R_L, G) \tag{2.3}$$

The number of data blocks in the physical volume depends on the RAID level and the group size G. The group size G determines the sequence of data and mirror or parity blocks according to the number of disks G in the disk group. The example in Fig. 2.4 shows a physical implementation of a logical volume using R1, and a group size of two. Since $G = 2$, each data block has a mirrored copy on another drive. The physical volume transformation of the logical volume is shown in Fig. 2.4 is

$$PV_{ID} = (LV_{ID}, R_1, 2, S_b) = \{db_1, db_1^{'}, db_2, db_2^{'}, ..., db_B, db_B^{'}\} \tag{2.4}$$

where each data block db_i has its corresponding copy $db^{'}_{i.}$

The disk array has a total number of disks D. The physical volume is allocated across all the D disks in groups of G disks.

2.1.4 Data Protection Policies

Data protection policies are the procedures a disk array executes to copy (replicate) the data on a disk array to protect against data loss. The typical data protection policies are sparing, snapshot and cloning (mirroring). Sparing is different from snapshot and cloning

because sparing has the goal of recovering the RAID redundancy in a disk group and is only executed when a drive failure occurs. Snapshot and cloning have the goal of replicating logical volumes at some point in time and are executed by user request, not because of a failure. Table 2.2 shows a comparison of the data protection policies.

Table 2.2: Comparison of data protection policies

	Purpose	**Time of execution**	**Procedure used**	**Data level affected**
Sparing	Reconstruct the redundancy of data in RAID disk group	When a drive failure occurs	The data that was on the failed drive is read from the surviving disks	RAID level
Snapshot	Replicate the data stored in a logical volume at some point in time	On user request	Copy only modified parts of a source logical volume to a backup logical volume	Logical volume
Cloning	Replicate the data stored in a logical volume at some point in time	On user request	Copy an entire source logical volume to a backup logical volume	Logical volume

2.1.5 SPARING DATA PROTECTION POLICY

Sparing is the data protection policy that is executed when a drive fails and the data on the failing drive loses its RAID level redundancy. This policy restores the redundancy of the data that was stored on the failing disk by copying the non-redundant data to the surviving disks, therefore restoring the redundancy of that data. This policy operates at the RAID level, i.e., this policy does not create new logical volumes, only ensures that all data blocks that lack

redundancy are copied so they have redundancy again according to their RAID level. *Sparing* is also known as *Rebuild*. The SNIA defines rebuild as *the regeneration and writing onto one or more replacement disks of all of the user data and check data from a failed disk in a mirrored or RAID array. In most arrays, a rebuild can occur while applications are accessing data on the array's logical volumes.*

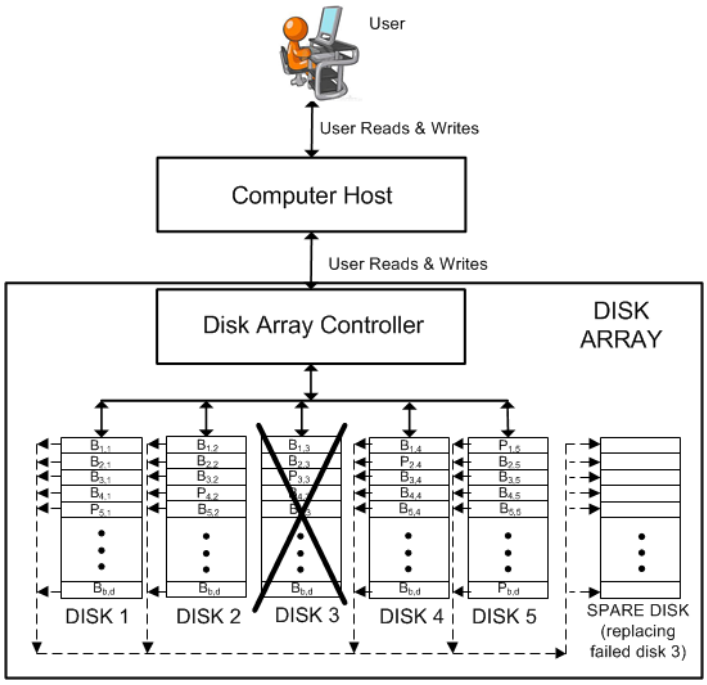

Fig. 2.5: Sparing data protection policy example

Fig. 2.5 shows an example of a RAID 5 disk group with a group size of G = 5 disks. In this example the disk labeled "DISK 3" failed and the regeneration of the data that was stored on the failed disk is being regenerated from the parity and data from the surviving four disks of this disk group. The dotted lines show the copy of data from the surviving disks to the spare disk. The copy of data to the spare disk recovers the RAID level redundancy lost by the failed disk. The sparing process is executed in the background and therefore is considered a background job. Chapter 4 presents a fuzzy control scheme for sparing and explains the sparing process in more detail.

2.1.6 Point-In-Time Data Protection Policy

Disk arrays protect the data in logical volumes using a Point-In-Time (PIT) data protection policy. The SNIA defines the Point-In-Time copy as *a fully usable copy of a defined collection of data that contains an image of the data as it appeared at a single instant*

in time. A PIT copy is considered to have logically occurred at that point in time, but implementations may perform part or all of the copy at other times (e.g., via database log replay or rollback) as long as the result is a consistent copy of the data as it appeared at that point in time. Implementations may restrict point in time copies to be read-only or may permit subsequent writes to the copy. The snapshot and cloning data protection policies are PIT data protection policies that are now standard features of disk arrays.

2.1.7 Snapshot Data Protection Policy

Snapshot or *Delta Snapshot* is a Point-In-Time data protection policy. By using the snapshot feature, users can create a point-in-time copy of a logical volume. From the user's standpoint, the snapshot feature creates an instant copy of the original logical volume. This gives users the means to preserve a point-in-time copy (the snapshot) of the data in a source logical volume. If the data in the source gets corrupted or lost, the user can go back to the snapshot and recover the data from that point in time. The SNIA defines delta snapshot as *a type of point in time copy that preserves the state of data at an instant in time, by storing only those blocks that are different from an already existing full copy of the data.*

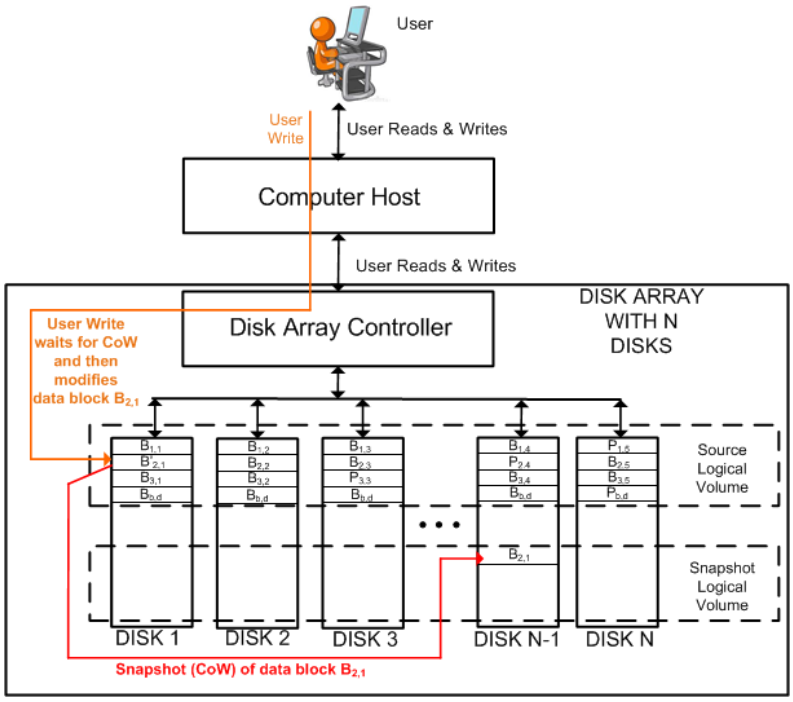

Fig. 2.6: Snapshot data protection policy example

Snapshot is a data protection feature that produces a point-in-time copy of a logical volume. The logical volume data blocks are copied *on-demand* when a user modifies a data block by writing to it. The data block is replicated before allowing the user write to proceed. A Copy-on-Write (CoW) takes place when a data block has to be copied before a user write can proceed on said data block. The example in Fig. 2.6 shows a *source* logical volume that is protected by the snapshot policy. The original volume with the data to be replicated will be referred to as the source volume or just the *source*, for short. The copy of the original volume will be referred to as the snapshot or replica volume or the *snapshot*, for short. The user writes to data block $B_{2,1}$ but since that data block has not been copied (snapped) then the user write has to wait for the CoW to proceed to copy the data block to the *snapshot* logical volume.

The snapshot data protection feature is space-efficient by only copying the modified (written to) data blocks but it impacts user request latency by forcing a user write to wait for a data block to be copied if the data block has not been copied before. Also, the source and the snapshot logical volumes are attached (linked) because the snapshot logical volume only contains the modified data blocks and the rest of the data blocks are still in the source logical volume. Chapter 5 presents a fuzzy control scheme for snapshot and explains the snapshot data protection policy in more detail.

2.1.8 Cloning Data Protection Policy

Cloning, like snapshot, is a Point-In-Time data protection policy; but cloning, unlike snapshot, is not an on-demand data protection policy. Cloning does not wait for the user to modify (write to) a data block to copy it. Cloning copies all the logical volume data regardless of the state of the data blocks modified or unmodified. From the user's point of view, the cloning replication takes some time because the cloning replication copies all the data in a logical volume. This replication of data blocks gives users the means to preserve a point-in-time copy (clone) of the data from a source logical volume. The original volume with the data to be replicated will be referred to as the source volume or just the *source*, for short. The copy of the original volume will be referred to as the clone or replica volume or the *clone*, for short.

This data protection feature is not space-efficient like snapshot but it provides complete separation of the source and clone logical volumes. Users choose this feature because unlike snapshot, when cloning finishes replicating the source logical volume, they

can operate on each logical volume (source and clone) separately, since both logical volumes have all the data blocks that were originally in the source logical volume.

The cloning data protection policy can impact user request latency due to the cloning background activity or CoWs generated by users writing to data blocks in the source logical volume during the cloning replication process. The example in Fig. 2.7 shows a *source* logical volume that is protected by the cloning policy. The dotted lines show the copy of all the data blocks from the source logical volume to the clone logical volume. The cloning process occurs in the background as already said and can be processed serially, i.e., one data block at a time) or in parallel, i.e., multiple data blocks being copied at a time. Chapter 6 presents a fuzzy control scheme for cloning and explains the cloning data protection policy in more detail.

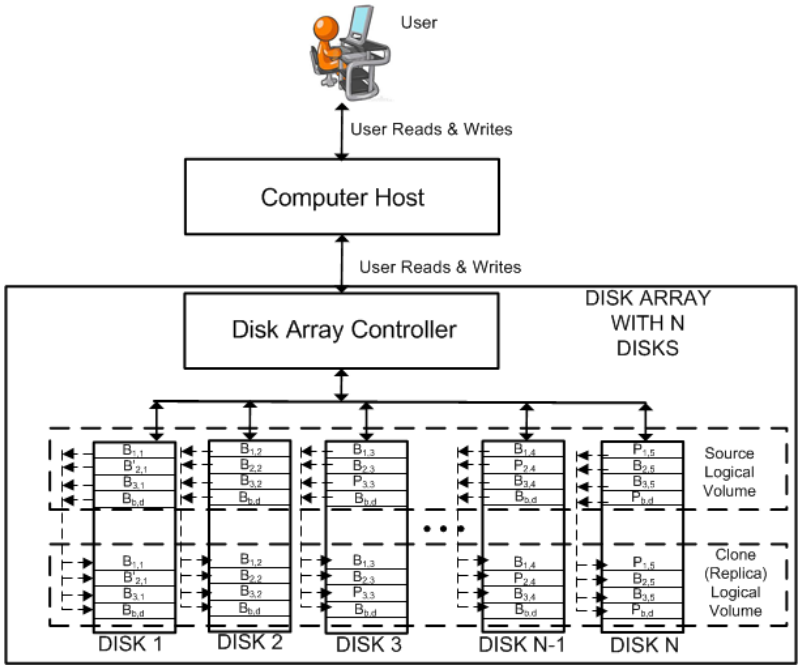

Fig. 2.7: Cloning data protection policy example

2.1.9 Disk Array Performability and Data Protection Policies

Sparing, snapshot, and cloning, are data protection policies that operate in the background, i.e., concurrently with the service of user reads and writes. Those three data protection policies make use of the same resources (CPU, disks, memory, IO ports) to make copies of data instead of serving user reads and writes. Therefore, those three data protection

policies can potentially impact the user read/write performance in both throughput and latency. The purpose of the fuzzy controllers presented by this dissertation in chapters 4, 5 and 6 is to minimize the performance impact of the data protection policies running in the background (background jobs), therefore, improving the performability of the disk array under those three data protection policies. The performability approach of chapter 4, 5 and 6 is background-jobs based. In those chapters the purpose is to improve the performance of the user services under non-optimal conditions, i.e., in the presence of a background job that can be sparing, snapshot or cloning.

2.2 PERFORMABILITY

2.2.1 PERFORMABILITY OF DISK ARRAYS

In section 1.1, a general performability definition was presented. This section will refine the definition of performability, but first, effectiveness is defined. Effectiveness is the ability of a system to meet its specified needs. A quantitative measure of effectiveness (MoE) was proposed by Smith and Clark in [Smith 04]. The definition of performability has evolved since it was first proposed by Meyer [Meyer 78a],[Tai 96a]. Meyer first defined performance as *the effectiveness of an object to deliver a specified service in a time interval [0,t]*; and reliability as *the probability of an object to deliver a specified service in a time interval [0,t]*. With these two definitions, Meyer then defines performability as *the unification of performance and reliability.* We consider this definition of performability a *reliability-based performability.*

As described in section 1.1, the concept of performability was extended by Zhang et al. [Zhang 06a] to consider background tasks or jobs besides failure conditions. Zhang goes over the use of background jobs in disk arrays but did not present a formal definition of background jobs. Therefore, for this dissertation we propose two definitions for disk arrays. We define a foreground task (job) as *an interactive task with a hard, short deadline*. We define a background task (job) as *a non-interactive task with a soft deadline that is independent of the foreground jobs*. We consider the performability proposed by Zhang as the *background-jobs based performability*. We define the background-jobs based performability as *the measure of the probability of the performance impact on foreground tasks caused by the execution of background tasks*.

It is possible to unify both Meyer's and Zhang's definitions of performability for disk arrays if we consider the performability from the point of view of foreground jobs. For a foreground job the *optimal condition* exists when there are neither failures nor background jobs present in a disk array. Conversely, we can consider that for a foreground job a *non-optimal condition* exists when there is either a failure or a background job in the disk array.

Based on the definitions of optimal and non-optimal conditions, we define performability of disk arrays as *a measure of the probability of user requests to achieve a performance level under non-optimal conditions*.

2.2.2 FUNDAMENTAL CONCEPTS

The system under performability evaluation is referred to as the *total system S*. For performability evaluations a total system *S* is considered to have an *object system C* and an *environment E*. The object system *C* is the system or component that provides the service and is the object of the performability evaluation. The environment *E* is the system of components or events that affect the ability to perform of the object system *C*. A good example of *E* is the workload that the environment *E* applies to *C*.

The dynamics of the object system *C* are modeled by a stochastic process referred to as the object system model X_C:

$$X_C = \{X(S_C, t) \mid t \in T\} \tag{2.5}$$

where $X(S_C,t)$ is a random variable with sample space S_C and index *t*, where *t* is time and is in the range of the total interval of time *T*. The dynamics of the environment *E* are also modeled by a stochastic process referred to as the environment system model X_E:

$$X_E = \{X(S_E, t) \mid t \in T\} \tag{2.6}$$

where $X(S_E,t)$ is a random variable with sample space S_E and index *t*, where *t* is time and is in the range of the total interval of time *T*. Modeling the environment *E* with a stochastic process can be optional. Sometimes the environment can be replaced with a set or list of values used as inputs to the object system *C*.

The stochastic process is defined by a set of events based on a probability space. The events are also referred to as states in stochastic models such as in Markov models.

The stochastic process developed to model X_C can consist of a set of states Q_C. The stochastic model developed for X_E can also consist of a set of states Q_E. A stochastic model that includes both Q_C and Q_E makes use of the product space Q, which is the space product of Q_E and Q_C:

$$Q = Q_C \times Q_E \tag{2.7}$$

The total system S can be modeled using a stochastic model X that includes both X_C and X_E:

$$X = (X_C , X_E) \tag{2.8}$$

The stochastic process X makes use of the state space Q. Some common stochastic models used to model X and estimate performability are:

1) Markov Chains (MC) with rewards (Markov Reward Models)
2) Queueing models
3) Stochastic Petri Nets (SPNs)
4) Series-parallel graphs
5) Simulations packages such as CSIM

Depending on the specific performability evaluation, sometimes only the object system model X_C and the states Q_C are used:

$$X = X_C \qquad Q = Q_C \tag{2.9}$$

Or only the environment model X_E and the states Q_E are used:

$$X = X_E \qquad Q = Q_E \tag{2.10}$$

The second important definition in performability is the *performability variable Y*, a random variable from the stochastic model *X*. The third important definition in performability is *accomplishment*: A desired value or range of values for the performability variable *Y* is defined, usually at design time. The performability variable *Y* and the accomplishment *A* are related by the probability of achieving the accomplishment:

$$Perf(A) = \Pr[A_L \leq Y \leq A_H] \tag{2.11}$$

The *Perf(A)* is the performability of the accomplishment *A* and is defined as the probability of the performability variable *Y* to be in the range of the values of accomplishment *A*. A_L (A_H) is the lower (high) value of *A*. This definition comes from the intention of performability evaluations to obtain the probability density function, and therefore the cumulative distribution function. If the cumulative distribution can be obtained then the accomplishment can be expressed as:

$$Perf(A) = \Pr[Y \leq A] \tag{2.12}$$

The *Perf(A)* is also referred to as the *performability measure*.

2.2.3 Performability Evaluations

A performability evaluation consists of the three parts mentioned in the previous section as part of the fundamental concepts: 1) a model of the system or feature under study with a stochastic process *X*, for example, a Markov model; 2) a performability variable *Y*, for example, probability of completing a service, financial benefit, latency, or throughput; 3) a value or range of values for the accomplishment *A*. The result of the performability evaluation is a relation between the performability variable *Y* and some other variable; usually time, but the result can also be *Y* vs. an input to the system under study. This result is most commonly presented as an x-y graph with time or an input to the system on the horizontal axis and the performability variable *Y* on the vertical axis. The three parts are related in the following way:

1) By the performability model (PM), which is the pair (*X,Y*):

$$PM(Y) = (X, \mathrm{Y}) \tag{2.13}$$

2) By the performability of the accomplishment *A*, as shown in (2.12)

It is important to state that *solving a performability problem means to use the stochastic process X as the approach to obtain values for the variable Y of Perf(A)*. The best solution is to obtain the probability density function of *Y* but that may be difficult and sometimes one or two moments is the acceptable solution. Solving a performability problem is also known as doing a *performability evaluation* or *estimation*.

2.2.4 PERFORMABILITY MEASURES

Different mathematical models have been used for performability analysis. Markov Chains (MC), Series-Parallel Graphs, Stochastic Activity Networks (SANs) and Markov Reward Models (MRMs) have been used, among others. The MRM has been one of the most used models for performability estimation.

The MRM will be used as the base model to explain the performability concept in a more mathematical form. MRMs extend the Markov Model by attaching a value *r* to each state of a Markov Model. This is referred to as *state reward* or simply *reward* and can be constant or time dependent. The reward value is what makes the MRM very suitable for performability analysis because the performability variable Y can be represented or estimated by a reward value or combined reward values of an MRM.

Let *N* be the number of possible states a system can operate in. The state of the system is defined by a time stochastic process $X = \{X(t),\ t \geq 0\}$. The state reward r_i is specified by some performance measurement Examples of reward measures are throughput and latency (response time). The reward can be time dependent $r(t)_i$ or time independent r_i. The random variable

$$Z(t) = r_{X(t)} \tag{2.14}$$

Is the *instantaneous reward rate* of a MRM at time *t*. There is a difference between reward rates r_i associated with individual states and the overall reward rate *Z(t)* of the MRM

characterizing the whole stochastic process. With this definition, the *cumulative performability*, *Y(t)*, can be:

$$Y(t) = \int_0^t Z(\tau)\, d\tau = \int_0^t r_{X(\tau)}\, d\tau \tag{2.15}$$

The general definition of performability by Meyer was [Meyer 78a]:

$$Perf(y,t) = P[Y(t) \le y] \tag{2.16}$$

The probability at time *t*, of being in state *i* is denoted by $\pi_i(t)$. The *transient performability* (TP) is defined as in [Bolch 06a],[Haverkort 01a]:

$$E[Z(t)] = TP(t) = \sum_{i=1}^{N} \pi_i(t)\, r_i \tag{2.17}$$

The expected reward rate when $t \to \infty$ is:

$$E[Z(\infty)] = SSP = \sum_{i=1}^{N} \pi_i\, r_i \tag{2.18}$$

This is the measure used as the steady state performability (SSP) [Haverkort 01a]. The π_i is the steady state probability of the state *i* in *N*.

2.2.5 Performability Example

We present an example that illustrates the performability concept and estimation. This example makes use of the MRM presented in section 3.1 and the equations in section 3.5. The reader is advised to read those two sections first.

For this example we assume we have a disk array that we want to use as a video server. Each video stream corresponds to one user. Therefore, the number of streams equals the number of users. Each user requires 8 Mbps (Megabits per second) of bandwidth. This translates into 1MB/s (Megabyte per second) of bandwidth per user. Each user (stream) is

charged $3.99 as long as we can provide 1MB/s of bandwidth for each user. The goal of this example is to evaluate the probability of loss in revenue up to 15% with respect to the original revenue when the servers start providing service for the first time (t = 0), and we want to estimate the time t when the 15% percentage of loss may occur. For this example, the parameters for the performability estimation are:

BW_u = 1MB/s ≡ Bandwidth per user

M_c = $3.99 ≡ Membership price

T = time period of the 15% loss in revenue. The time is expressed in months.

Rev(0) ≡ Original revenue obtained from the media server at the beginning of the performability evaluation (t = 0)

Rev(t) ≡ Current revenue obtained from the media server at time t. The time t will be expressed in months.

L(t) ≡ Loss in revenue with respect to *R(0)* at time t. The time t will be expressed in months.

The loss in revenue at time t, *L(t),* with respect to the original revenue *R(0.)* is:

$$L(t) = Rev(0) - Rev(t) \tag{2.19}$$

For this example, a disk array with the following parameters is considered:

λ = 1 failure / 50,000 hours = 0.00002 failures/hour

μ = 1 repair / 24 hours = 0. 0417 repairs/hour

N = 200 total disks in the disk array

G = 4 disks per disk group

BW_d = 25MB/s ≡ Bandwidth provided by each of the N disks.

S_d ≡ Number of streams supported by each of the N disks

$S_d = BW_d / BW_u$ = 25MB/s / 1MB/s = 25

D_s = 0.15 ≡ Percentage of performance degradation suffered by the disk array performance while sparing non-redundant data due to a drive failure.

The performability variable Y is the loss in revenue at time t, *L(t)*:

Performability variable Y ≡ Loss in Revenue ≡ L(t)

The accomplishment *A* is that loss in revenue should be at most 15% for the first 36 months.

Accomplishment A ≡ 15% of loss in revenue in the first 36 months

The performability measure, *Perf(A)*, is then:

Perf(A) = Pr [Y ≤ A] ≡ Probability of having a loss in revenue of up to 15% in the first 36 months.

In this example the disk array is the object system *C*. To define the object system model X_C, we use the Markov Model of a RAID disk group that is presented in section 3.1. That Markov Model defines three states for the state space Q_C. The equations for the Markov Model used as the stochastic process X_C are presented in section 3.5. The estimation of the probability of the three states Q_C of the Markov Model could be accomplished with the equation presented in section 3.5. We are not modeling the environment with a stochastic process X_E, therefore for this example $X = X_C$.

The reliability at time *t* (in months) of the model is defined as the probability of the Markov Model of the RAID group to be in S_0 and S_1:

$$Rel(t) = P_{S0}(t) + P_{S1}(t) \tag{2.20}$$

The probability of failure at time t (in months) of the model is defined as unreliability or the complement of the reliability:

$$Fail(t) = 1 - Rel(t) \tag{2.21}$$

The unreliability is the probability of having a second fault on the same disk group while the RAID group is trying to reconstruct the data of the first fault. The reward function r_0 is defined as:

$$r_0 = NS_d M_c \tag{2.22}$$

where r_0 is the reward of state S_0, N is the number of disks in the disk array, S_d is number of streams supported by each disk in the disk array and M_c is the membership cost ($3.99). The reward function r_1 is defined as:

$$r_1 = (N-1)(1-D_s)S_d M_c \tag{2.23}$$

where r_1 is the reward of state S_1, N is the number of disks in the disk array, S_d is number of streams supported by each disk in the disk array and D_s is the percentage of performance degradation due to the disk array executing the sparing data protection policy in the background. The reward function r_2 is defined as:

$$r_2 = 0 \tag{2.24}$$

The transient performability $TP(t)$ is used to estimate the revenue at time t, $R(t)$, based on the probability of the two states S_0 and S_1:

$$TP(t) = Rev(t) = \sum_{i=0}^{1} P_{Si}(t) r_i \tag{2.25}$$

Once the equations are applied for a period of 40 months (t = 0,1,…,40), the results of the revenue per month based on the state rewards and their probabilities are computed. Based on the revenue per month the performability variable Y, the loss in revenue can be estimated.

First, Fig.2.8 shows how the revenue drops monthly according to the reliability of the disk array for this example. It can be seen that after 36 months the revenue per month drops 15% down to $17,000.

Second, Fig.2.9 shows the probability loss in revenue drops according to the reliability of the disk array.

Fig. 2.8, shows that the initial revenue, *R(0)* is $20,000 per month. Therefore, a 15% loss in revenue would be a $20,000 x 0.15 = $3,000 loss. The performability evaluation, as shown by Fig. 2.9, provides the answer to the performability measure: that the probability is 13% of *Y*, the loss in revenue, to be lower than or equal to the accomplishment *A*, which is $3,000.

$$Perf(A) = \Pr[Y \le A] = \Pr[Y \le \$3{,}000] = 0.13 \tag{2.26}$$

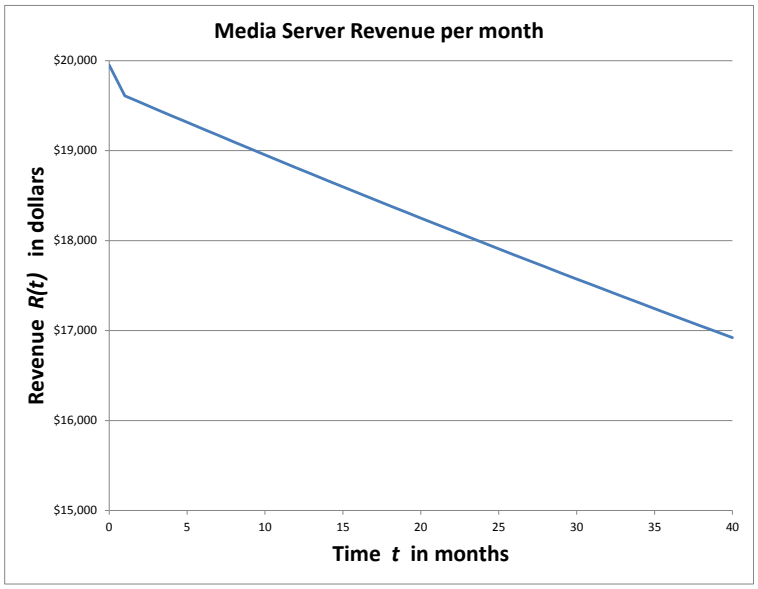

Fig. 2.8: Revenue per month for performability example

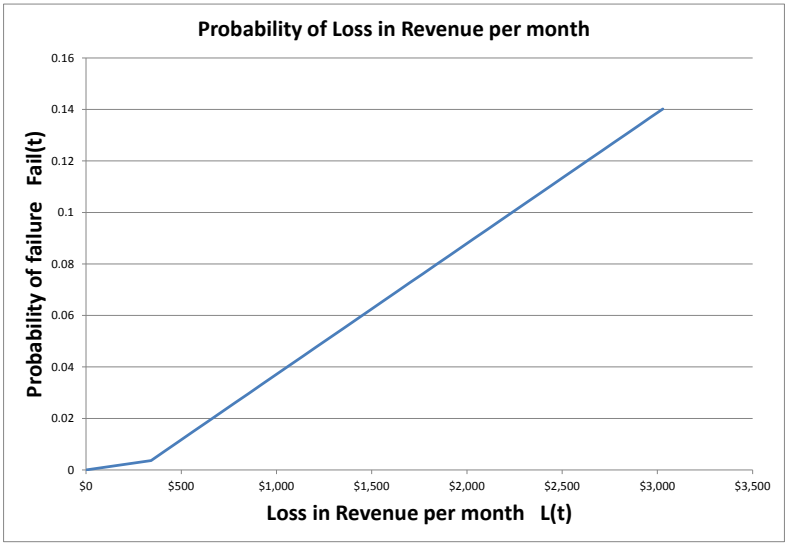

Fig. 2.9: Loss in revenue per month for performability example

2.3 FUZZY CONTROL

2.3.1 FUZZY NUMBERS AND ARITHMETIC

A crisp set *A*, defined in terms of a relevant universal set *X* can be described, according to classical set theory, in several ways including listing all of its members, providing a conditional description of all members of *A*, or by specifying a certain binary characteristic function such as $\mu_A \in \{0,1\}$, in which an element *x* either completely belongs to set *A* or it does not. Therefore, this crisp set *A* can be described as:

$$\mu_A(x) = \begin{cases} 1, & if\ x \in A \\ & \\ 0, & if\ x \notin A \end{cases}, \quad \forall x \in X \tag{2.27}$$

This kind of belonging to a set will be referred to as *crisp membership*. The set theory that assumes crisp membership will be referred to as *crisp set theory*.

An example of a classical set could be a set A = {1, 3, 5, 7}. In this example, we can say that the number 3 is part of the set A, thus, $\mu_A(3) = 1$. The number 3 belongs in the set A, while $\mu_A(4) = 0$.

Unlike classical set theory, fuzzy set theory states that an element can have a *degree* of belonging to a particular set. Fuzzy set theory can be seen as a generalization of crisp set theory, because the degree of belonging of an element *x* to set *A* is determined by a membership grade $\mu_A(x)$ taking on value from the unit interval [0, 1]. The fuzzy set *A* in the universe of discourse *X* can be defined as a set of ordered pairs of element *x* and its degree of membership $\mu_A(x)$:

$$A = \left\{ \left(x,\ \mu_A(x)\right) \,\middle|\, x \in X \right\} \tag{2.28}$$

The fuzzy set concept arose from the need to deal with imprecise data. A fuzzy set *A*, is defined in terms of a relevant universal set *X*, by a membership function. This function assigns to each element *x* of *X* a number $\mu_A(x)$, in the closed unit interval [0,1] that characterizes the degree of membership of *x* in *A*. Membership functions are functions of the form $\mu_A(x)$: X→[0,1]. The reader is referred to [Klir 95a], [Hanss 10a] for a complete

treatment of the definition and representations of fuzzy sets and fuzzy numbers. In this document triangular fuzzy numbers will be used. The representation used in this document for a triangular fuzzy number is $A=[x_l,x_c,x_h]$, where x_l is the low value, x_c is the central value and x_h is the high value (Fig. 2.10).

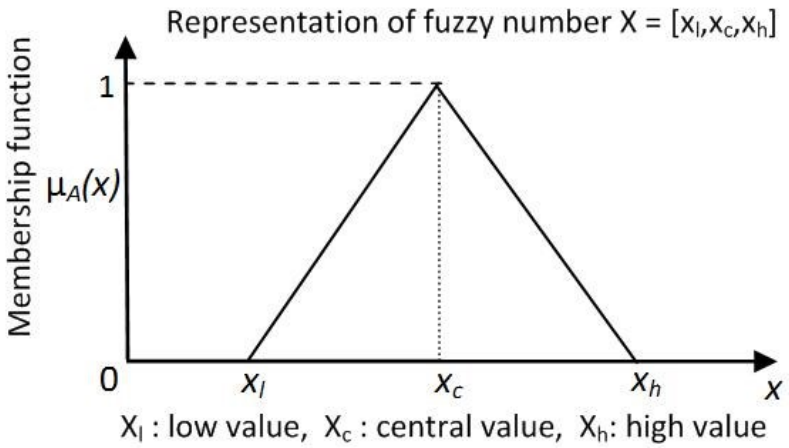

Fig. 2.10: Triangular fuzzy number *A=[xl, xc, xh]*

The two basic methods to compute fuzzy arithmetic are: 1) extension principle and 2) α-cuts and interval arithmetic. Fuzzy numbers can be discretized so they can be represented as a finite set of $(x_i^{(l)},\mu_i)$ and $(x_i^{(r)},\mu_i)$ tuples. There is the value of x for the μ_i on the left side of the central value (apex), in the case of a triangular fuzzy number, and $x_i^{(r)}$ is the value of x for the same μ_i on the right side of the apex. With this, discrete fuzzy sets for which the fuzzy arithmetical operations can be defined using Zadeh's extension principle. One approach to discretized a fuzzy number is to split the μ-axis into a number of equally space n segments, each with $\Delta\mu = 1/n$. The fuzzy number then is turned into a discrete fuzzy number that can be represented in the form shown in Fig. 2.11. The fuzzy number A, then can be discretized in a form proposed in [Hanss 10a].

$$\mu_i = \mu_A(x_i^{(l)}) = \mu_A(x_i^{(r)}) \tag{2.29}$$

$$A_d = \{(x_0^{(l)},\mu_0),\ldots,(x_n^{(l)},\mu_n),(x_0^{(r)},\mu_0),\ldots,(x_n^{(r)},\mu_n)\} \tag{2.30}$$

$\mu_i = \mu_{i-1} + \Delta\mu$, $i = 1,\ldots,n$, where $\mu_0 = 0$ and $\mu_n = 1$. (2.31)

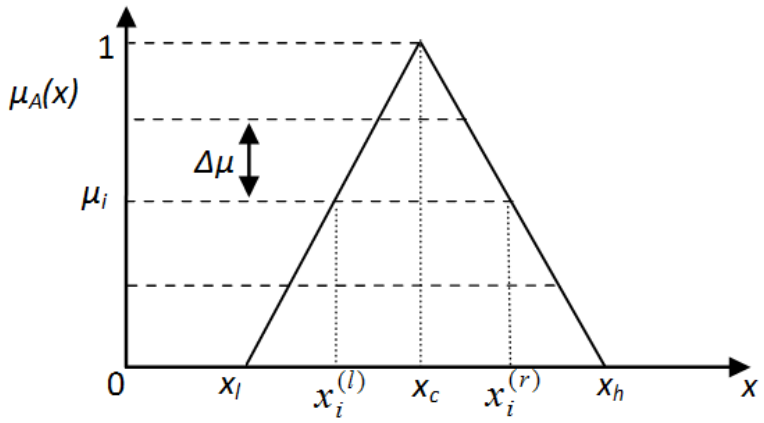

Fig. 2.11: Triangular fuzzy number A=[x_l, x_c, x_h] and its discretization

In [Hanss 10a], it is shown that using the discretized fuzzy numbers as in (2.30) and (2.31), the arithmetical operations can be implemented by defining the operations to be executed separately for the elements of each degree of membership μ_i. The arithmetical operations can be implemented by combining only the elements of the low (left) and right (high) value side of the apex (central value) of the triangular fuzzy number. The four basic arithmetic operations are implemented in the following form:

$$z_i^{(l)} = x_i^{(l)} \otimes y_i^{(l)} \ , \ z_i^{(r)} = x_i^{(r)} \otimes y_i^{(r)} \text{ and } i = 0,1,2,\ldots,n \tag{2.32}$$

where $\otimes$ represents the four basic arithmetic operations $(+,-,\times,/)$. For a complete explanation, the reader is referred to [Hanss 10a].

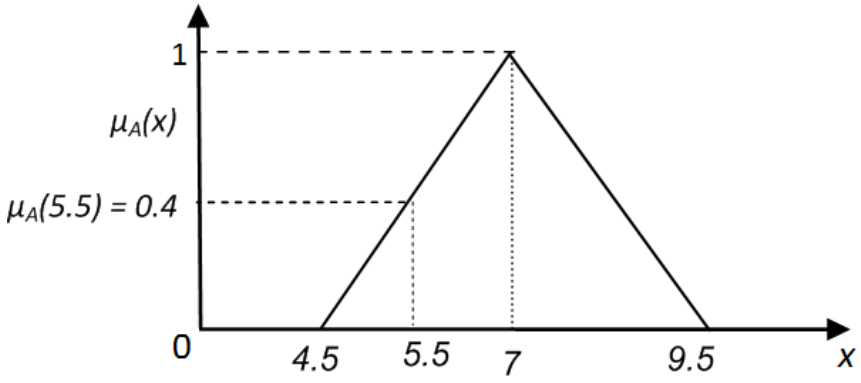

Fig. 2.12: Triangular fuzzy number SEVEN=[4.5,7,9.5]

An example of a fuzzy number is shown in Fig. 2.12. In this figure a triangular fuzzy number is defined as SEVEN=[4.5,7,9.5]. The degree of membership for the value x=5.5 is $\mu_{SEVEN}(5.5)=0.4$

2.3.2 JUSTIFICATION FOR FUZZY CONTROL

Fuzzy control can be considered an alternative to classical design for controllers [Michels 06a] which requires differential equations that model the system to control. In control theory the system to control is usually referred to as the *plant*. If a differential equation of the plant is available, then using classical control theory has advantages. A systematic mathematical process can be followed to predict the stability, robustness and response of the controller. It can be said that classical control is *model-based*.

Fuzzy control has a different approach to the control of the process because no model of the plant is constructed. Fuzzy control approaches the problem of controlling a plant by the design of rules. It can be said that fuzzy control is *rule-based*. Of course, the rules are not arbitrary. The rules are based on the available knowledge of the plant. The understanding of the plant can be analytical, heuristic (rule of thumb or educated guess), or a combination of both.

There are situations in which the components of a plant are ruled by complex algorithms and provided by manufacturers that do not reveal their algorithms. The manufacturers reveal only the external behavior of the products they sell, which is not usually enough to model a component using a differential equation. This is the case in disk arrays. The disks to be controlled are ruled by non-linear complex logic embedded in them. The only information available about the disks' behavior comes from the manuals, which usually do not cover the entire spectrum of conditions the disk will be subjected to, or by experiment. Thus, a different approach for control must be used for disk arrays.

Fuzzy control is based on heuristics [Michel 06a] and can be applied successfully in situations where classical control would difficult or impossible to apply. The term "plant" is used in control theory to refer to the system or component to control. Using fuzzy control makes the most sense when [Michels 06a]:

1. No model of the plant exists in a differential or difference equation form.
2. The behavior of the plant is non-linear.

3. The goals are fuzzy, e.g., "ensure a proper latency of user requests and a proper completion time for disk repair when both are executed concurrently"
4. The plant and the control strategy are simple enough that the design of a fuzzy controller takes less time than the classical controller modeling and design.

In addition to this, fuzzy logic opens up the possibility of using other computational intelligence techniques such as neural networks for the performability control of disk arrays.

2.3.3 FUZZY LOGIC CONTROLLER

The first model of fuzzy controller was introduced by Mamdani [Mamdani 75a]. Like a classical controller, a fuzzy controller takes crisp inputs from the plant and a reference or references to compare against. Also, like a classical controller, the fuzzy logic controller (FLC) produces crisp control outputs that control the process in the plant. There are four parts to a fuzzy controller that must be designed: fuzzifier, rule base, inference engine and defuzzifier. Fig. 2.13 shows the block diagram of an FLC. Fuzzy inputs and outputs are fuzzy numbers, which means that the numbers have a degree of membership to a particular set.

2.3.4 FUZZY LOGIC CONTROLLER: FUZZIFIER

The fuzzifier performs the fuzzification of the crisp control inputs. There are two types of crisp inputs:

1) Outputs from the plant that are fed back into the control scheme to compute the difference with respect to the reference(s).
2) Parameters of the plant, $x_1,\ldots,x_n$. These are known as *state parameters* or *state variables*.

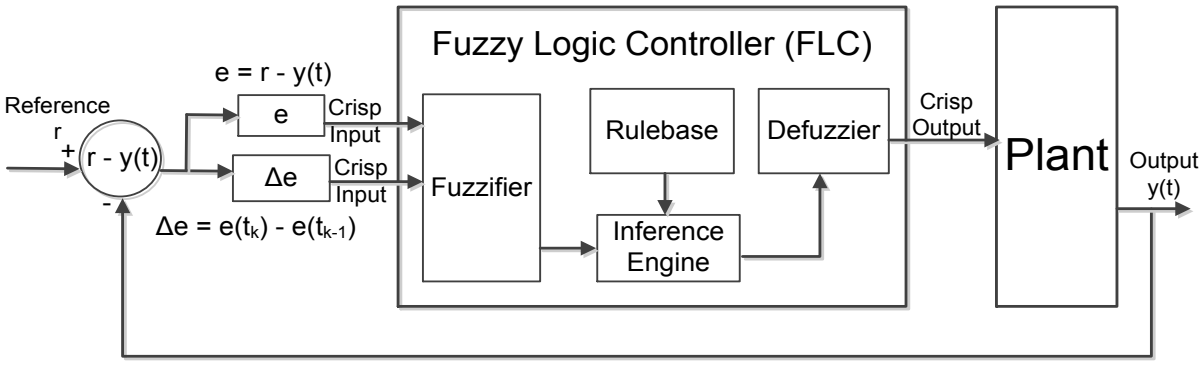

Fig. 2.13: Fuzzy Logic Controller (FLC) model with error computation

The fuzzifier performs the fuzzification of the crisp control inputs into fuzzy values. For example, a FLC can accept an error input *e* for which three fuzzy sets are defined: big error (BE); medium error (ME) and small error (SE). The crisp input error *e* can be computed subtracting the fed back output from the plant with a reference *r*. Then the crisp input error *e* can be fuzzified by computing its degree of membership (number between 0 and 1) in each of three fuzzy sets mentioned. Fig. 2.14 shows an example.

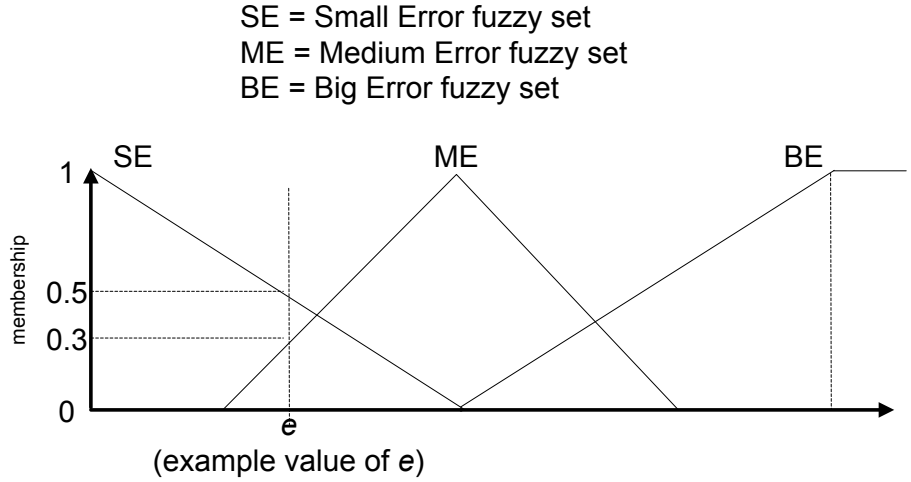

Fig. 2.14: Fuzzification of a value of error *e*

2.3.5 FUZZY LOGIC CONTROLLER: RULE BASE

A rule base contains the knowledge related to the particular control model. It contains the control actions (rules) in the form of if-then-conclusion statements. These statements use the fuzzy values provided by the fuzzifier. It can be said that the rules provide policies [Zhang 05a] for the control of the specific process or system to control.

2.3.5.1 MAMDANI CONTROLLERS

The Mamdani controllers were introduced by Mamdani in 1975 [Mamdani 75a]. They comprise a finite set of rules of the form

$$R_i: \text{ if } x_1 \in FS_1 \text{ and} \ldots \text{and } x_n \in FS_n \text{ then } y \in FS_{Ri} \tag{2.33}$$

R_i is rule i; FS_i are fuzzy sets; $x_1 \ldots x_n$ are input variables and y is the output variable.

For example, the error e and the change in error Δe can be compared to the fuzzy sets shown in the previous section. We can also define a fuzzy set named small output, SO, for the output y and we could build a rule like this:

$$R_i : \text{ if } e \in \text{SE and } \Delta e \in \text{SE then } y \in \text{SO} \tag{2.34}$$

This would say that if the e and the Δe have small values, i.e., belong to the SE fuzzy set, then the output value y should belong to the SO fuzzy set.

2.3.5.2 TAKAGI-SUGENO-KANG CONTROLLERS

The Takagi-Sugeno-Kang (TSK) controllers were introduced in 1985 [Takagi 85a]. The TSK controllers have rules of the form:

$$R_i : \text{ if } x_1 \in FS_1 \text{ and} \ldots \text{and } x_n \in FS_n \text{ then } y = f(x_1, \ldots, x_n) \tag{2.35}$$

where the difference with respect to the Mamdani rules (2.33) is that the output can be a mathematical function using crisp values for both the inputs $x_1 \ldots x_n$ and the output y.

For example, the error e and the change in error Δe can be compared to the fuzzy sets shown in the previous section. We can also define a function $f(y) = x_1 + 0.5$ for the output y in which x_1 is a state variable. Then we could build a rule like this:

$$R_i : \text{ if } e \in \text{SE and } \Delta e \in \text{SE then } y = x_1 + 0.5 \tag{2.36}$$

This would say that if the e and the Δe have small values, i.e., belong to the SE fuzzy set then the output value y should be the crisp value $x_1 + 0.5$.

2.3.5.3 TABULAR REPRESENTATION OF FUZZY CONTROLLERS

Rule bases can be represented in a tabular format, which is used extensively in the fuzzy control literature. Each row of the table represents a rule. The columns represent the input variables and the leftmost column represents the output. Table 2.2 shows an example of

a tabular representation of a TSK controller based on the rules of the form in (2.36). In this example there are nine rules R_i and the function $f(y)$ to be applied for the output y depends on the rule that becomes valid according to the fuzzification of the input values e and Δe.

Table 2.2: Example of tabular representation of a TSK controller

	Inputs		Output
Rule	e	Δe	y
R1	SE	SE	$y = x_1 + 0.5$
R2	SE	ME	$y = x_1 + 1$
R3	SE	BE	$y = x_1 + 1.5$
R4	ME	SE	$y = x_1 + 2$
R5	ME	ME	$y = x_1 + 2.5$
R6	ME	BE	$y = x_1 + 3$
R7	BE	SE	$y = x_1 + 3.5$
R8	BE	ME	$y = x_1 + 4$
R9	BE	BE	$y = x_1 + 4.5$
SE=Small error	ME=Medium error		BE=Big error

2.3.6 Fuzzy Logic Controller: Inference Engine

The inference engine performs the evaluation of all rules in order to choose the result that will become the fuzzy output of the controller. One technique is to aggregate all the rules in one fuzzy relation; this is known as *composition inference*. The composition inference is not a common technique in fuzzy control. The most common technique is to compute each rule individually using min or product t-norms [Hanss 05a] and compute the output based on the individual results from each rule by using the max (supremum) s-norm [Hanss 05a] This technique is known as *individual rule firing*. [Zhang 05a]

2.3.7 FUZZY LOGIC CONTROLLER: DEFUZZIFIER

The defuzzifier converts the fuzzy output of the inference engine into a crisp number that can be used as a control value for the plant to control. The deffuzification depends on the type of fuzzy controller.

For the Mamdani controllers, the rules can be converted to a crisp value using the center of gravity method [Zhang 05a][Michels 06a]. When the output of the inference engine is a fuzzy set, there is the problem of which value to use from the set. Several solutions have been proposed, but at the end the best option is to adapt the conversion to the specific control.

For the TSK controllers the output y is already a crisp value, therefore no defuzzification is needed.

CHAPTER 3: PERFORMABILITY ANALYSIS OF DISK ARRAYS USING FUZZY LOGIC

This chapter presents a numerical fuzzy logic performability model for disk arrays. The performability of disk arrays systems has been studied before in analytic form by presenting closed-form solutions of Markov Models [Islam 93a], [Barnett 98a]. The numerical method presented in this chapter is a simpler and adaptable alternative to closed-form solutions: simpler because it does not require the closed-form solution of a Markov Model, and adaptable because it can be adapted to particular conditions, e.g., a RAID level like RAID6 that supports double disk failure, or a function can be introduced to change the reward of the states, or make the reward time-dependent instead of fixed.

Also, in this chapter a performability analysis of a disk array used as an e-mail server is presented [Navarro 06a], [Navarro 07a]. We base the analysis on some of the rules of thumb for the configuration of an MSExchange Server 2003 [Microsoft 07a], [Microsoft 07b], [Microsoft 04a]. It is not claimed this document presents a complete performability study of a MSExchange e-mail server. Rather, based on a selected number of MSExchange configuration recommendations, the author demonstrates the following proof of concept: performability analysis enhanced by fuzzy arithmetic can be effectively used for a predictive performability analysis of an e-mail server, also referred to as mail server.

3.1 MARKOV MODEL OF A DISK ARRAY

For the purposes of the fuzzy performability analysis of disk arrays using fuzzy logic, a disk array with a total of N disks divided in groups of G disks is considered. The Markov Chain (MC) used for the reliability analysis of this configuration is shown in Fig. 3.1.

This Markov Chain does not consider the failure of other components of a disk array, such as controller failures. RAID reliability studies with the consideration of failure of components besides disks can be found in the literature [Schulze 89a].

The MC makes use of a parameter named disk failure rate, λ, number of failures per time unit. For example, if a disk fails one time in 1000 hours, the failure rate, $\lambda = 1$ failure / 1000 hours, or $\lambda = 0.001$ failures/hour. The inverse of the failure rate is the time to failure: $1/\lambda$ = 1000 hours / 1 failure = 1000 hours/failure

When a single disk fails, the disk array goes to the non-optimal state S_1. This implies the loss of the data redundancy. But the data is still not completely lost, since it is available on one of the *G-1* disks that are still working in the group. The data lost on the failed disk must be then rebuilt from the redundant data. The repair rate μ is referred to as the *repair rate* or *rebuild rate* and is measured in the number of repairs per time unit, for example, a data drive repair requires 10 hours to rebuild the data redundancy lost by a failed drive; then we can say a repair takes 10 hours and the repair rate μ = 1repair/10hours or μ = 0.1 repairs/hour. From this example we can clearly see that the ratio $1/\mu$ gives us the repair time. In the MC model, after a time $1/\mu$, the disk array completes the rebuild of the redundancy and the disk array goes back to state S_0 (back to the state with *G* working disks).

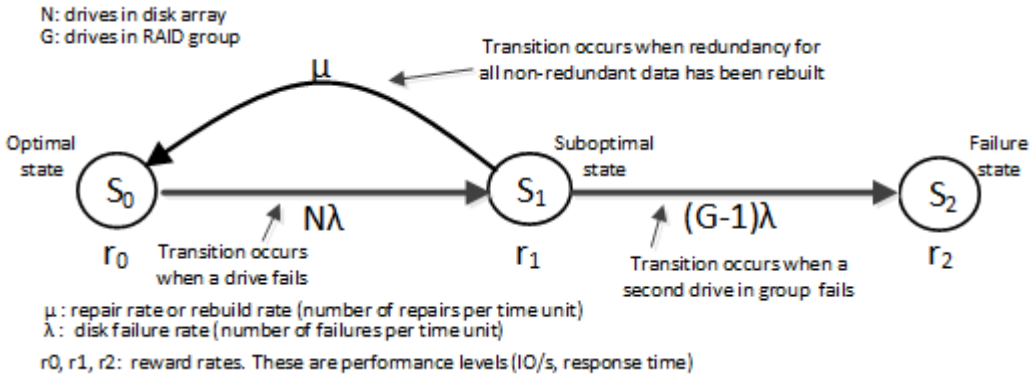

Fig. 3.1: Markov Reward Model of a RAID disk group

If during the time $1/\mu$ while the disk array is in state S_1 another disk within the disk group with the non-redundant data fails, the data is lost. In this case the disk array goes to failure state S_2. It can be said that the unreliability (or probability of failure) for a RAID disk group is nothing but the probability of a second failure in the same disk group. If this event occurs, the user must restore the data using the backup on tape or some other media. The MC shown in Fig. 3.1 is for a disk group with G disks and one parity disk. That is why it has three states. For RAID levels with two parity disks, like RAID6 [Patterson 94a], the number of states would be four. The reward of the states, r0, r1 and r2 are the reward values associated to each state. For the performability analysis presented in this chapter, the reward values of each state are performance levels that the disk array can deliver, e.g., IO/s, latency, number of Mailboxes supported by the disk array or number of Users supported by the disk array.

The system of differential equations for the Markov Model of the reliability of a disk array group (Fig. 3.1) is described via probabilities of being in state S_0, S_1 and S_2:

$$\begin{aligned}
\frac{dP_{S0}(t)}{dt} &= -N\lambda P_{S0}(t) + \mu P_{S1}(t) \\
\frac{dP_{S1}(t)}{dt} &= N\lambda P_{S0}(t) - [(G-1)\lambda + \mu]P_{S1}(t) \\
\frac{dP_{S2}(t)}{dt} &= (G-1)\lambda P_{S1}(t)
\end{aligned} \tag{3.1}$$

The system of differential equations gives rise to the following system of equations using the Laplace transform:

$$\begin{aligned}
P_{S0}(s) &= \frac{s + (G-1)\lambda + \mu}{s^2 + [(N+G-1)\lambda + \mu]s + N(G-1)\lambda^2} \\
P_{S1}(s) &= \frac{N\lambda}{s^2 + [(N+G-1)\lambda + \mu]s + N(G-1)\lambda^2} \\
P_{S2}(s) &= \frac{N(G-1)\lambda^2}{s\{s^2 + [(N+G-1)\lambda + \mu]s + N(G-1)\lambda^2\}}
\end{aligned} \tag{3.2}$$

With the system of equations (3.2), we find that the reliability of the disk array represented by the Markov Model from Fig. 3.1 is as (3.3):

$$R(s) = \frac{s + (G-1)\lambda + \mu + N\lambda}{s^2 + [(N+G-1)\lambda + \mu]s + N(G-1)\lambda^2} \tag{3.3}$$

By applying the Final-value theorem of Laplace transform we get:

$$MTTF_{GROUP} = \lim_{s \to 0} R(s) \tag{3.4}$$

We derive an equation that we can use as the $MTTF_{RAID}$.:

$$MTTF_{RAID} = \frac{(N+G-1)\lambda + \mu}{N(G-1)\lambda^2} \tag{3.5}$$

The rebuild process is performed automatically. Certainly, the failed disk must be manually replaced at some point [Patterson 88a].

Equation (3.5) can be verified against the equation proposed by Chen in [Patterson 94a]. If we have a high lambda, like λ=500,000 and a disk array with *N*=200 disks using RAID1, so *G*=2, and with a rebuild time of 8 hours, we have:

$(N+G-1)\,\lambda = (200+2-1) * (1/500{,}000) = 0.000402$

And $\mu = 1/8\text{hrs.} = 0.125$.

It is easy to see that $(N+G-1)\,\lambda << \mu$ and we can again make the same approximation made in Shooman [Shooman 02a] and remove the $(N+G-1)\,\lambda$ term. This turns (3.5) in

$$MTTF_{RAID} = \frac{\mu}{N(G-1)\lambda^2} \tag{3.6}$$

Equation (3.6) is the classical $MTTF_{RAID}$ estimation proposed by Patterson and Chen in [Patterson 94a]. We can use the Markov Model shown in Fig. 3.1 for the reliability estimation of the disk array.

If we consider a lower lambda, like λ=10,000 and again, a disk array with *N*=200 disks using RAID1, so *G*=2, and with a rebuild time of 8 hours. We have:

$(N+G-1)\,\lambda = (200+2-1) * (1/10{,}000) = 0.0201$

It is easy to see that in this case $(N+G-1)\,\lambda << \mu$ does not hold and we would have to use the (3.5) with all its terms for the estimation of $MTTF_{RAID}$. This is the same consideration as the MTTF equation obtained by Shooman in [Shooman 02a].

In order to estimate the system reliability we need to estimate the probability of the Markov Chain being in state S_i at time *t*. This probability is designated as $P_{Si}(t)$ and can be estimated by means of the initial probability vector $PS(0) = [P_{S0}(0), P_{S1}(0), \ldots, P_{Sm}(0)]$ of the $(m+1)$ states and the *state transition probability matrix* (TPM) of the Markov Model of the disk array. The transition probabilities among states S_0, S_1 and S_2 are shown in Fig. 3.1 and can be translated into the TPM matrix (3.7):

$$P = \begin{array}{c} \\ S_0 \\ S_1 \\ S_2 \end{array} \begin{array}{c} \begin{array}{ccc} S_0 & S_1 & S_2 \end{array} \\ \begin{bmatrix} 1-N\lambda\Delta t & N\lambda\Delta t & 0 \\ \mu\Delta t & 1-[(G-1)\lambda+\mu]\Delta t & (G-1)\lambda\Delta t \\ 0 & 0 & 1 \end{bmatrix} \end{array} \tag{3.7}$$

The initial probability of S_0 is $P_{S0}(0)$=1 while the initial probabilities for S_1 and S_2 are $P_{S2}(0)$=0, and $P_{S3}(0)$=0. Therefore, the initial probability vector is $PS(0)$=[1,0,0]. Failure rate (λ) and repair rate (μ) are assumed to be constant during the life of the disk array.

The estimation of probabilities of the states for the disk array was done during discrete iterations of time. Thus, the time t at which the probabilities of all states (S_0, S_1, S_2) was evaluated was using a value n that ranged from 0 to certain maximum value, i.e., n = (0,1,2,…,n_{max}). The time t was obtained by multiplying this value n by a time increment Δt (one hour delta for the example in this section). We estimated the reliability of the disk array every hour from 0 through n_{max} hours. The criterion to choose the hour-based discretization steps is consistent with disk manufacturers that provide their failure rates in hours.

The probabilities of all states $PS(t) = [P(t)_{S0},P(t)_{S1},P(t)_{S2}]$ at some time $t=n\Delta t$ was estimated using:

$$PS(n\Delta t) = P^n PS(0) \tag{3.8}$$

Once the probabilities $PS(t)$ are calculated, the reliability of the RAID system can be obtained as:

$$R(n\Delta t) = P_{S0}(n\Delta t) + P_{S1}(n\Delta t) \tag{3.9}$$

It can be seen from (3.9) that the unreliability of a disk RAID group is nothing more than the probability of having a second failure on a disk in the same disk group. The $P_{S2}(t)$ is the unreliability, i.e., the probability of the second failure.

3.2 Performability Model of Disk Arrays

The two performance measures used for the performability evaluation of the disk array were: 1) the throughput in IO/s (I/O requests per second) and 2) the number of mailboxes the mail server can support based on the performance and reliability.

The throughput that a disk array can deliver depends on three factors: 1) the total number of IO/s that can be delivered by the disks installed in the disk array; 2) the RAID level used, and 3) the ratio of reads and writes.

In order to estimate the IO/s a disk array with N disks can yield, a model for the throughput of a single disk must be used. The model used is based on [Patterson 07a] with some modifications. The average disk service time (τ_d) per I/O is estimated using the equation:

$$\tau_d = S_t + R_t + \frac{B_s}{\chi_b} \tag{3.10}$$

where S_t is the average seek time, R_t is the average rotational latency, B_s is the size of the transferred block of data, and χ_b is the bandwidth of the bus that connects the disk with the disk array controller. We are considering the same S_t for both reads and writes. Although in reality disks have different average seek times S_t for reads and writes, for the purposes of this analysis this simplification was made.

The inverse of the τ_d time gives us the throughput of one disk (χ_d) in IO/s:

$$\chi_d = \frac{1}{\tau_d} \tag{3.11}$$

This is another simplification, since the throughput of a disk also depends on the internal seek reordering algorithms [Patterson 07a]. The throughput of N disks is then:

$$\chi_d(N) = N\chi_d \tag{3.12}$$

The equations shown so far can be used to calculate the number of IO/s we can get from the disks in a disk array without considering the RAID level. For this document a RAID1 and a RADI5 disk array is assumed. If RAID1 is used the data must be mirrored and G=2. If RAID5 is used, then G=5.

For RAID1 we have to consider that every data write is translated into two writes to different disks. Therefore, for RAID1 writes, the total number of IO/s that can be delivered by the disks must be divided by two. For the RAID1 reads it is only required to read the data from one disk. Thus, the number of IO/s that can be delivered by the disks is the number of IO/s for the reads. The ratio of reads R_p is also a factor that determines the disk array throughput (χ_{DA}) in IO/s. Thus, the equation to estimate the RAID1 disk array throughput is:

$$\chi_{DA}^{R1}(N) = R_p \chi_d(N) + (1 - R_p)\frac{\chi_d(N)}{2} \tag{3.13}$$

The reward r_0 of the optimal state S_0 for a RAID1 disk array is therefore:

$$r_0^{R1} = \chi_{DA}^{R1}(N) \tag{3.14}$$

For RAID5 we have to consider the kind of writes used for the analysis. In our case we used the typical small 4KiB accesses that an Exchange 2003 Server performs. The RAID5 level suffers from what is known as the “read-modify-writes” [Patterson 88a]. Every write is translated into two reads and two writes. Therefore, for RAID5 writes, the total number of IO/s that can be delivered by the disks must be divided by four. For the RAID5 reads it is only required to read the data from one disk. Thus, the number of IO/s that can be delivered by the disks is the number of IO/s for the reads. Again, the ratio of reads R_p is also a factor that determines the disk array throughput (χ_{DA}) in IO/s. Thus, the equation to estimate the RAID5 disk array throughput is:

$$\chi_{DA}^{R5}(N) = R_p \chi_d(N) + (1 - R_p)\frac{\chi_d(N)}{4} \tag{3.15}$$

The reward r_0 of the optimal state S_0 for a RAID5 disk array is therefore:

$$r_0^{R5} = \chi_{DA}^{R5}(N) \tag{3.16}$$

The reward r_1 for S_1, the non-optimal state, can be estimated by two factors: 1) One disk failed so we now have the throughput of *N-1* disks. 2) The disk array is also copying the data that was stored on the failed disk on other disk besides servicing user requests. Besides estimating the throughput for the case of *N-1* disks we need to add a factor that will drop the throughput a little more. We introduced a factor, R_f , with a value from [0,1]. This factor was the same for RAID1 and RAID5. For example, if the drop in performance caused by the reconstruction of the data redundancy is 5%, we assign $Rf = 0.05$. If more accuracy is needed, we introduce two factors, one for RAID1 and one for RAID5. So, the reward estimated for r_1 is:

$$r_1^{R1} = (1 - Rf)\chi_{DA}^{R1}(N-1) \tag{3.17}$$

$$r_1^{R5} = (1 - Rf)\chi_{DA}^{R5}(N-1) \tag{3.18}$$

Finally, the reward for $r_2 = 0$, since the disk array is the failed state.

The transient performability (TP) was defined in section 2.2.4. The TPM (3.7) gives us the probability of each state and with that we can estimate the performability of the disk array for every n_{th} iteration of Δt time by using:

$$TP^{R1}(n\Delta t) = P_{S0}(n\Delta t)r_0^{R1} + P_{S1}(n\Delta t)r_1^{R1} \tag{3.19}$$

$$TP^{R5}(n\Delta t) = P_{S0}(n\Delta t)r_0^{R5} + P_{S1}(n\Delta t)r_1^{R5} \tag{3.20}$$

where (3.19) and (3.20) are used to estimate the disk array performability in IO/s

Table 3.1: User profiles and corresponding usage patterns

User Type	Database Volume IO/s	Send/Receive per day	Mailbox Size
Light	.5	20 sent/50 received	50 MB
Average	.75	30 sent/75 received	100 MB
Heavy	1.0	40 sent/100 received	200 MB
Large	1.5	60 sent/150 received	500 MB

Now we need to come up with a way to estimate the performability of the mail server in number of users based on the performability in IO/s. We base the analysis on some of the recommendations for the configuration of an Exchange Server 2003 [Microsoft 07b].

The formula to estimate the performability in mailboxes, i.e., users the mail server can support is based on three factors: 1) user profiles shown in Table 3.1; 2) the formula (3.21) for the IO/s needed to support a number of mailboxes depending on the user type [Microsoft 07b]:

$$IO/s = (Number\_of\_Mailboxes) \times (UType) \tag{3.21}$$

and 3) the fact that 90% of the IO/s are user interaction and the other 10% go to the logs maintained by the mail server. The formulas are:

$$PM^{R1}(n\Delta t) = \frac{0.9(TP^{R1})}{UT_{Type}} \tag{3.22}$$

$$PM^{R5}(n\Delta t) = \frac{0.9(TP^{R5})}{UT_{Type}} \tag{3.23}$$

where $UType = (Light, Average, Heavy, Large)$. PM^{R1} and PM^{R5} and the performability in mailboxes for a R1 and R5 mail server.

3.3 Results of the Fuzzy Performability Analysis of the E-Mail Server

The intention of applying the fuzzy arithmetic to the performability analysis is to deal methodically with uncertainty. For the purpose of this example the authors decided to use a λ=1/10000 failure/hrs. Some of the parameters do not have a crisp value but a fuzzy value expressed in discretized form [Hanss 10a]. The discretized representation of fuzzy numbers used to deal with the Markov Chain model of performability can be expressed as fuzzy sets with five tuples $(x_i,\mu(x_i))$ where x_i is the value of the number and $\mu(x_i)$ is the corresponding membership value of x_i.

$$\tilde{P}^* = [(x_1,0),(x_2,0.5),(x_3,1),(x_4,0.5),(x_5,0)] \tag{3.24}$$

The fuzzy parameters for the this analysis are shown in a more concise form, where the $\mu(x_i)$ is omitted for brevity:

$$\tilde{p} = [x_1, x_2, x_3, x_4, x_5] \tag{3.25}$$

The parameters for this analysis were the following:

The life span of the mail server is 43,800 hours (5 years).

G for R1 = 2, Number of disks for a R1 group

G for R5 = 5, Number of disks for a R5 group

N = 200, Total number of disks

λ=[0.3x10^4, 0.5x10^4, 1x10^4, 2x10^4, 3x10^4] Failure rate

μ=[1/24, 1/16, 1/8, 1/4, 3/8] Repair Rate

R_p = [0.55, 0.6, 0.65, 0.7, 0.75], Percentage of Reads

R_t = [0.002,0.002,0.002,0.002,0.002] , Time for a rotation

S_t = [0.0038,0.0039,0.004,0.0041,0.0042], Time for a seek

B_s = [4096,4096,4096,4096,4096], Block size

χ_b = [2x10^8, 2x10^8, 2x10^8, 2x10^8, 2x10^8], Transfer rate

R_f = [0.03,0.04,0.05,0.06,0.07], Rebuild impact on reward

The resulting performability estimation is a fuzzy number with five values. For every iteration of the time t given by $t=n\Delta t$ (3.8), a fuzzy number representing a transient performability result is generated. The algorithm used to estimate the reliability and transient performability is presented in Table 3.2.

Table 3.2: Algorithm to compute the performability of disk array

I: Total number of iterations

P: Transition Probability Matrix

Δt: Time delta, e.g., 1 hour

t: Time elapsed at iteration i with a Δt

$PS(t)$: Vector with state probabilities at time t

$P_{Sk}(t)$: Probability of being in state k at time t

$R(t)$: Reliability of at time t

$TP(t)$: Transient Performability at time t

For i=1 to I do:

{

$$P^i = P^{i-1}P$$

Matrix P is normalized

$$t = i\Delta t$$

$$PS(t) = PS(0)P^i$$

$$R(t) = P_{S1}(t) + P_{S2}(t)$$

$$TP(t) = P_{S1}(t)r_{S1} + P_{S2}(t)r_{S2}$$

}

The discretized representation of the fuzzy number $\tilde{P}^*$ as shown in (3.25) is shown in graphical format in Fig. 3.2. The number $\tilde{P}^*$ is a triangular fuzzy number. The five x_i values are shown with their respective membership value $\mu(x_i)$ forming the five tuples that were obtained in each iteration of the algorithm presented in Table 3.2.

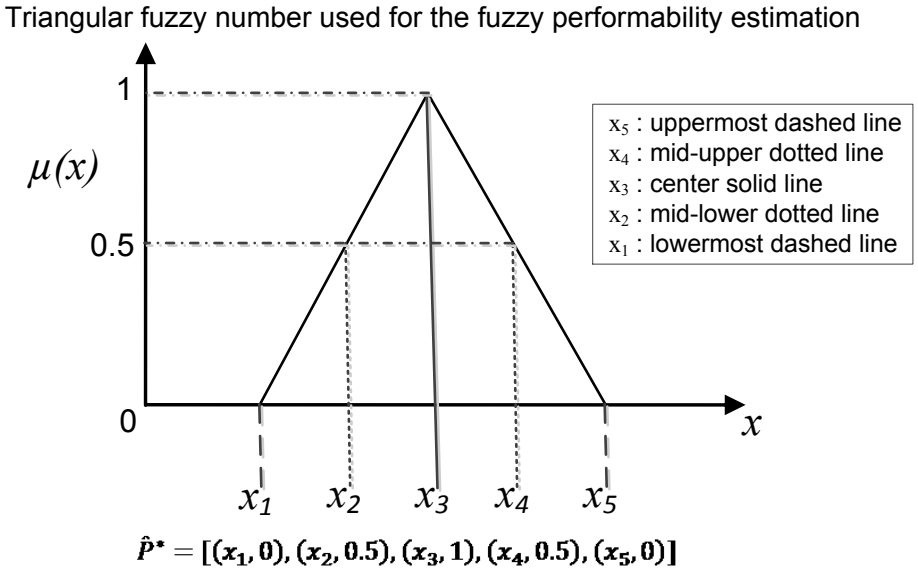

Fig. 3.2: Fuzzy number used for fuzzy performability estimation

The continuous line is the central value of the fuzzy result, x_3. The lower, x_2, and upper, x_4, dotted lines are values in between the central and both boundaries. The dashed lower line, x_1, is the lowest boundary of the fuzzy result. The dashed upper line, x_5, is the highest boundary of the fuzzy triangular result.

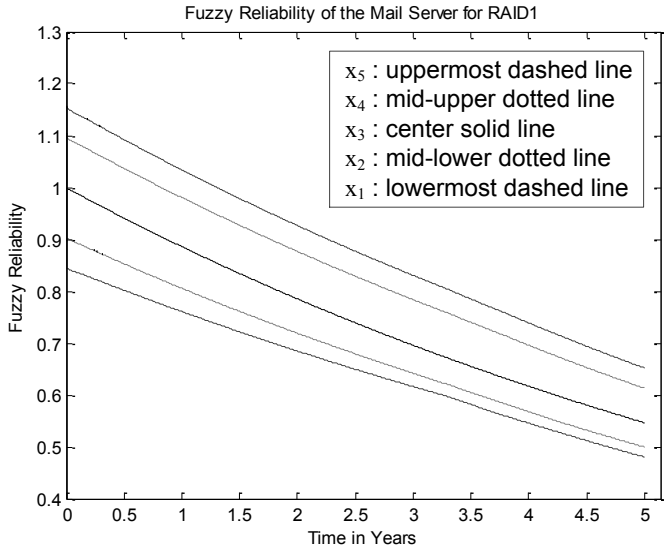

Fig. 3.3: Family of curves for fuzzy reliability RAID1

Fig. 3.3 shows the fuzzy RAID1 reliability of the mail server. It can be seen that there is a linear drop from 1 to 0.5 after 5 years of use. This is an indication that the mail server

most likely will not have any problems at the beginning of its life. At the end of its life there should be some provisions in case of failure.

Fig. 3.4 shows the fuzzy RAID5 reliability of the mail server. It can be seen that there is a linear drop from 1 to 0.1 after 5 years of use. This is an indication that the mail server most likely will fail as it gets closer to the end of its life. Here it is clear that provisions must be made to counter this. For example, a backup server should be considered or budgeted within the next 5 years in case the "main" mail server fails.

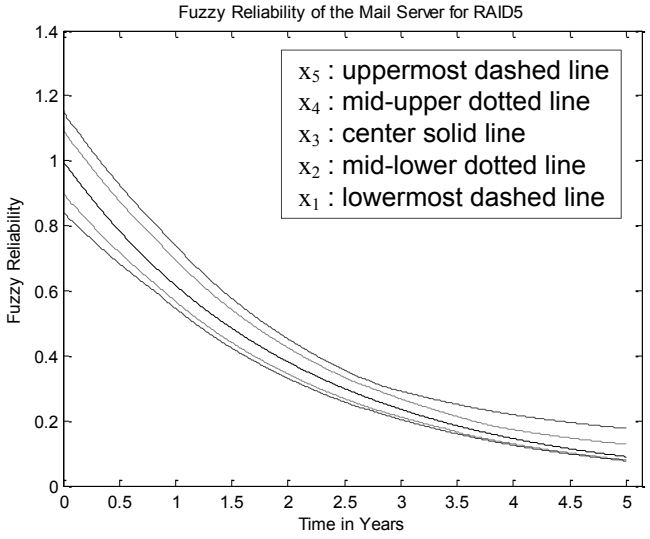

Fig. 3.4: Family of curves for fuzzy reliability RAID5

Fig. 3.5 shows the fuzzy RAID1 performability of the mail server. It can be seen that the IO/s range from around 40,000 to 20,000 at the beginning of the life of the mail server. The performability analysis tells us that after five years we can have throughputs in the order of 10,000IO/s to 25,000IO/s considering the reliability of the server. Depending on what level of service is expected in the next five years, plans should be made to adjust the amount of service the mail server will provide.

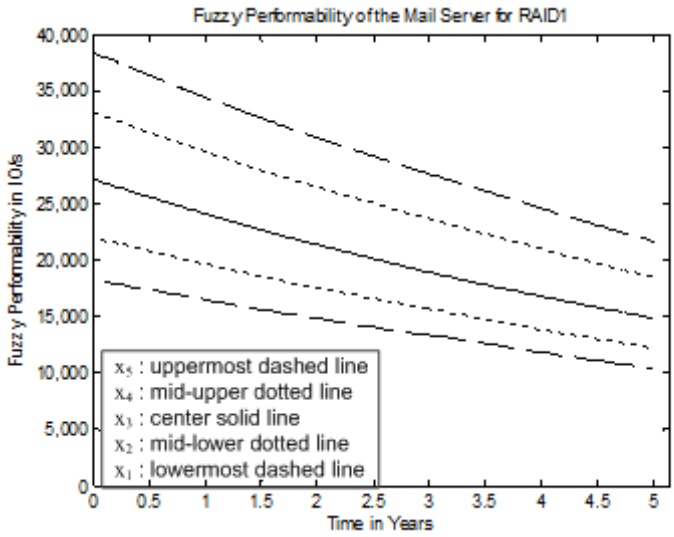

Fig. 3.5: Family of curves for fuzzy performability RAID1 in IO/s

Fig. 3.6 shows the fuzzy RAID5 performability of the mail server. It can be seen that the IO/s range from around 35,000 to almost 15,000 at the beginning of the life. The performability analysis tells us that after five years we can have no throughput. Here is very clear that if backup plans should be put in place to counter this future problem.

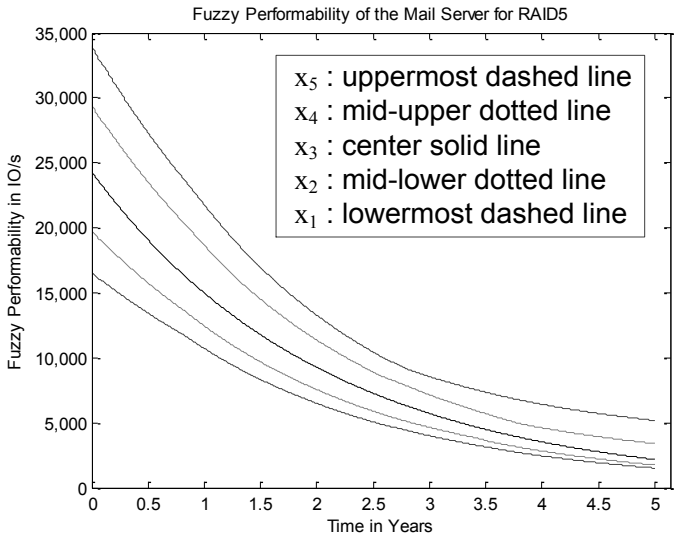

Fig. 3.6: Family of curves for fuzzy performability RAID5 in IO/s

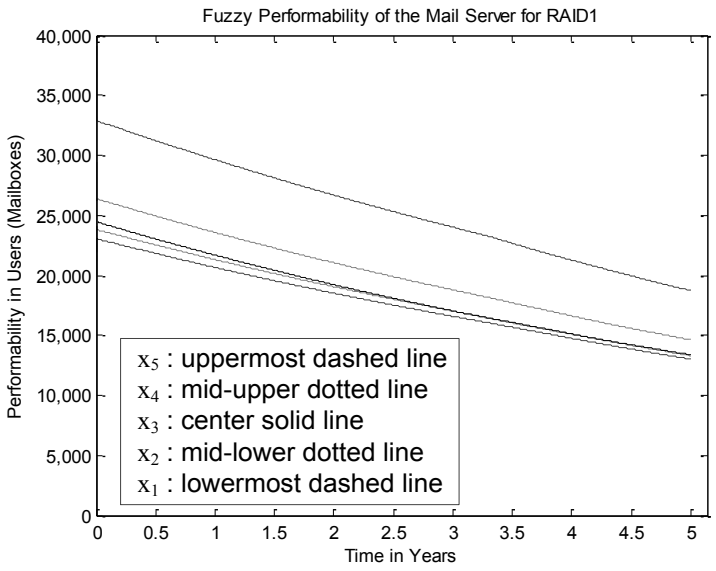

Fig. 3.7: Family of curves for fuzzy performability in Users (mailboxes) R1

Figures 3.7 and 3.8 show the performability in number of users over the life of the RAID1 mail server. This measure can serve to plan for the amount of service the system can yield. As we can see, at the beginning of the life of the mail server it can serve up to 30,000 light users or around 23,000 of the heavy users. If we want to keep this number of users constant we need to plan for the performability over the entire life of the product. In real life, figures 3.7 and 3.8 can be used to make the decision to use either RAID1 or RAID5 very easy based on the amount of service a business wants to provide.

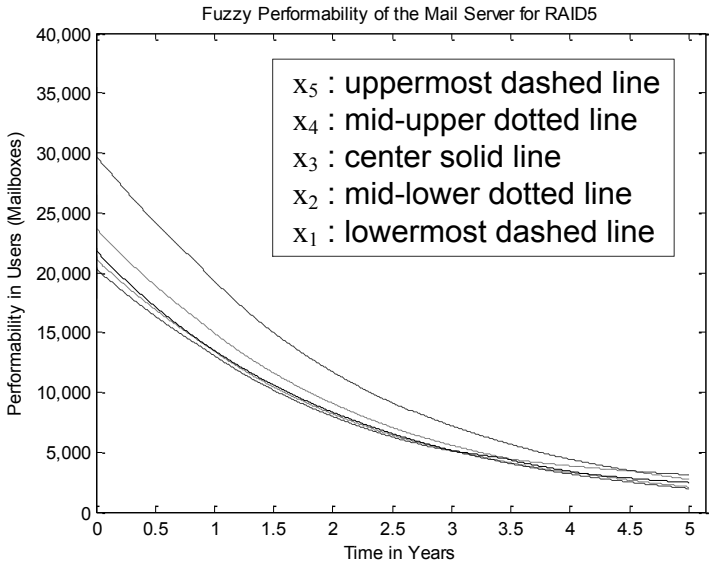

Fig. 3.8: Family of curves for fuzzy performability in Users (mailboxes) R5

3.4 CONCLUSIONS

It has been shown how performability analysis can be a tool for the analysis of the future capacity of service a computer system, and by extension, a service that a business can provide. In this section, it has been shown how performability based on specific fuzzy arithmetic approach can be a tool for planning the future in a way that allows a business to keep the quality of service promised to customers.

This section has also presented a numerical method in the form of an algorithm that can be used to estimate the reliability and performability of a Markov Model. The algorithm presented in this section can be a starting point for the estimation of the reliability and performability. But this algorithm is extensible because it can be adapted to particular conditions, e.g., a RAID level like RAID6 that supports double disk failure, or a function can be introduced to change the reward of the states, $r_{Sk}(t)$, and make the reward time-dependent instead of fixed.

Using the fuzzy arithmetic approach, all assets of the model presented were taken as they were – uncertain. By employing fuzzy arithmetic, aggregated inherent uncertainties of such a RAID system were modeled in one run. Extreme system performability behaviors illustrated by boundary curves paint an immediate picture of what are the worst and best case scenarios under given system parameter uncertainties. The approach of performability modeling based on a numerical method and using fuzzy arithmetic therefore provides a powerful tool for the effective design and business planning.

CHAPTER 4: FUZZY CONTROL OF SPARING FOR DISK ARRAYS

The analysis and modeling of the disk arrays under failure has been studied before [Muntz 90a], [Menon 93a], [Navarro 07b], [Navarro 07c]. But schemes of control of sparing have not been proposed in the literature before. This chapter presents two control schemes for the sparing data protection policy. The first is based on fuzzy logic and the second on a neural-fuzzy approach. Both schemes achieve a faster sparing than the traditional empty/no-empty control model, but without impacting the latency of user request.

Section 4.1 shows the fundamental model used in the fuzzy control scheme proposed in this chapter. The first model shown is the Queuing System with Vacations (QSV). The second model is of a disk array based on the QSV with the addition of the modeling of disks using disk-performance related measures. Also, models for the RAID1 and RAID5 rebuild processes are presented.

Section 4.2 presents a fuzzy-logic controller that uses three input parameters: 1) latency of user requests, 2) queue length and 3) time of sparing, to make the decision whether to allow user requests to proceed or continue with the sparing. This controller makes use of control of queues as proposed in [Phillips 99a] and [Zhang 05a].

Section 4.3 presents a neural-fuzzy controller (NFC) that uses three input parameters: 1) latency of user requests, 2) fraction of data spared and 3) time of sparing, to make the decision whether to allow user requests to be carried out or continue with the sparing.

Both sections, 4.2 and 4.3, compare their respective controllers against the traditional Queuing System with Vacations (QSV) model, or also referred to as empty/no-empty control model, where the sparing process only takes place when the queue is empty, or, in other words, when there are no users requests.

4.1 FUNDAMENTAL MODELS

4.1.1QUEUING SYSTEM WITH VACATIONS (QSV)

A queuing system in which the server may be disconnected (turned off) or removed is said to be a queuing system with vacations [Medhi 03a]. Fig. 4.1 illustrates the concept of queuing systems with vacations (QSV).

The requests arrive at a rate λ to the queuing system. The requests are processed at a rate μ. When the queue is empty, the server is idle. Then the server can turn itself off and execute some background process (go on vacation). After some time the server returns from executing the background process and rechecks the queue. If the queue is not empty, then the server turns itself on and serves the requests that arrived during the vacation of the server. But if the queue is still empty, the server keeps itself off and goes on vacation (execute the background process) again. This is referred to in this document as the empty/no-empty approach to control of the QSV.

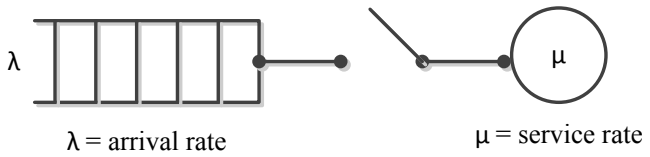

Fig. 4.1: Queuing System with Vacations (QSV)

4.1.2 Disk Array Queuing Model

The complete model of the disk array is based on a central server model with the addition of the queuing system with vacations (QSV). One of the advantages of fuzzy logic is the possibility of easily modeling and controlling systems in which mathematical models can be hard to derive. The problem of finding optimal policies for networks of queues is not trivial. Some queue optimization problems are probably intractable [Papadimitriou 94a]. Fig. 4.2 shows the model used for the disk array controller and the disks. This model combines the QSV with the queuing network formed by the disk array controller and the disks. The user requests to the queuing system arrive at a rate λ. The disk array controller then processes requests at a μ service rate.

The first approximation to a latency for the user requests can be obtained by saying the user request latency (response time) is the sum of the latency of the disk array controller, rt_{dac}, and the latency of the disk (for reads) or disks (for writes), rt_{disk}:

$$r_t = rt_{dac} + rt_{disk} \tag{4.1}$$

Estimating the latency of the disk array controller is not easy since it depends on two main factors: 1) the performance of the electronic components, e.g., CPU, memory; and 2) the software logic programmed in the disk array. This makes the modeling and estimation of the disk array controller latency, rt_{dac}, hard to obtain. This section shows that even with the lack of an exhaustive and detailed mathematical model, fuzzy logic can be applicable to the control of the sparing process. Also, because of the complexity of the model, this section uses simulation to show the improvements made by the fuzzy controller.

The model used in the simulation is based on the ST373454FC Seagate disk [Seagate 05a]. The service time, T_d, of a disk request depends on three factors: 1) rotational latency, t_{rot}, 2) seek time, t_{seek}, and 3) transfer time, t_{xfer}:

$$T_d = t_{rot} + t_{seek} + t_{xfer} \tag{4.2}$$

The disk positioning time is defined this way:

$$DPT = t_{rot} + t_{seek} \tag{4.3}$$

The disks will be modeled using the following equation:

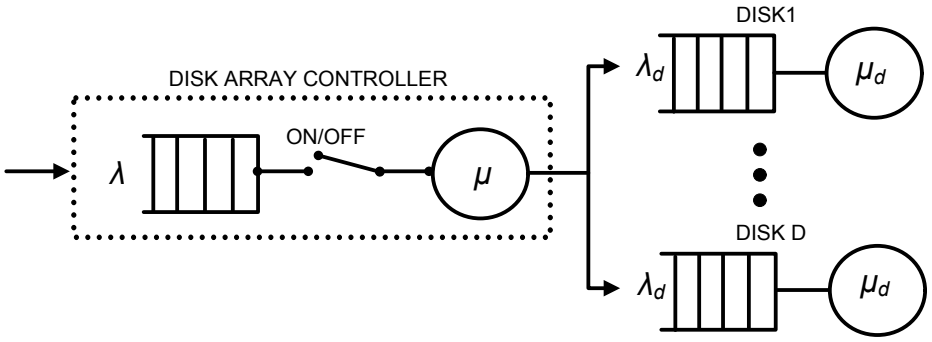

Fig. 4.2: Queuing system of controller and disks.

$$T_d = DPT + t_{xfer} \tag{4.4}$$

The disk service times are difficult to estimate since some factors, like disk specifications, disk caching and scheduling policy are hard to determine [Varki 03a]. The data

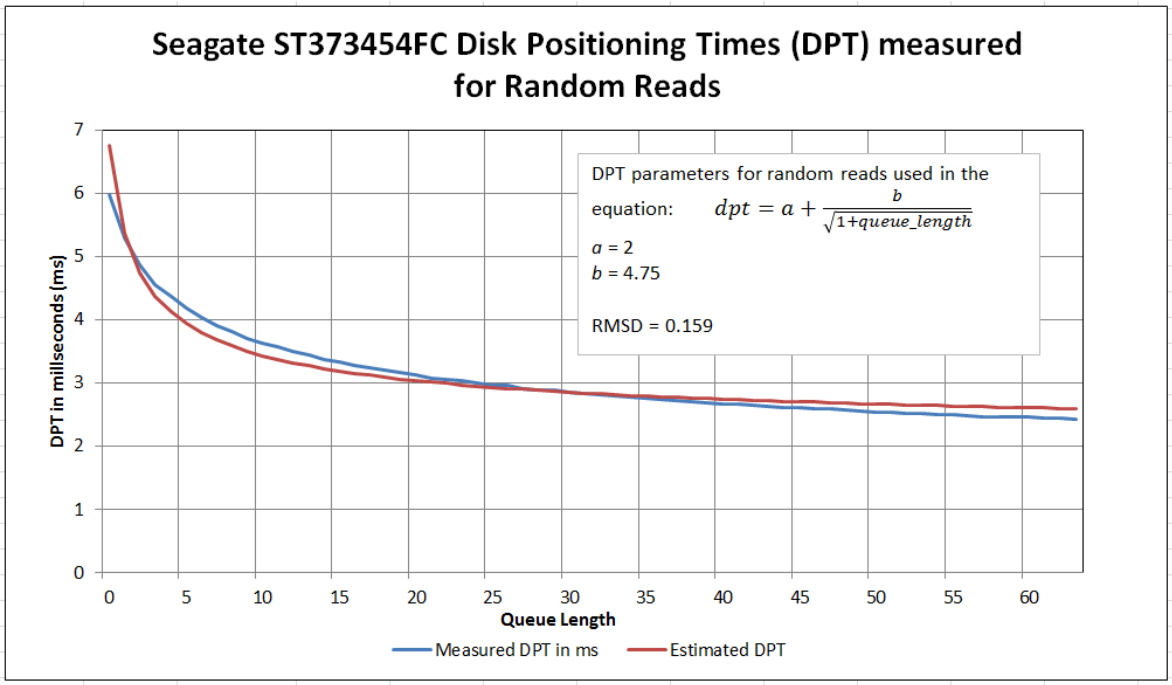

Fig. 4.3: Disk Position Times measured for Random Reads

used for this simulation came from measurements made on the ST373454FC Seagate disk. For random workloads the disk positioning time can be modeled by this equation in [Varki 03a]:

$$DPT = a + \frac{b}{\sqrt{1 + disk\_queue}} \tag{4.5}$$

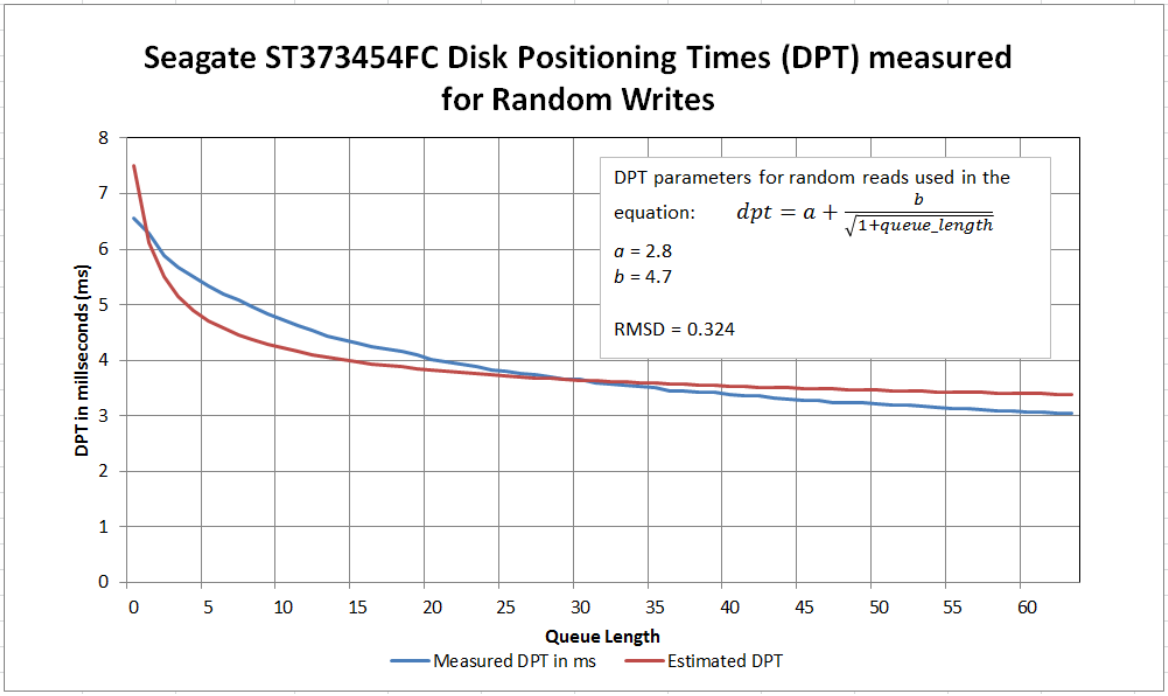

Fig. 4.4: Disk Position Times measured

Measurements were obtained from a ST37345FC disk to determine the parameters *a* and *b* for (4.5) for both random reads and writes. Fig. 4.3 shows the parameters were obtained for random reads, $a = 2$ and $b = 4.75$. The root-mean-square deviation (RMSD) between the measured and the estimated disk positioning time (DPT) was RMSD = 0.159.

Fig. 4.4 shows the parameters that were obtained for random writes, $a = 2$ and $b = 4.75$. The root-mean-square deviation (RMSD) between the measured and the estimated DPT was RMSD = 0.324.

The measured transfer time for random 4KiB transfers was t_{xfer} = 0.06ms and for random 128KiB transfers the t_{xfer} = 1.96ms. The simulations used later in this chapter made use of the parameters estimated for the DPT for random reads and writes as well as the txfer for 4KiB and 128KiB transfers.

4.1.3 RAID1 REBUILD MODEL

In this section, the new approach for sparing will be presented on the analysis of a RAID1 system. The RAID1 system can be in one of three modes: 1) *optimal*, when all the disks are working; 2) *degraded*, when one disk fails; 3) *failed*, when one pair of disks with the same data fail so there is no way to recover the data.

The RAID1 system consists of D disks, where D is an even number. The mirroring of the disks is by pairs of the d_{th} with d_{th}+1 disk, where d=1,3,5,…,D-1. The capacity of the disk is referred to as C_d. The disks are divided up in N_b number of disk blocks of size S_b. The number of N_b blocks per disk depends on the storage capacity of the disk C_d:

$$N_b = C_d / S_b \tag{4.6}$$

The disk block is the atomic unit of storage for the RAID1 system. When newly arrived data has to be stored on a disk, a new disk block is allocated. For this section, a block size S_b=128KiB will be used. This is the default size for the HP StorageWorks 1000/1500 MSA [HP 06a]. Each disk block is referred to as B_i, where i=1,2,3,…,N_b.

Fig. 4.5 shows the data layout of the RAID1 system in optimal mode. Each block B_i has a corresponding mirror on the other disk indicated by B'_i. For example, disk 1 (disk d_{th}) and 2 (disk d_{th}+1) form a pair of data and its mirror. The spare disk is in standby mode and no data blocks have been allocated on it.

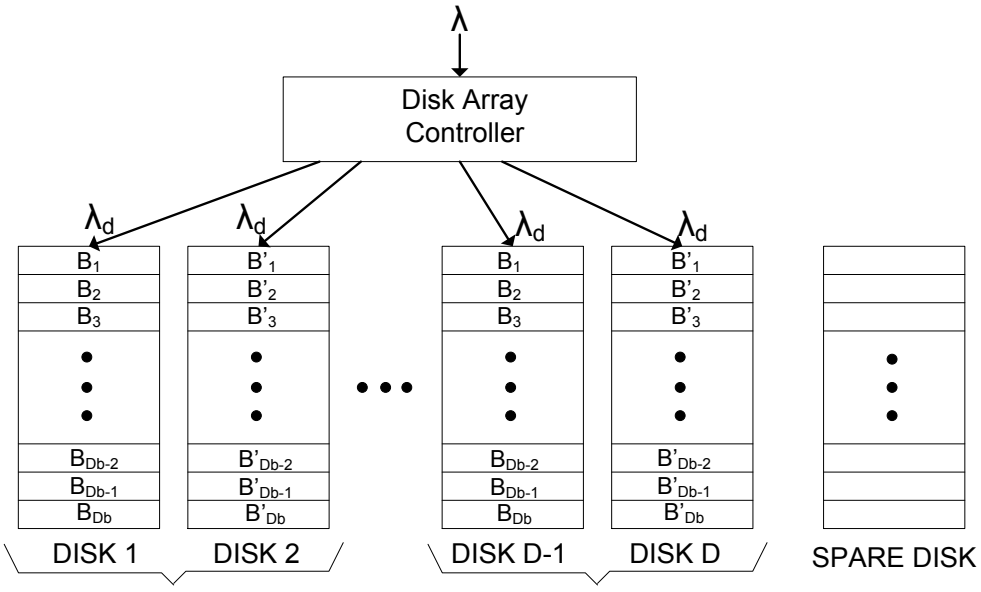

Fig. 4.5: RAID1 disk array data layout

A workload with arrival rate λ is applied to the RAID1 system controller by the users. The throughput χ in IOs requests per second (IO/s), can be specified by:

$$\chi = {}^{1}/_{\lambda} \tag{4.7}$$

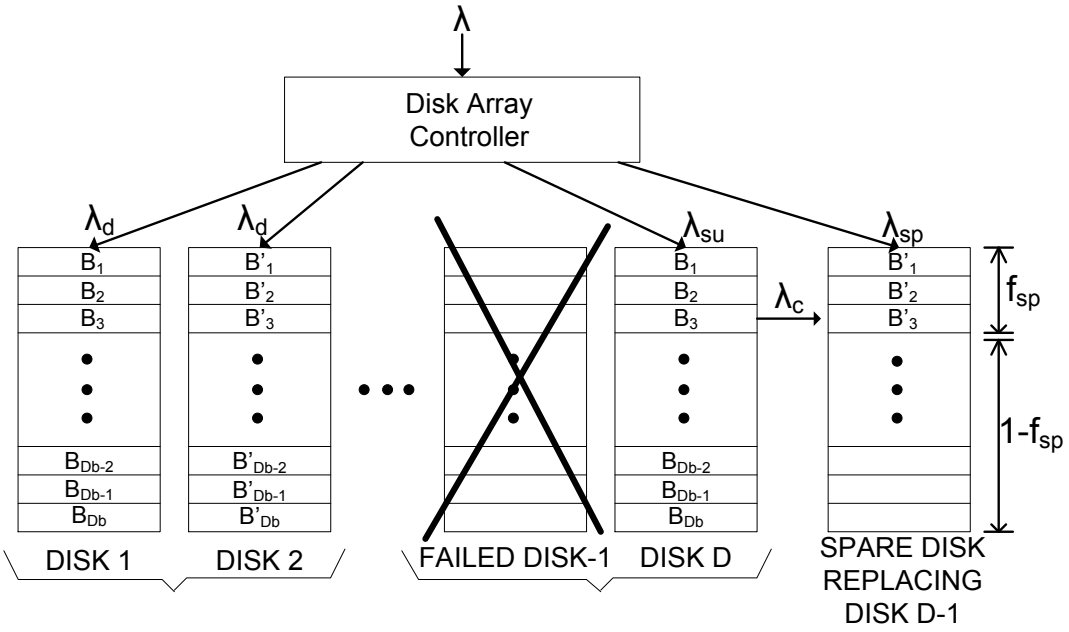

Fig. 4.6: Sparing process to replace failed disk D-1

The throughput is distributed across the disks. A balanced workload across the disks is considered in this section.

When a disk fails, the sparing process is started and the copy of the data on the surviving disk to the spare disk is performed on a block by block basis. Fig. 4.6 shows an example of a failed disk; in this case, disk *D*-1 failed and the spare disk is now in process of replacing disk *D*-1. The sparing process copies the disk blocks B_i from disk *D* to the spare disk that is now the new disk *D*-1.

The fraction of the N_b blocks copied is f_{sp}. If B_c is the number of disk blocks already copied to the spare disk, then fraction of the N_b blocks is:

$$f_{sp} = B_c / N_b \tag{4.8}$$

When the RAID1 system is sparing, the combined throughput of the disks changes from that of the optimal state, since now we have two different conditions: 1) the surviving disk is now serving its share of user requests and reading its disk blocks; 2) the spare disk is writing its disk blocks and serving read requests for the data already copied and new writes to the data on it. The other D-2 disks are serving requests as they would normally do. This procedure to reconstruct the RAID1 redundancy is known as the *baseline copy procedure* [Muntz 90a]. There are other ways to proceed with the reconstruction of the data that are also mentioned in [Muntz 90a]

4.1.4 RAID5 REBUILD MODEL

For this section, a RAID5 disk array will be used for the analysis. The disk array can be in one of three modes: 1) optimal, when all the disks are working; 2) degraded, when one disk fails; 3) failed, when one pair of disks with the same data fail so there is no way to recover the data. For this section, the disk array will be considered to be in the degraded state.

The disk array consists of D disks. The D disks are divided up in RAID5 disk groups of G disks [Patterson 88a]. Fig. 4.7 shows the data layout of the disk array in optimal mode. For the example shown in Fig. 4.7, G=5. The number of RAID5 groups is N_g:

$$N_g = D / G \tag{4.9}$$

The data on the disks is divided up in data blocks. The spare disk is in standby mode and no data blocks have been allocated on it. In each disk group, one data block disk stores the parity of the data blocks of the other G-1 disks. Each data block is referred to as $B_{i,j}$, where i=1,2,3,…,N_b and j=1,2,..,G. The group of data blocks $B_{i,j}$ where i is constant and j goes from j=1,2,…,G is referred to as a *stripe*. The number of N_b blocks per disk depends on the capacity of the disk, C_d as shown (4.6). The size of each data block $B_{i,j}$ is S_b. A practical example of the size of the disks blocks is taken from the Storage Works 1000/1500 MSA [HP 06a]. This disk array uses a default block size of 128KiB. Thus, for this section, a block size S_b=128KiB will be used.

The user reads and writes are executed differently depending on which disk is the data to be accessed. The possible cases considered for this example are:

1) *Optimal reads*. If the disk on which the data is located is a working disk, a read requested to the RAID5 disk array controller translates into one read on that particular disk.

2) *Degraded reads*. If the sparing process has already regenerated the data and written it on the spare disk, then it is possible to read the data from the spare disk directly, in this case this is an optimal read as in the previous case. But, if the disk on which the data is located is the failed disk, and the data has not been regenerated on the spare disks, then we have a degraded read. A degraded read requested to the RAID5 disk array controller translates into *G*-1 reads. This is because the data on the failed disk cannot be read and has to be reconstructed by reading the data (and the parity) on the other *G*-1 disks of the RAID5 disk group.

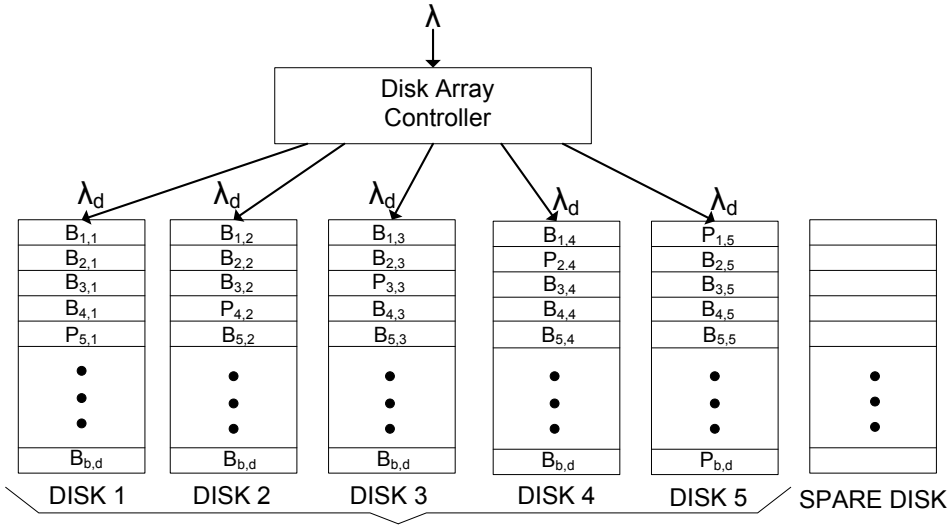

Fig. 4.7: RAID5 disk array data layout

3) *Optimal writes*. The example used in this section was of a disk array under small block (4KiB) randomly distributed writes. These are translated on the disks into the four accesses: two accesses to read the old data and the old parity, and two more accesses to write the new data and the new parity. This kind of writes is known as the read-modify-write (RMW) [Patterson 88a]. For the rest of this section, an optimal write is a RMW.

4) *Degraded write with a failed parity disk*. If the disk on which the parity is located is the failed disk, then only the new data is written on the working disk where the old data was located. Later on, the sparing process will reconstruct the parity on the spare disk. Of course, there is also the possibility that the parity on the failed disk has already been reconstructed on

the spare disk. If this is the case, then instead of a degraded write, this write is executed as an optimal write.

5) *Degraded write with a failed data disk*. If the disk on which the data is located is the failed disk, then it is necessary to read the data on the other G-2 disks in the disk group. The data of the G-2 drives along with the new data to be written is used to compute a new parity. This new parity is then written on the working parity disk.

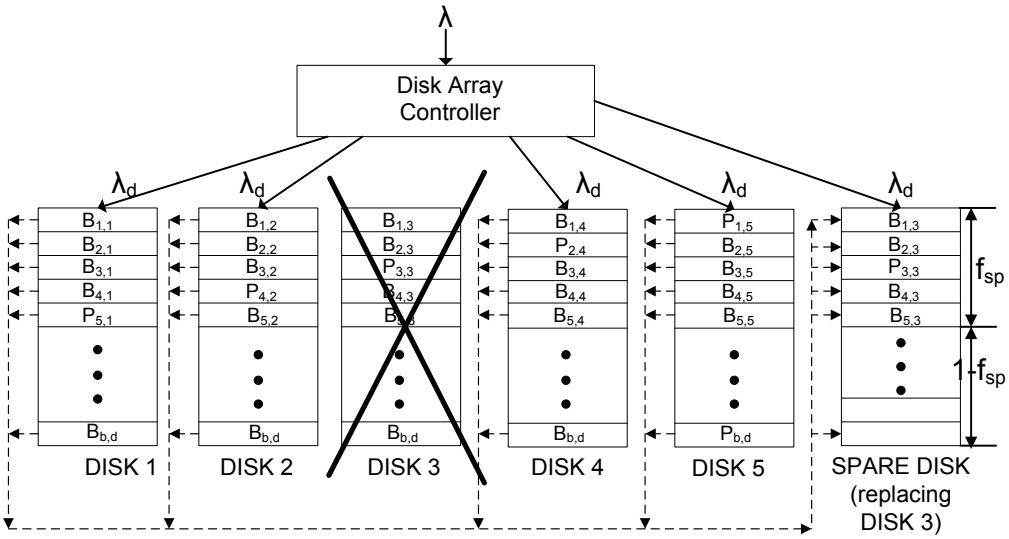

Fig. 4.8: RAID5 disk sparing process to replace failed disk

6) *Sparing write*. The reconstruction (sparing) of the data is performed this way: the data (and parity, depending on the stripe being reconstructed), of the other surviving G-1 disks is read. The data blocks $B_{s,j}$ where j=1,2,..,G-1 and s is the stripe being spared, are read. Then the G-1 data blocks are used to compute the $B_{s,G}$ block, which can be data or parity according to the rotating scheme of parity on the G disks. And then, the $B_{s,G}$ block is written on the spare disk. Fig. 4.8 shows a RAID5 disk group where one disk failed and the spare disk is in the sparing process.

In this model the fraction of the total N_b blocks copied is also f_{sp} as shown in (4.8). A workload with throughput λ is applied to the disk array controller by the users. The percentage of read requests in the workload applied to the disk array controller is represented by ρ and the percentage of write requests in the workload is represented by ω. Both percentages are related by:

$$\omega = 1 - \rho \tag{4.10}$$

The throughput of the user reads, λ_R, is:

$$\lambda_R = \rho\lambda \tag{4.11}$$

The throughput of the user writes, λ_{RMW}, is:

$$\lambda_{RMW} = \omega\lambda \tag{4.12}$$

One way to estimate the disk throughput is subdividing the throughput in two groups. One throughput, $\lambda_{OPT,}$ is made up by the D-G disks that make up the N_g-1 optimal groups. The other throughput, $\lambda_{DEG,}$, is composed of the G disks where the failed disk is located. We can say, then, that the total throughput on the disks, λ_D, is:

$$\lambda_D = \lambda_{OPT} + \lambda_{DEG} \tag{4.13}$$

For this section, a balanced workload is assumed. We know that for optimal reads, there is a one-to-one correspondence between the user reads and disk reads. For the read-modify-writes, two reads and two writes are performed for each one. With this, the throughput of the disks in the optimal groups is:

$$\lambda_{OPT} = \frac{(\lambda_R + 4\lambda_{RMW})}{D}(D - G) \tag{4.14}$$

The throughput of the disks in the degraded group has to consider the fact that as the sparing process progresses, the fraction of data spared, f_{sp}, goes to one and the accesses become optimal as more and more data has its redundancy reconstructed. Also, besides the user workload, the sparing process adds more requests on the disks. First, the reads in the degraded disk group can be estimated by reasoning this way: this is a balanced workload, so

each drive get its λ_R/D share of reads. For the G-1 working disks in the group, this translates into one disk read. But if the read is directed to the failed disk and the data has not been reconstructed on the spare drive, this translates into reads from the other G-1 disks. So, we can say that the throughput in the degraded disk group caused by the user reads, λ_{DEG_READ}, is:

$$\lambda_{DEG\_READ} = \frac{2(G-1)\lambda_R(1-f_{sp}) + \lambda_R G f_{sp}}{D} \tag{4.15}$$

The writes in the degraded disk group can be estimated by reasoning this way: again, this is a balanced workload, so each drive get its λ_{RMW}/D share of reads. For the G-1 working disks in the group, this translates into two reads and two writes. But if the write is directed to the failed disk and the data (or parity) has not been reconstructed on the spare drive, this gives rise to one of two possibilities:

1) the parity of the RMW was on the failed disk. Since the parity block rotates, we know that $1/G$ of each disk is used to store parity, so the probability of this case, p_{par}, is:

$$p_{par} = 1/G \tag{4.16}$$

And the equation that estimates the throughput caused by the degraded writes with a failed parity disk, λ_{DEG_PAR}, is:

$$\lambda_{DEG\_PAR} = \frac{(1-f_{sp})p_{par}\lambda_{RMW}}{D} \tag{4.17}$$

2) the data of the RMW was on the failed disk The probability of this, p_{dat}, is:

$$p_{dat} = (G-1)/G \tag{4.18}$$

And the equation that estimates the throughput caused by degraded writes with a failed data disk, λ_{DEG_DATA}, is:

$$\lambda_{DEG\_DATA} = \frac{(1-f_{sp})(G-1)p_{dat}\lambda_{RMW}}{D} \tag{4.19}$$

So, we can say that the throughput in the degraded disk group caused by the user reads, λ_{DEG_WRITE}, is:

$$\lambda_{DEG\_WRITE} = \frac{(G-1+f_{sp})4\lambda_{RMW}}{D} + \lambda_{DEG\_DATA} + \lambda_{DEG\_PAR} \tag{4.20}$$

The throughput in the degraded disk group also includes the throughput of the sparing process. The sparing writes, as mentioned above, require G-1 reads and one write. These accesses are of size S_b, and the throughput, $\lambda_{SPARING}$, depends on the characteristics of the drive, the throughput imposed by the user and the algorithm used for sparing. Putting the λ_{DEG_READ}, λ_{DEG_WRITE}, and the $\lambda_{SPARING}$, we have:

$$\lambda_{DEG} = \lambda_{DEG\_READ} + \lambda_{DEG\_WRITE} + \lambda_{SPARING} \tag{4.21}$$

With (4.13) and (4.21) together is now easier to understand why the throughput of a disk array drops when a disk fails.

4.2 Fuzzy Control of the Sparing Process

The proposed solution to find the optimal policy that balances the time needed to complete the sparing and the latency of the user requests is by using a fuzzy controller. This solution will be more flexible than the traditional QSV model, where the sparing process only occurs when the queue is empty. In this proposed solution more parameters will be considered. The input parameters will be fuzzified so we can base the decisions on fuzzy values. The use of fuzzy values allowed us the use of a rule base with the logic to control the sparing process. In Fig. 4.9 we show a graphical model of the proposed solution.

The input parameters of the fuzzy controller are three: 1) The queue length of the controller, ql; 2) the latency of the disk array controller r_t; and 3) the time elapsed since a disk failed and the sparing process started, t_{sp}.

For this example the three parameters were normalized. The first parameter *ql*, is considered to make an improvement of the traditional QSV, empty/no-empty approach. The idea is to allow the sparing process to execute even if there are requests in the queue waiting to be served. The *ql* can be normalized by using Little's theorem [Zhang 05a]. The normalization of variables make it easier to map the crisp values of the variable to fuzzy values. For example, if we assume an average latency of RT_{avg} = 10ms for the users and an average throughput of λ_{avg} = 1,000 IO/s, then we can use Little's theorem and estimate the average queue length:

$$L = RT_{avg} * \lambda_{avg} = 0.010 * 1000 = 10 \tag{4.22}$$

The *ql* then can be considered to be 10 as an average. We considered 20 as the ql_{max} and the normalization of the *ql* was using this formula:

$$ql_n = \frac{ql}{ql_{\max}} \tag{4.23}$$

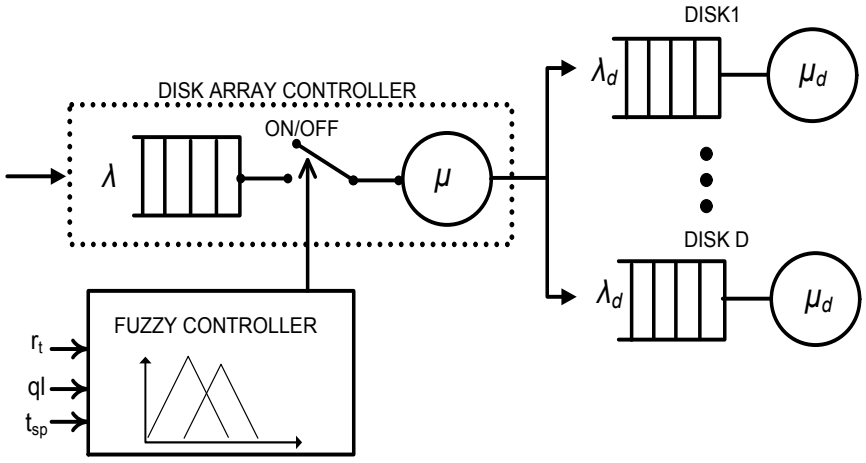

Fig. 4.9: Fuzzy controller of the QSV for sparing

The other two parameters are normalized by making two assumptions:

1) The latency, r_t, can be normalized if we consider that there are some upper limits to the latency the user applications can withstand without causing problems; such as high user latencies or timeouts of user applications. One example is with the Microsoft Exchange Servers. There are some latencies that are considered the maximum acceptable (50ms) [Microsoft 06a] and above those latencies there can be problems. For the simulation shown in this section, it was assumed that a delay of rt_{max}=50ms was the maximum that can be tolerated. The normalized response time (latency), rt_n, used by the fuzzy controller is then:

$$rt_n = \frac{r_t}{rt_{\max}} \tag{4.24}$$

2) The time elapsed in the sparing process since a disk failed, t_{sp}, is normalized also. The assumption made is that there is a maximum time acceptable for the user without the redundancy of the data restored. This is a reasonable assumption since the purpose of a disk array is to guarantee the redundancy of the data so there is no data loss when a disk fails. The maximum time allowed for a sparing to finish was assumed to be tsp_{max}=24 hours. With this assumption the normalized time elapsed in the sparing process, tsp_n is:

$$tsp_n = \frac{t_{sp}}{tsp_{\max}} \tag{4.25}$$

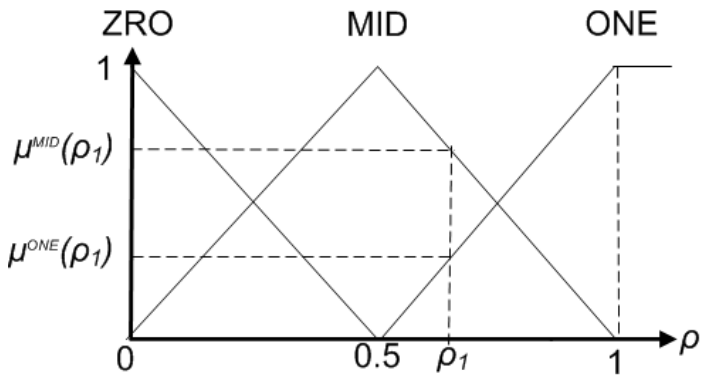

Fig. 4.10: Membership functions for the normalized values

With the three input parameters normalized, now the membership function can be defined. Three linguistic values were assigned three fuzzy descriptors, ZRO, MID, and ONE, which stand for "zero", "middle value" and "one". The fuzzification of the three fuzzy descriptors was performed via a triangular membership function for each descriptor. This technique is shown in [Zhang 05a]. Fig. 4.10 shows the triangular membership functions for all three input parameters. The graph shows also an example using a dummy variable ρ that can be replaced by any one of the three normalized parameters, ql_n, rt_n and tsp_n. The membership function $\mu^{ZRO}(\rho)$ for the fuzzy descriptor ZRO is:

$$\mu^{ZRO}(\rho) = \begin{cases} -2\rho + 1 & if\ \ 0 \le \rho \le 0.5 \\ 0 & if\ \ \rho > 0.5 \end{cases} \tag{4.26}$$

The membership function $\mu^{MID}(\rho)$ for the fuzzy descriptor MID is:

$$\mu^{MID}(\rho) = \begin{cases} 2\rho & if\ \ 0 \le \rho \le 0.5 \\ -2\rho + 2 & if\ \ 0.5 < \rho \le 1 \end{cases} \tag{4.27}$$

The membership function $\mu^{ONE}(\rho)$ for the fuzzy descriptor ONE is:

$$\mu^{ONE}(\rho) = \begin{cases} 0 & if\ \ \rho < 0.5 \\ 2\rho + 1 & if\ \ 0.5 \le \rho \le 1 \end{cases} \tag{4.28}$$

The fuzzy value of rt_n is defined as F_{rtn}:

$$F_{rtn} = \max\left[\, \mu^{ZRO}(rt_n),\ \mu^{MID}(rt_n), \mu^{ZRO}(rt_n) \,\right] \tag{4.29}$$

The fuzzy value of ql_n is defined as F_{qln}:

$$F_{qln} = \max\left[\, \mu^{ZRO}(ql_n),\ \mu^{MID}(ql_n), \mu^{ZRO}(ql_n) \,\right] \tag{4.30}$$

The fuzzy value of tsp_n is defined as F_{tspn}:

$$F_{tspn} = \max\left[\, \mu^{ZRO}(tsp_n),\ \mu^{MID}(tsp_n), \mu^{ZRO}(tsp_n) \,\right] \tag{4.31}$$

Now the next step is the specification of the rules for the rule base. The linguistic criteria can be summarized:

1) The latency of the disk array controller, r_t, should be kept as low as possible. This is one of the features that must be balanced during the sparing process. If the latency r_t, is low, we can proceed with the sparing.

2) The sparing process should be finished as soon as possible. This is the other feature that must be balanced. The closer we are to the maximum time allowed for a sparing process to finish, tsp_{max}, more priority should be given to the sparing process.

3) The lower the queue length is, the more we can spare since just few processes will be delayed.

With these linguistic criteria, the rule base can be built. The output of each rule is a binary value of YES, which means continue the sparing process, or NO, which means to hold off the sparing process. The complete rule base of the fuzzy control of sparing is in Table 4.1.

Table 4.1: Rule base of the fuzzy control of Sparing

Rules 1-9				Rules 10-18				Rules 19-27			
rt_n	ql_n	tsp_n	*out*	rt_n	ql_n	tsp_n	*out*	rt_n	ql_n	tsp_n	*out*
ZRO	ZRO	ZRO	YES	MID	ZRO	ZRO	YES	ONE	ZRO	ZRO	YES
ZRO	ZRO	MID	YES	MID	ZRO	MID	YES	ONE	ZRO	MID	YES
ZRO	ZRO	ONE	YES	MID	ZRO	ONE	YES	ONE	ZRO	ONE	YES
ZRO	MID	ZRO	YES	MID	MID	ZRO	YES	ONE	MID	ZRO	NO
ZRO	MID	MID	YES	MID	MID	MID	YES	ONE	MID	MID	NO
ZRO	MID	ONE	YES	MID	MID	ONE	YES	ONE	MID	ONE	YES
ZRO	ONE	ZRO	YES	MID	ONE	ZRO	YES	ONE	ONE	ZRO	NO
ZRO	ONE	ONE	YES	MID	ONE	MID	YES	ONE	ONE	MID	NO
ZRO	ONE	ONE	YES	MID	ONE	ONE	YES	ONE	ONE	ONE	YES

The output of the fuzzy controller is the decision to turn on/off the disk array controller to server user requests (on) or regenerate the redundancy (off). The deffuzification of the output is done by applying the rule and the result is a zero (NO) or a one (YES). We define the output set as the crisp set:

$$C_{out\_f} = \{\, \text{YES},\ \text{NO} \,\} \tag{4.32}$$

The rules are of the form:

$$if\ \boldsymbol{rt_n} \in F_{rtn}\ and\ \boldsymbol{ql_n} \in F_{qln}\ and\ \boldsymbol{tsp_n} \in F_{tspn}\ then\ \boldsymbol{out}\ \in C_{out\_f} \quad (4.33)$$

4.2.1 Simulation and Results

The model of the RAID5 system used for the simulation is based on the description in 4.1.4 with that addition of a central server and the QSV as shown in Fig. 4.9. The user requests to the queuing system arrive at a rate λ. The RAID controller (the server) processes requests at a μ service rate. A simulation of the queuing system in Fig. 4.2 was the approach used in this section to show the improvements made by the fuzzy controller. The disk parameters used for this simulation were presented at the end of section 4.1.2. The simulation was done using the CSIM19 simulation package [Mesquite 07a]. The workload applied was 75% reads (3:1 ratio), as typical for Exchange Server environments [Microsoft 06b]. A disk array with 80 ST373454FC disks was simulated using a RAID5 model as shown in section 4.14. The disk array controller $1/\mu$ used was 0.08ms with an exponential distribution. The throughputs applied for comparison were 1000, 2500, 5000, and 7500 IO/s. The throughputs were maintained constant during the entire duration of the simulation. The intention was to measure the variations in latency and the duration of the sparing process.

The graphs used for the comparison show on the horizontal axis the total time taken for the sparing process to complete, and on the vertical axis the latency measured for the user requests.

Fig. 4.11 shows the result for the 1000 IO/s throughput applied to the disk array. This result shows the fuzzy controller outperforming the traditional empty/no-empty controlled sparing. The graph shows the fuzzy-controlled sparing finishing in 1.5 hours after the disk failed; whereas the empty/no-empty sparing finished in 3.2 hours. For both cases, the latency was around 10ms. There is a great improvement in the reduction of the sparing time by half with no impact on the user request latency

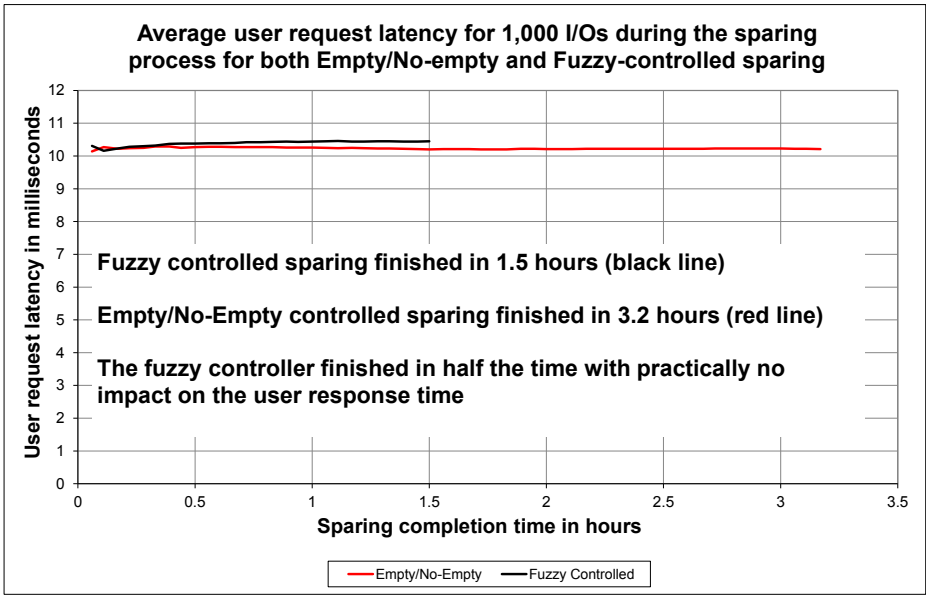

Fig. 4.11: User request latency comparison for 1,000 IO/s with fuzzy control

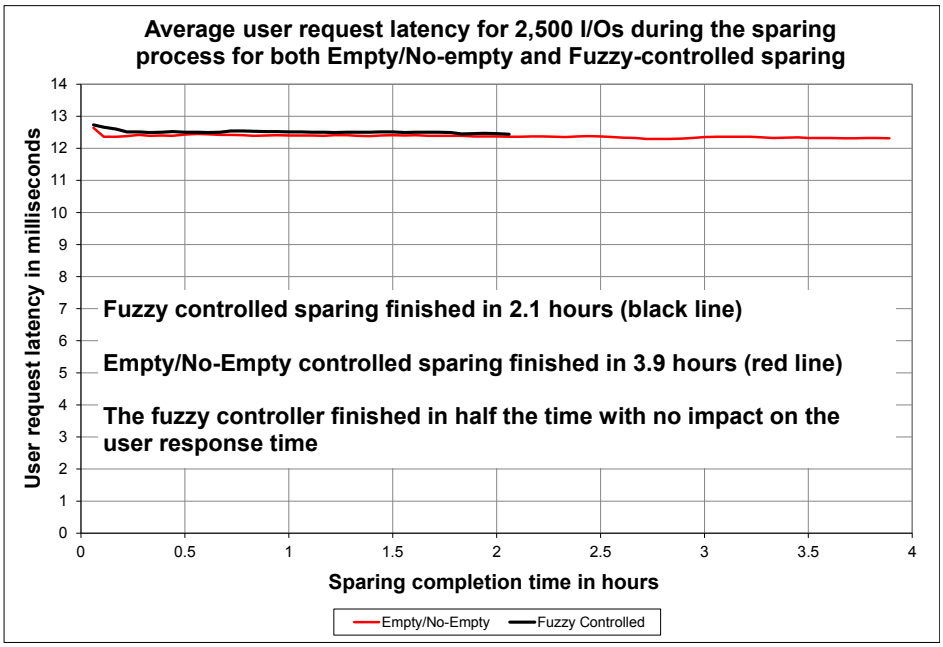

Fig. 4.12: User request latency comparison for 2,500 IO/s with fuzzy control

Fig. 4.12 shows the result for the 2,500 IO/s throughput applied to the disk array. This result shows the fuzzy controller outperforming the traditional empty/no-empty controlled sparing. The graph shows the fuzzy-controlled sparing finishing in 2.1 hours after the disk failed, whereas the empty/no-empty sparing finished in 3.9 hours. For both cases, the latency

was around 12.5ms. There is an improvement in the reduction of the sparing time by half with no impact on the user request latency.

Fig. 4.13 shows the result for the 5,000 IO/s throughput applied to the disk array. This result shows the fuzzy controller outperforming the traditional empty/no-empty controlled sparing. The graph shows the fuzzy-controlled sparing finishing in 3.1 hours after the disk failed; whereas the empty/no-empty sparing finished in 5.4 hours. For both cases, the latency was around 17.5ms. There is an improvement in the reduction of the sparing time by half with no impact on the user request latency.

Fig. 4.14 shows the result for the 7,500 IO/s throughput applied to the disk array. This result shows the fuzzy controller outperforming the traditional empty/no-empty controlled sparing again. The graph shows the fuzzy-controlled sparing finishing in 8.1 hours after the disk failed; whereas the empty/no-empty sparing finished in 4.6 hours. For both cases, the latency was around 24.5ms. There is an improvement in the reduction of the sparing time by half with no impact on the user request latency.

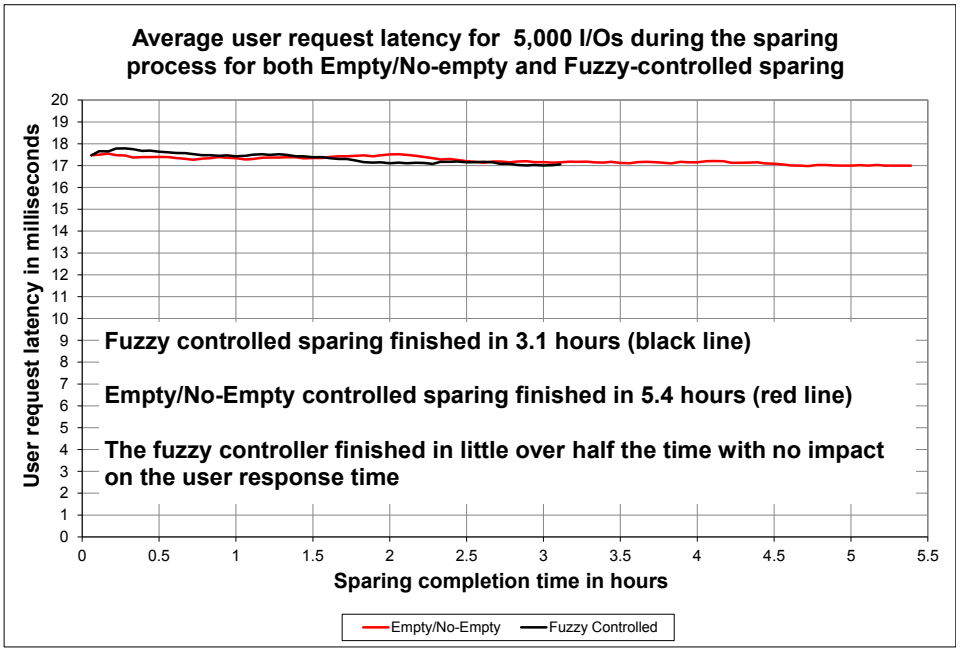

Fig. 4.13: User request latency comparison for 5,000 IO/s with fuzzy control

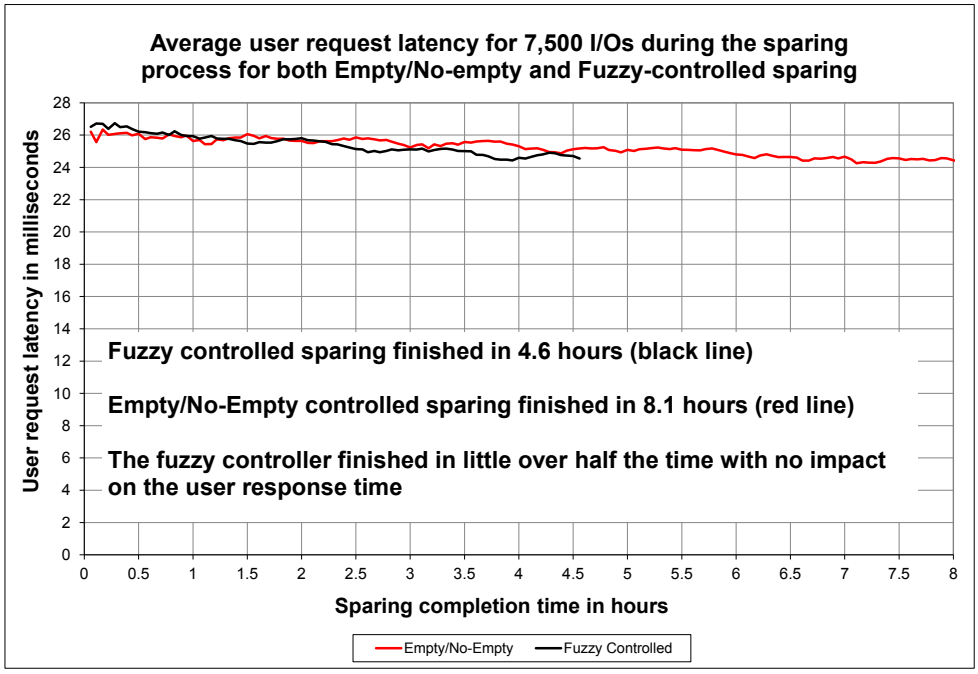

Fig. 4.14: User request latency comparison for 7,500 IO/s with fuzzy control

Table 4.2 with a comparison of the four results, shows an improvement of the sparing process by cutting the duration in half with no impact on the user request latency. This simulation shows the value of using fuzzy controlled logic for the improvement of the sparing process.

Table 4.2: Comparison of results of the fuzzy control of sparing

IO/s	Empty/No-empty controlled sparing		Fuzzy-controlled sparing	
	Sparing Duration (hrs)	Latency (ms)	Sparing Duration (hrs)	Latency (ms)
1,000	3.2	10	1.5	10
2,500	3.9	12.5	2.1	12.5
5,000	5.4	17.5	3.1	12.5
7,500	8.1	24.5	4.6	24.5

4.3 NEURAL-FUZZY ALGORITHM FOR SPARING IN RAID SYSTEMS

This neural-fuzzy sparing control scheme is based on the fuzzy sparing control scheme presented in section 4.2 but with two differences: 1) the rule base was implemented with an artificial neural network; and 2) the fraction of data that has been already spared from the surviving disks is used instead of the queue length. This neural-fuzzy sparing control scheme proposed to balance the time needed to complete the sparing and the latency of the user requests is composed of two neural nets with the following features: 1) the input parameters to the neural net controller are normalized so they are in the [0,1] range; 2) the input parameters are fuzzified using three membership functions, LOW, MED and HIG; 3) the fuzzification of the input parameters is made by the first neural net; 4) the second neural net implements the rule base and makes the decision whether to keep sparing or hold the sparing temporarily, based on the fuzzified parameters from the first neural net. In Fig. 4.15 we show a graphical model of the proposed solution.

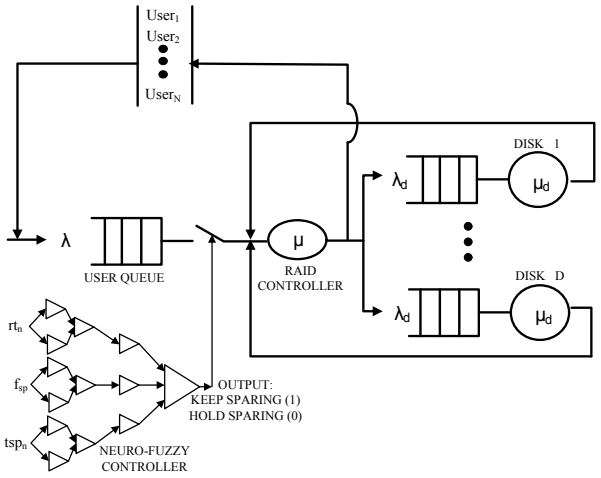

Fig. 4.15: Neural-Fuzzy controller of the QSV for sparing

The input parameters of the fuzzy controller are three: 1) The fraction of data already spared from the surviving disks, i.e., the fraction of sparing, f_{sp}; 2) the latency of the RAID controller r_t; and 3) the time elapsed since a disk failed and the sparing process started, t_{sp}.

For this implementation of the neural-fuzzy controller the three parameters were normalized. The first parameter f_{sp} from (4.8), is in the range [0,1]. The other two parameters are normalized by making two assumptions.

The first assumption is that the latency r_t can be normalized with respect to certain upper limits of the latency that the user applications consider excessive. One example is with

the Microsoft Exchange Servers for which there are some latencies that are considered the maximum acceptable and above those latencies there can be problems such as slow response experienced by customers or timeouts in applications [Microsoft 06a]. Here, it was assumed that a delay of rt_{max}=50ms was the maximum that can be tolerated, and that was the value used in the simulation of the sparing. The normalized response time (latency) rt_n used by the fuzzy controller is the same as in (4.24)

The second assumption made is that there is a maximum time acceptable for the user without the redundancy of the data restored. This is a reasonable assumption since the purpose of a RAID system is to guarantee the redundancy of the data so there is no data loss when a disk fails. The time elapsed in the sparing process since a disk failed t_{sp} is normalized also. The maximum time allowed for a sparing to finish was assumed to be tsp_{max}=12 hours. With this assumption the normalized time elapsed in the sparing process tsp_n is the same as in (4.25)

With the three input parameters normalized, the membership function can be defined. Three linguistic values were assigned, LOW, MED, and HIG, which stand for "low", "medium" and "high" value. This is following the same technique shown by Philips et. al. [Zhang 05a]. Fig. 4.16 shows the triangular fuzzy membership functions for all three input parameters. As it can be seen, the triangular functions are the same as the ones used for the fuzzy sparing control scheme presented in the previous section 4.2. And like in the previous section, Fig.4.16 shows also an example using a dummy variable ρ that can be replaced by any one of the three normalized parameters, f_{sp}, rt_n and tsp_n. The membership function for LOW is the same (4.26), the MED membership function is the same as (4.27) and the HIG membership function is the same as (4.28).

The fuzzy value of rt_n is defined as F_{rtn}:

$$F_{rtn} = \max [\, \mu^{LOW}(rt_n),\ \mu^{MED}(rt_n), \mu^{HIG}(rt_n) \,] \tag{4.34}$$

The fuzzy value of f_{sp} is defined as F_{fsp}:

$$F_{fsp} = \max [\, \mu^{LOW}(f_{sp}),\ \mu^{MED}(f_{sp}), \mu^{HIG}(f_{sp}) \,] \tag{4.35}$$

The fuzzy value of tsp_n is defined as F_{tspn}:

$$F_{tspn} = \max [\, \mu^{LOW}(tsp_n),\ \mu^{MED}(tsp_n), \mu^{HIG}(tsp_n) \,] \quad (4.36)$$

The three normalized parameters (f_{sp}, rt_n, tsp_n) in the range [0,1] are the input to the fuzzifier neural net. Fig. 4.17 shows the structure of the neural net used. Notice the two sections. The first neural net section, based on the value of the normalized parameter, will output a number 0, 0.5 or 1 that will correspond to one of the three possible fuzzy values (LOW, MED, HIG). The second section implements the rule base.

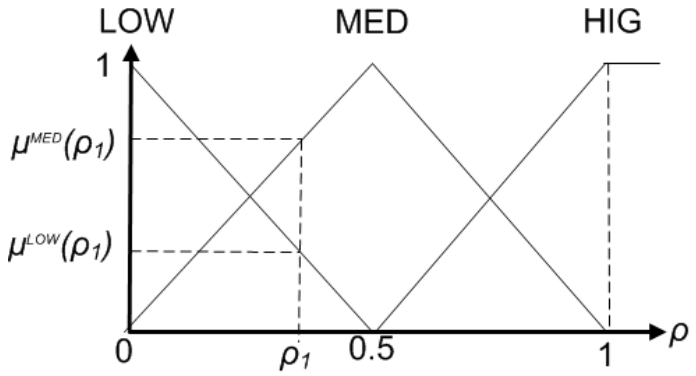

Fig. 4.16: Membership functions for the normalized parameters

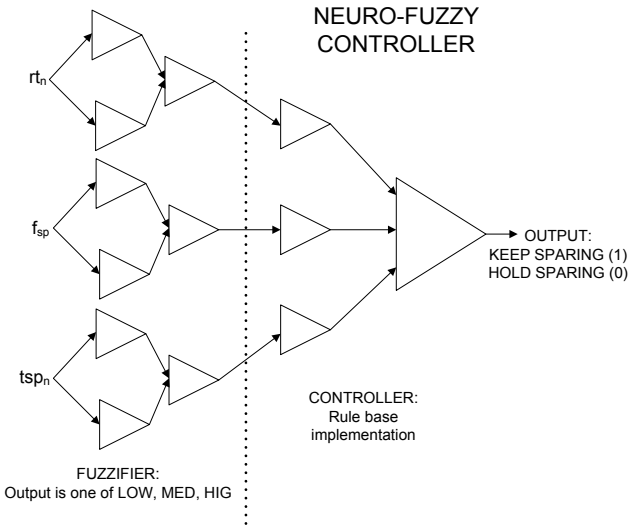

Fig. 4.17: Neural net layers of the Neural-Fuzzy controller for sparing

The rule base can be implemented according to the following linguistic criteria: 1) the latency of the RAID controller r_t, should be kept as low as possible. If the latency r_t, is LOW, the sparing can continue without any risk of affecting the user request latency. 2) the sparing process should be finished within the maximum allowed, tsp_{max}. The closer we are to HIG, the

more priority should be given to the sparing process. 3) the fraction of sparing data spared f_{sp}, should be as close to HIG as possible. If the fraction of data already spared is close to zero, then the sparing process is favored over the latency.

With these linguistic criteria, the rule base can be built. The output of each rule is a binary value of KEEP (1), which means continue the sparing process, or HOLD (0), which means to hold off the sparing process. The complete rule base is in Table 4.3. The output of the fuzzy controller is the decision to turn on/off the RAID controller to serve user requests (on) or regenerate the redundancy (off). The deffuzification of the output is done by applying the rule and the result is a zero (HOLD) or a one (KEEP)

The output of the fuzzy controller is the decision to turn on/off the disk array controller to server user requests (on) or regenerate the redundancy (off). The deffuzification of the output is done by applying the rule and the result is a zero (NO) or a one (YES). We define the output set as the crisp set:

$$C_{out\_nf} = \{\, \text{KEEP},\ \text{HOLD} \,\} \tag{4.37}$$

The rules are of the form:

$$if\ \ \boldsymbol{rt_n} \in F_{rtn}\ \ and\ \ \boldsymbol{f_{sp}} \in F_{fsp}\ \ and\ \ \boldsymbol{tsp_n} \in F_{tspn}\ then\ \ \boldsymbol{out}\ \in C_{out\_nf} \tag{4.38}$$

Table 4.3: Rule base Neural-Fuzzy controller for Sparing

Rules 1-9				Rules 10-18				Rules 19-27			
rt_n	f_{sp}	tsp_n	*OUT*	rt_n	f_{sp}	tsp_n	*OUT*	rt_n	f_{sp}	tsp_n	*OUT*
LOW	LOW	LOW	KEEP	MED	LOW	LOW	KEEP	HIG	LOW	LOW	HOLD
LOW	LOW	MED	KEEP	MED	LOW	MED	KEEP	HIG	LOW	MED	HOLD
LOW	LOW	HIG	KEEP	MED	LOW	HIG	KEEP	HIG	LOW	HIG	KEEP
LOW	MED	LOW	KEEP	MED	MED	LOW	KEEP	HIG	MED	LOW	HOLD
LOW	MED	MED	KEEP	MED	MED	MED	KEEP	HIG	MED	MED	HOLD
LOW	MED	HIG	KEEP	MED	MED	HIG	KEEP	HIG	MED	HIG	KEEP
LOW	HIG	LOW	KEEP	MED	HIG	LOW	HOLD	HIG	HIG	LOW	HOLD
LOW	HIG	HIG	KEEP	MED	HIG	MED	KEEP	HIG	HIG	MED	HOLD
LOW	HIG	HIG	KEEP	MED	HIG	HIG	KEEP	HIG	HIG	HIG	KEEP

4.3.1 SIMULATION AND RESULTS

The complete model of the RAID1 system is based on the model described in 4.1.3 with that addition of a central server and the QSV as shown in Fig. 4.2. The user requests to the queuing system arrive at a rate λ. The RAID controller (the server) processes requests at a μ service rate. Fuzzy logic offers the possibility of easily modeling and controlling systems in which mathematical models can be hard to derive. A simulation of the queuing system in Fig. 4.15 shows the improvements made by the neural-fuzzy controller. The disk parameters used for this simulation were presented at the end for section 4.1.2. The neural network training was performed in Matlab. The resulting weights and biases were translated into the simulation. The simulation was done using the CSIM19 toolkit, which allows the discrete-event simulation models [Mesquite 07a]. The testing parameters were chosen to resemble a typical Exchange Server environment [Microsoft 06b]: 75% reads (3:1 ratio). A RAID1 system with 60 ST373454FC Seagate disks was simulated. The RAID controller had a μ = 10,000 IO/s.

The throughputs 1000, 2000, 4000 and 8000 IO/s, were maintained constant during the entire duration of the simulation, in order to measure the variations in latency and the duration of the sparing process.

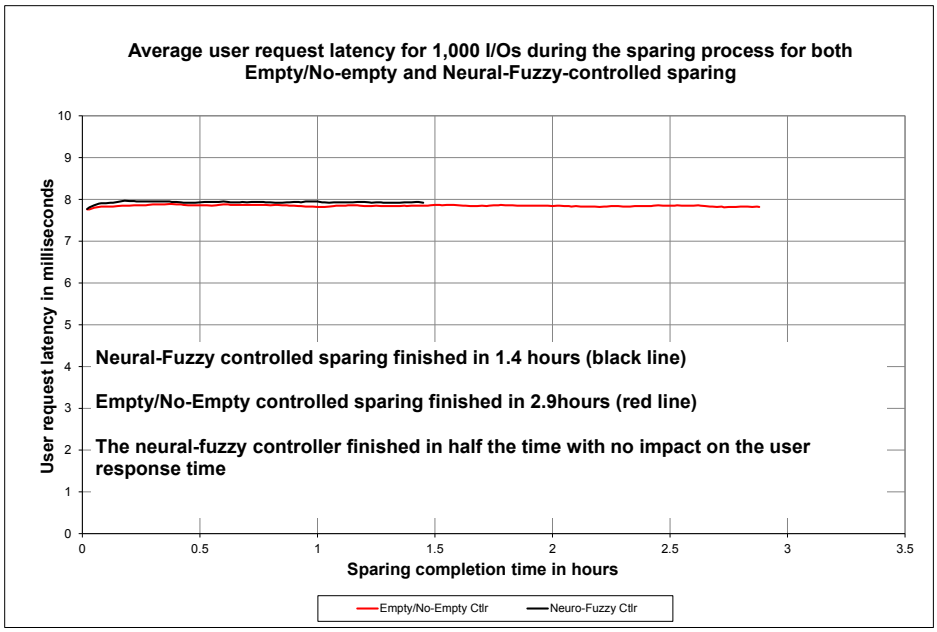

Fig. 4.18: User request latency comparison for 1,000 IO/s with neural-fuzzy control

The horizontal axis shows the total time taken for the sparing process to complete. The vertical axis shows the latency seen by the user requests.

Fig. 4.18 shows the result for the 1,000 IO/s applied to the disk array. This result shows the neural-fuzzy controller outperforming the traditional empty/no-empty controlled sparing. The graph shows the neurro-fuzzy-controlled sparing finishing in 1.4 hours after the disk failed; whereas the empty/no-empty sparing finished in 2.9 hours. For both cases, the latency was around 8ms. There is an improvement in the reduction of the sparing time by half with no impact on the user request latency.

Fig. 4.19 shows the result for the 2,000 IO/s applied to the disk array. This result shows the neural-fuzzy controller outperforming the traditional empty/no-empty controlled sparing. The graph shows the neural-fuzzy-controlled sparing finishing in 1.5 hours after the disk failed; whereas the empty/no-empty sparing finished in 3.1 hours. For both cases, the latency was around 9ms. There is an improvement in the reduction of the sparing time by half with no impact on the user request latency.

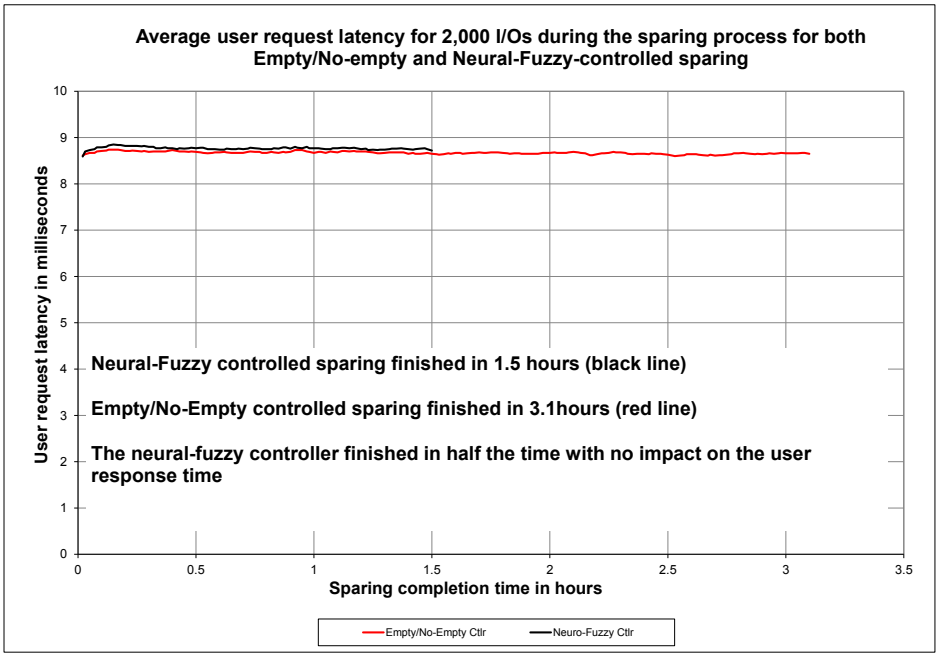

Fig. 4.19: User request latency comparison for 2,000 IO/s with neural-fuzzy control

Fig. 4.20 shows the result for the 4,000 IO/s applied to the disk array. This result shows the neural-fuzzy controller outperforming the traditional empty/no-empty controlled sparing. The graph shows the neural-fuzzy-controlled sparing finishing in 1.7 hours after the disk failed; whereas the empty/no-empty sparing finished in 3.8 hours. For both cases, the

latency was around 11ms. There is an improvement in the reduction of the sparing time of 55% less time with no impact on the user request latency.

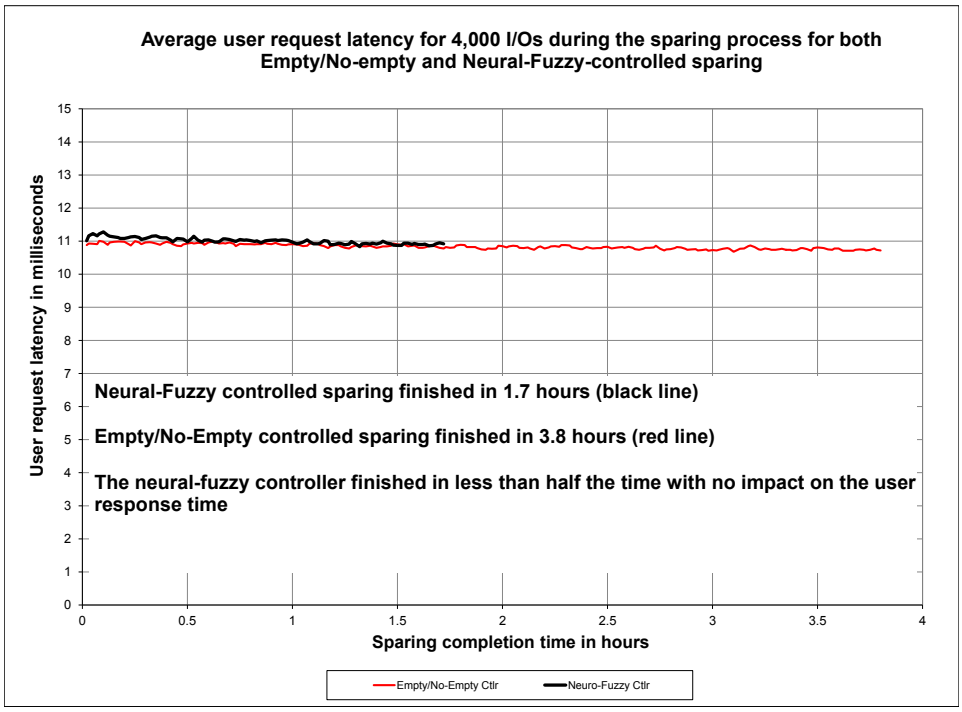

Fig. 4.20: User request latency comparison for 4,000 IO/s with neural-fuzzy control

Fig. 4.21 shows the result for the 8,000 IO/s applied to the disk array. This result shows the neural-fuzzy controller outperforming the traditional empty/no-empty controlled sparing. The graph shows the neural-fuzzy-controlled sparing finishing in 4.2 hours after the disk failed; whereas the empty/no-empty sparing finished in 7.5 hours. For both cases, the latency was around 19.5ms. There is an improvement in the reduction of the sparing time of 44% with no impact on the user request latency.

Table 4.4 presents a comparison of the four results, showing an improvement of the sparing process by cutting the duration in half with no impact on the user request latency.

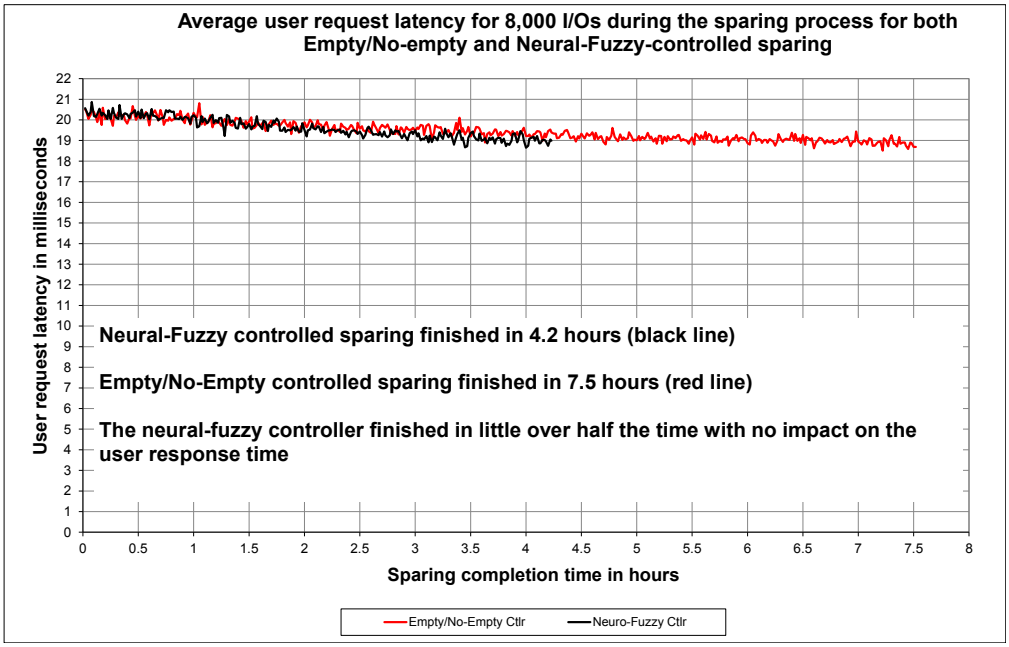

Fig. 4.21: User request latency comparison for 8,000 IO/s with neural-fuzzy control

Table 4.4: Comparison of results of the neural-fuzzy control of sparing

IO/s	Empty/No-empty controlled sparing		Fuzzy-controlled sparing	
	Sparing Duration (hrs)	Latency (ms)	Sparing Duration (hrs)	Latency (ms)
1,000	2.9	8	1.4	8
2,000	3.1	9	1.5	9
4,000	3.8	11	1.7	11
8,000	7.5	19.5	4.2	19.5

4.4 CONCLUSIONS

This chapter demonstrated that fuzzy and neural-fuzzy logic can be applied successfully to improve the sparing process in disk arrays. Both, the fuzzy-based and the neural-fuzzy controllers presented in this chapter outperformed the traditional empty/no-empty sparing process by finishing in half the time without impacting the user request latency.

CHAPTER 5: FUZZY CONTROL OF LV SNAPSHOT REPLICATION

Snapshot of logical volumes is an area of research of high interest for storage companies that aim at improving the availability of the data while at the same time providing data replication [Simitci 03a], [NetApp 07a]. Logical volume snapshot is a feature that translates into easier backup management, faster recovery, and reduced exposure to data loss [Simitci 03a], [Xiao 06a]. The snapshot feature is typically provided by storage companies like IBM (Tivoli Storage Manager), HP (Business Copy), EMC (SnapView), NetApp (SnapDrive), and Hitachi (Copy-on-Write Snapshot) [Brooks 06a][HP 08a][EMC 08a][NetApp 07a][Betrand 04a][Dufrasne 09a].

By using the snapshot feature, users can create a point-in-time copy of a logical volume (LV). From the user's standpoint, the snapshot feature creates an instant copy of the original logical volume. This gives users the means to preserve a point-in-time copy (the *snapshot*) of the data in a source logical volume. If the data in the source gets corrupted or lost, the user can go back to the snapshot and recover the data from that point in time. The original volume with the data to be replicated will be referred to as *source volume*, or just *source*, for short. The copy of the original volume will be referred to as the *snapshot volume*, or the *snapshot*, for short.

Improvements in the management of snapshot replication have been proposed in [Azagury 02a][Elnikety 05a][Shira 05a][Brinkmann 06a]. Performance improvement in terms of data transfer have been shown in [Guangjun 08a] by Guangjun. Shah proposed a Logical Volume Manager 2 (LVM2) scheme that is an optimization of LVM that improved the read performance of the snapshot volume by 40% in [Shah 06a]. Variations of the basic snapshot algorithm such incremental or iterative snapshots have been proposed before in [Zhenjun 06a][Guanping 05a][Zhong 04a]. Brinkman et al. proposed a scheme for snapshot in cluster environments [Brinkmann 07a].

This section presents a snapshot fuzzy control algorithm that significantly improves the latency of the user requests (reads or writes) during the snapshot process. The organization of this section is as follows: Section 5.1 presents the copy-on-write and redirect-on-write snapshot techniques. Section 5.2 presents a model for the snapshot and the modified process deriving a new equation for the snapshot replication process. Section 5.3 presents the fuzzy

control algorithm. Section 5.4 presents the experimental results. Section 5.5 presents the conclusions.

5.1 Background of Point-In-Time Copy Technologies

The snapshot fuzzy controller improves the latency during the snapshot process by providing an intelligent way of combining two snapshot technologies: 1) Copy-on-Write (CoW) and 2) Redirect-on-Write (RoW). These two snapshot technologies will be described in two following subsections. The classification of snapshot techniques will be based mostly on Simitci in [Simitci 03a] and Xiao in [Xiao 06a].

5.1.1 Copy-on-Write (CoW)

Source logical volumes are divided into D_{Bv} data blocks, where B_v is the total number of data blocks composing the source volume. Right after the snapshot volume is created, the pointers to data blocks on each volume (source and snapshot), point to the source volume (these pointers to data blocks are in some papers also referred to as metadata [Shah 06a]). This is illustrated in Fig. 5.1. If the user reads a block of data that has not been written to since the creation of the snapshot volume, the data will be read from the source volume. On the other hand, if the user reads a data block that has been written to since the creation of the snapshot, the data will be read from the snapshot volume. The first user write to a data block after the snapshot volume has been created will be referred to as the *first user write*.

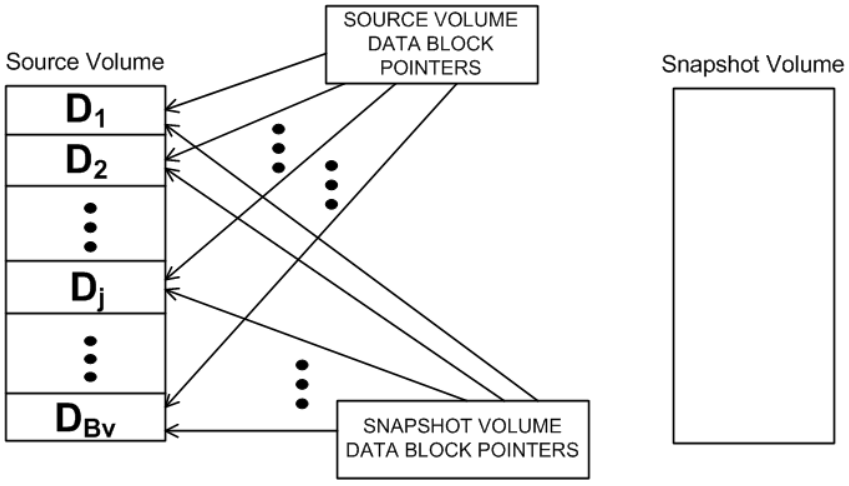

Fig. 5.1: Snapshot right after creation

If a first user write occurs to one of the data blocks in the source volume, for example D_j, then this block of data must be copied to the snapshot volume before that first user write occurs so that the original point-in-time data block D_j is preserved. Once the first user write occurs, the D_j data block in the source volume is modified so it is now referred to as the *updated* D_j*'* data block. This snapshot technology is called *copy-on-write* (CoW) because

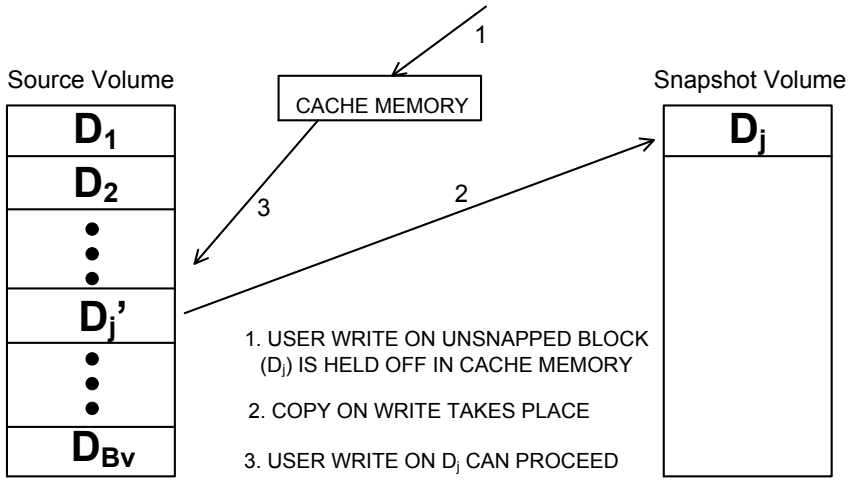

Fig. 5.2: Snapshot copy-on-write process

every first user write to the source volume causes the disk array to copy the original data block from the source to the snapshot volume before proceeding with the user write. The copy of a data block to the snapshot volume before the first user write can occur adds an extra delay to that first user write, as it has to wait for the copy. The extra delay is called the copy-

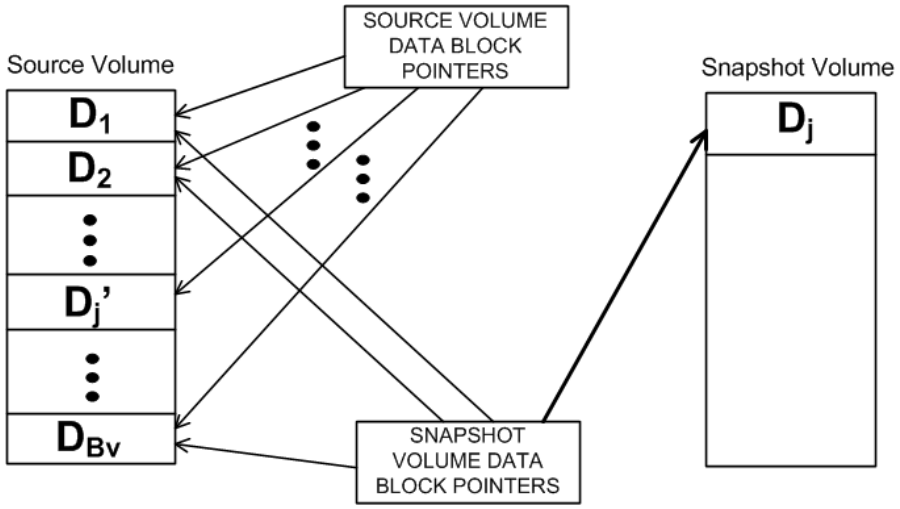

Fig. 5.3: Snapshot after copy-on-write

on-write *penalty*. When a data block from the source volume has been copied to the snapshot volume, then the original data block is said to have been *snapped*.

After the copy-on-write is accomplished, the pointers to the respective data blocks (metadata) must be updated. Fig 5.3 now shows the source volume with the updated D_j' block and the snapshot volume with the original D_j block. The snapshot volume data block pointers have to point to the original data blocks to maintain access to the point-in-time data. Therefore, the snapshot volume data block pointer to the original D_j block now points to the snapshot volume because that is where the original D_j block is preserved now. If the user accesses the snapshot volume, the user will be able to read the original D_j data block. If the user accesses the source volume, the user will read the newly updated D_j' data block. Fig 5.3 illustrates the space efficiency advantage of the snapshot solution. The space used on the snapshot volume is used only if there are new first writes to the source volume. Hence, subsequent writes to the same data block will not cause a copy-on-write.

5.1.2 REDIRECT-ON-WRITE (ROW)

In case of RoW, the new user writes to the source volume are redirected to another volume, set aside for the snapshot [Xiao 06a]. This redirection avoids the copy-on-write penalty since the writes proceed without the need of a copy-on-write of the original data to the snapshot volume. But in this case, the original volume still contains the original point-in-time

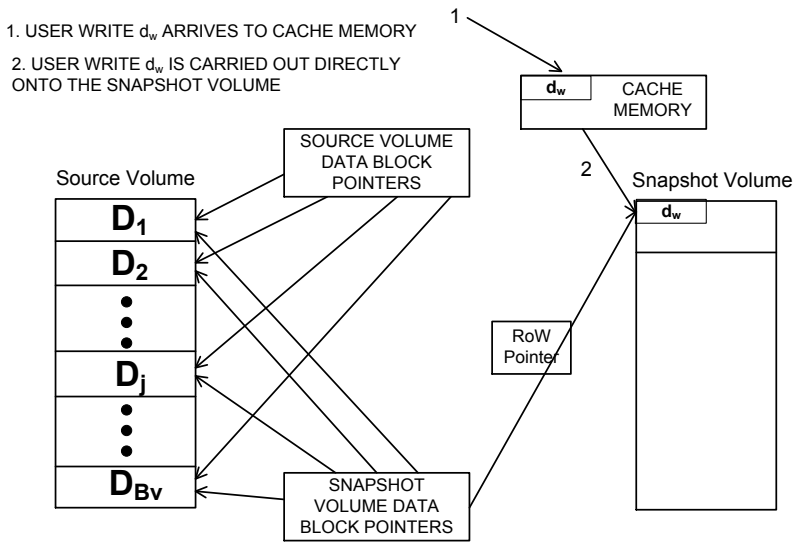

Fig. 5.4: User data write after redirect-on-write

data, while the snapshot volume contains the updated block, which is the reversal of the copy-on-write scenario. See Fig. 5.4.

5.2 Modeling of the Copy-On-Write Snapshot

5.2.1 Markov Chain Model of the Probability of a Snap

The purpose of this section is to derive the equations for the probability of a copy-on-write (CoW). In this section, the term *snap* will be used as a synonym for copy-on-write. The snapshot process can be modeled by a Markov Chain (MC) with a finite number of states under three considerations: 1) the write workload applied to the source volume is randomly distributed over the source volume, 2) the size (in KiB) of the user writes to the source volume is constant, and 3) writes to the source volume do not cross data block boundaries, that is, a write will only modify the data within one data block. These assumptions are in line with the accesses to databases, like Oracle ™ [Chan 08a]. The process can be understood intuitively by explaining how the snapping occurs. At the beginning, right after a snapshot volume has been created, the snapshot volume is empty. After the creation of the snapshot volume, write requests from a user come at a constant rate λ into the source volume. Since no data blocks have been snapped, the writes will cause a snap to occur. In other words, the probability is one that a write will cause a snap right after the snapshot volume is created. As more data blocks are snapped, the probability of a user write causing a snap will decrease. The sum of the snapped data blocks for a volume will be denoted by b and B_v is the total number of blocks that make up the source volume. The probability of a write causing a snap then is:

$$P_{snap} = \frac{B_v - b}{B_v} \tag{5.1}$$

This formula corresponds to the intuitive expectation. If no data blocks have been snapped, then $b = 0$ and the probability of a user write causing a snap is 1. If all of the data blocks have been snapped, then $b = B_v$, and the probability of a write causing a snap is zero, which means no more snaps will occur. The Markov Chain that models those probabilities is shown in Fig. 5.5

To derive the equation for the transient analysis of the MC, differential equations were obtained assuming equilibrium in terms of the input and output flow from each state [Kleinrock 75a]. The differential equation for the probability of being in the state P_0 at time t is:

$$\frac{dP_0(t)}{dt} = -\lambda P_0(t) \quad \text{with } P_0(0) = 1 \tag{5.2}$$

The solution of (5.2) is:

$$P_0(t) = e^{-\lambda t} \tag{5.3}$$

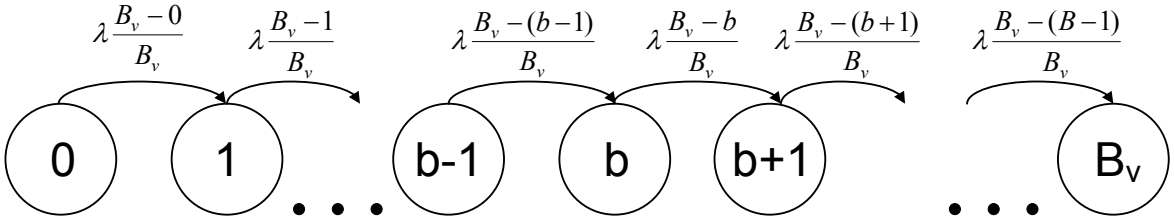

Fig. 5.5: Markov chain of copy-on-write Snapshot

The differential equation for the probability of $P_1(t)$ is:

$$\frac{dP_1(t)}{dt} = \lambda P_0(t) - \lambda\left(\frac{B_v - 1}{B_v}\right)P_1(t) \tag{5.4}$$

The solution of (5.4) is:

$$P_1(t) = B_v(1 - e^{-\frac{\lambda t}{B_v}})e^{-\frac{\lambda t}{B_v}(B_v - 1)} \tag{5.5}$$

The differential equation for the probability of $P_2(t)$ is:

$$\frac{dP_2(t)}{dt} = \lambda\left(\frac{B_v - 1}{B_v}\right)P_1(t) - \lambda\left(\frac{B_v - 2}{B_v}\right)P_2(t) \tag{5.6}$$

The solution of (5.6) is:

$$P_2(t) = \frac{(B_v - 1)B_v}{2}(1 - e^{-\frac{\lambda t}{B_v}})^2 e^{-\frac{\lambda t}{B_v}(B_v - 2)} \tag{5.7}$$

By induction, the probability of being in state *b* is:

$$P_b(t) = \frac{B_v!}{b!(B_v - b)!}(1 - e^{-\frac{\lambda t}{B_v}})^b e^{-\frac{\lambda t}{B_v}(B_v - b)} \tag{5.8}$$

The factorial term in equation (5.8) is a binomial coefficient, so the equation now becomes:

$$P_b(t) = \binom{B_v}{b}(1 - e^{-\frac{\lambda t}{B_v}})^b e^{-\frac{\lambda t}{B_v}(B_v - b)} \tag{5.9}$$

Equation (5.9) can be interpreted as the probability of having *b* blocks snapped at time *t*.; the snapshot process for a constant write arrival rate λ is governed by a binomial distribution.

5.2.2 Practical Snapshot probability equation

Since the generation of CoW is a binary event in which a user write may or may not generate a CoW, it is not unexpected to have obtained an equation of a binomial probability. Equation (5.9) has the form of a binomial probability mass function (p.m.f.):

$$p_n(k) = \binom{n}{k} p^k q^{(n-k)} \tag{5.10}$$

where the equivalent terms are:

$$n = B_v, \quad k = b, \quad q = e^{-\frac{\lambda}{B_v}t}, \quad p = 1 - q$$

The problem with (5.9) is that for practical uses, the number of blocks B_v that make up a volume is large. For example, a 64GiB source volume will be made up of B_v = 64GiB/128KiB = 524,288 blocks. Obtaining the factorial of such big numbers can render the use of (5.9) impractical. Factorials as big as this are not computed in practice. That is why the authors propose the use of the equivalent terms *p* and *q* of the binomial p.m.f:

$$q = e^{-\frac{\lambda}{B_v}t} \tag{5.11}$$

$$p = 1 - e^{-\frac{\lambda}{B_v}t} \tag{5.12}$$

Consider the behavior of (5.11) and (5.12) at t=0 and as $t \to \infty$. At t=0, or at the beginning of the snapshot process, the probability of causing a snap is one, as it has been established by (5.9). It can be observed that (5.11) has a value of one at t=0 and (5.12) has a value of zero. As time goes by and the user writes keep arriving at a λ rate into the source volume, the value of (5.11) goes to zero. The snapshot probability equation $p_{snap}(t)$ is then:

$$p_{snap}(t) = e^{-\frac{\lambda}{B_v}t} \tag{5.13}$$

The probability of not causing a snap would be described by (5.12) and it could be now taken as the probability of not having a snapshot:

$$\overline{p_{snap}}(t) = 1 - e^{-\frac{\lambda}{B_v}t} \tag{5.14}$$

Equations (5.13) and (5.14) can be used to determine how the disk array will recover the latency and throughput that it had before the snapshot process started. These equations explain why user requests may experience high latencies at the start of a snapshot when the disk array is subjected to a constant arrival OLTP workload. Equation (5.13) was tested against a snapshot setup to confirm its usefulness as a prediction of the behavior of a snapshot volume under a constant OLTP workload of user writes. Fig. 5.6 shows a comparison of the predicted probability of a snapshot occurring vs. the percentage of data blocks snapped. The equation lines up almost perfectly with observed fraction (percentage) of unsnapped data blocks, i.e., data blocks to still to be snapped.

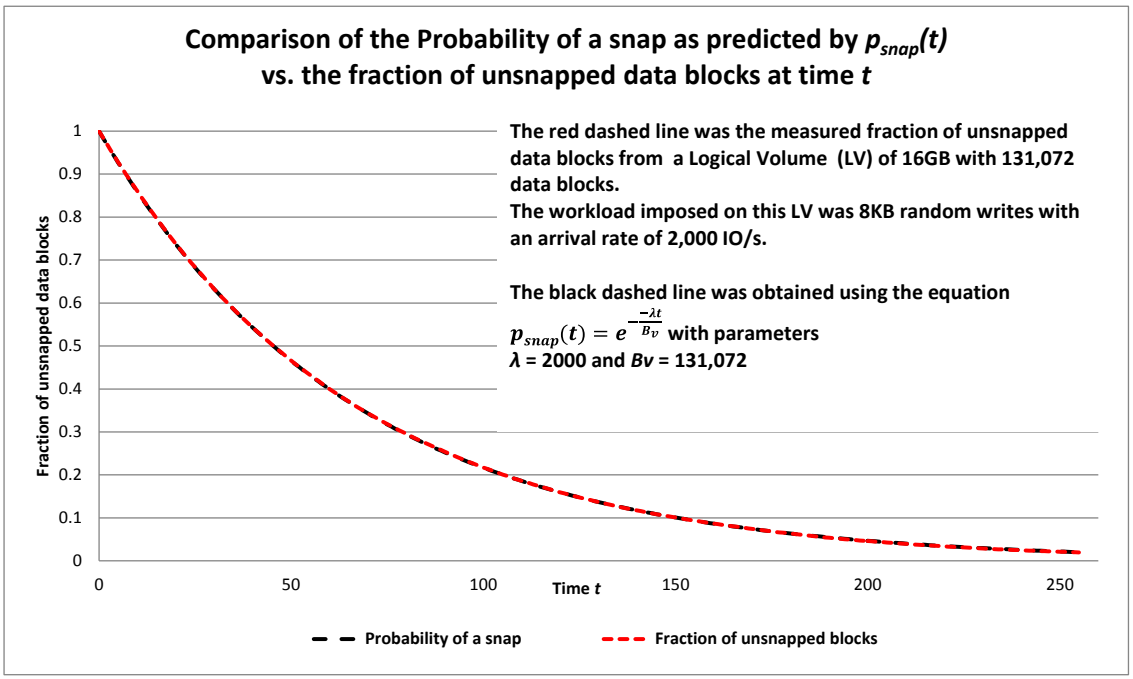

Fig. 5.6: Graph of the p_{snap} equation predicting the fraction of unsnapped blocks

5.2.3 Model of the CoW process

The model of the copy-on-write process is based on the latency delivered by disks under an OLTP workload. The two most important measures of the OLTP workload imposed on the disk array are the arrival rate in IO requests per second (IO/s) and the latency in milliseconds [ms]. Assuming the write cache memory is in write-through mode, the latency that disks deliver under certain IO/s arrival rate is the key feature that will determine the latency of the user accesses (reads or writes).

The latency of an access (read or write), t_{acc}, from a disk is a function of the arrival rate on the disk, λ_d:

$$t_{acc} = f(\lambda_d) \tag{5.15}$$

The latency introduced by the copy-on-write process, T_{cow}, is influenced by the delay of a read of the data block, T_r, from the disk where the source data block is located plus the delay of the write of that data block, T_w, to the disks where the snapshot data block will be located. This can be expressed as:

$$T_{cow} = T_r + T_w \tag{5.16}$$

The capital "*T*" letters indicate that the latency is for large block transfers. The data blocks copied during the copy-on-write process are large in size compared to the user writes. For example, data blocks can be 128KiB in size whereas user writes can be 8KiB in size.

A flow of user writes is received by a disk array. Some of the user writes, according to the p_{snap} probability will cause a snap, and therefore those user writes will have to wait for the copy-on-write before being carried out (copy-on-write penalty). And some of the other writes, according to the 1-p_{snap} probability, will be carried out immediately. The arrival rate of the user writes, λ_w, along with the p_{snap} probability, determines the arrival rate all disks in the disk array will receive, λ_D. Fig. 5.7 illustrates this process.

The copy-on-write process causes extra disk accesses on the disk array. If a write to a data block causes a snap that triggers a copy-on-write, then a data block (for example, 128KiB in size), has to be read from a disk and it has to be written on some other disks depending on the RAID level used by the snapshot volume. For example, if RAID1 is used on the snapshot volume, then a copy-on-write will generate one read of a data block from a disk and two writes to different disks. Therefore, three more accesses on disks in the disk array were generated in the background. The accesses generated by the copy-on-write that depend on the RAID level of the snapshot volume defined by the α_{RL} factor. For RAID1 the α_{RL} =2, which is the number of disk writes needed for each user data write.

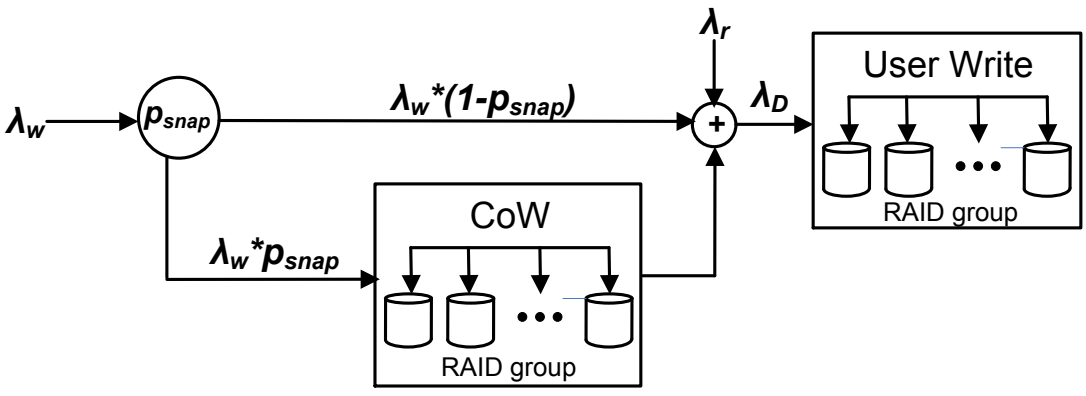

Fig. 5.7: User writes arrival rate and arrival rate caused by snaps

The total extra arrival rate on the disk array generated by the copy-on-writes, λ_{cow}, is:

$$\lambda_{cow} = (1 + \alpha_{RL})\lambda_w p_{snap} \tag{5.17}$$

The total arrival rate on the disk array, λ_D, including user reads, is:

$$\lambda_D = \lambda_r + \alpha_{RL}\lambda_w + \lambda_{cow} \tag{5.18}$$

For the sake of simplicity, it was assumed that the arrival rate is balanced across all the disks in a disk array, N_d, and the arrival rate on each disk is

$$\lambda_d = \lambda_D / N_d \tag{5.19}$$

The snapshot process occurs while users are accessing a disk array. If a user write causes a snap to occur, then it has to wait for the snap to take place before proceeding (the copy-on-write penalty):

$$t_{cow} = t_w + T_{cow} \tag{5.20}$$

The average time for the user writes is:

$$\overline{t_w} = t_{cow} p_{snap} + (1 - p_{snap}) t_w \tag{5.21}$$

This can be more simply expressed by combining (5.20) and (5.21):

$$\overline{t_w} = t_w + T_{cow} p_{snap} \tag{5.22}$$

5.2.4 MODEL OF THE PROPOSED COW-ROW PROCESS

This dissertation presents a snapshot process that reduces the latency during the snapping of the source volume. It is a combination of the CoW and RoW processes, facilitated by the fuzzy controller.

The snapshot process is modified by introducing a control input parameter named *snap throttle factor* u_{th}. This actuating variable (control input), represents the percentage of copy-on-write that will be allowed out of the all the snaps generated by user writes. The other snaps will generate a redirect-on-write. The modified CoW-RoW process is illustrated in Fig. 5.8.

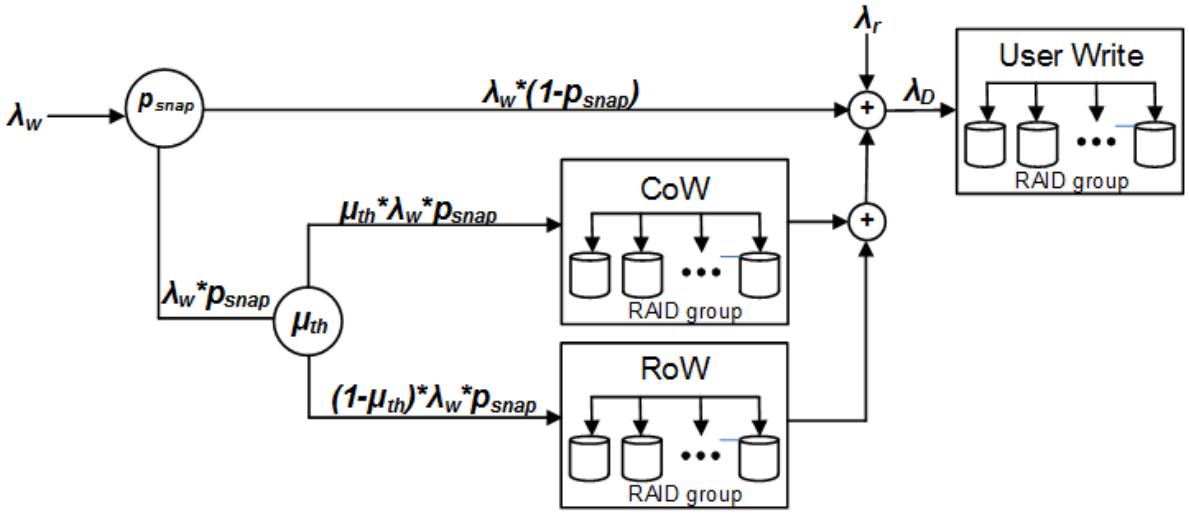

Fig. 5.8: Modified CoW-RoW process

The modified CoW-RoW process now redirects a fraction of the copy-on-writes to redirects-on-write. The reduction in the number of copy-on-writes reduces the arrival on the disks which in turn reduces their latency. The extra arrival rate on the drives is now:

$$\lambda_{row-cow} = \lambda_w[u_{th}(1 + \alpha_{RL})p_{snap} + (1 - u_{th})\alpha_{RL}] \tag{5.23}$$

And the total arrival rate on the disk array, λ_D, including user reads, is:

$$\lambda_D = \lambda_r + \lambda_w\alpha_{RL} + \lambda_{row-cow} \tag{5.24}$$

The user writes now will experience smaller latencies since the delay introduced by the redirect-on-writes, t_{row} is significantly lower than t_{cow}. The average latency experienced by user writes with the modified CoW-RoW process is expressed in the following equation:

$$\overline{t_w} = (1 - p_{snap})t_w + [\mu_{th}t_{cow} + (1 - \mu_{th})t_{row}]p_{snap} \quad (5.25)$$

One possible simplification can be made if it is assumed that the redirects-on-write are the same as user writes, since the user write is redirected to the snapshot volume instead of the source volume but with no other extra step in the process. This further entails that $t_{row} \approx t_w$, and (5.25) can be simplified as:

$$\overline{t_w} = t_w + \mu_{th}T_{cow}p_{snap} \quad (5.26)$$

This equation shows why the latency is better with the CoW-RoW process if the snap throttle factor, u_{th}, is less than 1. The determination of the input control u_{th} and the control of the snapshot process with the fuzzy control are explained in the next section.

5.3 Snapshot Fuzzy Control

5.3.1 Purpose and Rationale of the Snapshot Fuzzy Controller

The snapshot fuzzy controller can be considered as a dynamic and optimal Takagi-Sugeno fuzzy-logic based controller. The block diagram of the snapshot fuzzy controller is illustrated in Fig. 5.9. The purpose is to minimize the average latency of user accesses t_w, and t_r during a snapshot process by controlling the dynamics of the snapshot process.

5.3.2 High Level Modeling of the Snapshot Fuzzy Controller

From a control standpoint the disk array is the controlled system. The controlled system has two inputs: the arrival rate of writes, λ_w, and the arrival rate of reads, λ_r.

The total arrival rate, λ, is the sum of the input parameters:

$$\lambda = \lambda_r + \lambda_w \quad (5.27)$$

The outputs of the system to be controlled (disk array) are the average latencies experienced by the user accesses (reads or writes), t_r, and t_w:

$$y(t_i) = [y_1 \quad y_2] = [t_w \quad t_r] \tag{5.28}$$

The state variables required for the snapshot fuzzy controller are 1) the probability of snapped blocks in the volume, p_{snap}, which is a value in the [0,1] range; and 2) the numbers of copy-on-writes per time unit, in other words, the arrival rate of copy-on-writes in the disk array, λ_{cow}.

$$x(t_i) = [x_1 \quad x_2] = [p_{snap} \quad \lambda_{row-cow}] \tag{5.29}$$

The control input variable is the snap throttle factor, u_{th}

$$u(t_i) = [u_1] = [u_{th}] \tag{5.30}$$

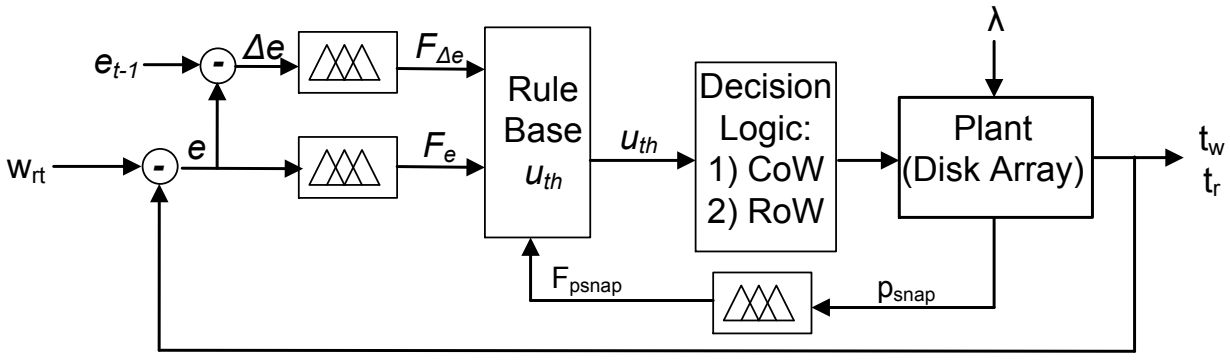

Fig. 5.9: Snapshot fuzzy controller

The snapshot fuzzy controller also requires a reference variable, the *reference latency* w_{rt}, the maximum acceptable user request latency during the snapshot process. The maximum latency used in this section was 30ms, from the Oracle ™ performance tuning guide [Chan 08a] as a latency value that gives a good indication of an overly active I/O system.

The outputs have to be periodically monitored every T_m seconds. The decision on how often to monitor can be based on the maximum acceptable latency and the performance of the disk array controller. Each sample is denoted by (t_i), where i is the *i-th* sample of the output that occurred at a time t_i, as in:

$$t_i = iT_m \qquad \text{where } i = 0,1,2,\ldots \tag{5.31}$$

5.3.3 Decision Logic

If a user write causes a snap, then the snapshot fuzzy controller makes a decision about the three possible choices to execute: 1) perform a copy-on-write at the time when the user write is being served; 2) defer the copy-on-write operation by executing a redirect-on-write; 3) perform a copy-on-write of the target data block if a redirect-on-write already took place for that data block. The fuzzy controller throttles the snapshot process by controlling the percentage of copy-on-writes that are caused by user writes (option 1), versus the percentage of user writes with deferred copy-on-write (option 2). This percentage is the output of the snapshot fuzzy controller and is named *snap throttle factor* u_{th}. For example, if $u_{th} = 0.4$, this means that only 40% of the user writes that cause a snap will also generate a copy-on-write. The other 60% of the user writes that are causing a snap will generate a redirect-on-write.

5.3.4 Estimation and fuzzification of the probability of a snap

The probability of a snap is used as part of the determination of the snap throttle factor. The $f_{snap}(t_i)$, in addition to being an indication of the percentage of blocks snapped at a time t_i, also denotes the probability of further snaps. For example, if 90% of the blocks in a volume have been snapped, the probability of user accesses causing further snaps is only 10% (assuming a random user access over the volume). The probability of a snap at time t_i is:

$$p_{snap}(t_i) = 1 - f_{snap}(t_i) \tag{5.32}$$

The probability of a snap $p_{snap}(t_i)$, the error $e(t_i)$, and the change in error $\Delta e(t_i)$, are the three variables used by the fuzzy controller to compute the snap throttle factor, $u_{th}(t_i)$. These

three variables need to be first fuzzified as shown in [Michels 06a]. The fuzzification of p_{snap} is done in very straightforward fashion. If the probability of snap is less than or equal to 0.5, it is mapped to the *Low Probability* (LP) fuzzy descriptor. If the probability of a snap is greater than 0.5, it is mapped to the the *High Probability* (HP) fuzzy descriptor. The membership function of probability of a snap is therefore defined by:

$$\mu_{psnap}(p_{snap}) = \begin{cases} 0 & if \ p_{snap} \le 0.5 \\ 1 & if \ p_{snap} > 0.5 \end{cases} \tag{5.33}$$

The final fuzzification of the p_{snap} value is denoted by $F_{psnap}(\mu_{snap})$, and is defined as:

$$F_{psnap}(\mu_{snap}) = \begin{cases} LP & if \ \mu_{snap} = 0 \\ HP & if \ \mu_{snap} = 1 \end{cases} \tag{5.34}$$

5.3.5 Control Error computation and fuzzification

The output $y(t_i)$ is compared with the reference latency w_{rt} to compute the control error, e:

$$e(t_i) = y(t_i) - w_{rt} \tag{5.35}$$

The change in the control error, Δe, is also computed:

$$\Delta e(t_i) = e(t_i) - e(t_{i-1}) \tag{5.36}$$

The final goal in the fuzzification of the control error e and change in the control error Δe is to map them to one of three fuzzy descriptors, *Zero* (ZE), *Positive Error* (PE), and *Negative Error* (NE), respectively. These fuzzy descriptors apply to both the control error e and change in control error Δe. The purpose of these fuzzy descriptors is to indicate when the control error is close to zero, or in case where the error does exist, whether the control error is positive or negative. This fuzzification is first performed via three triangular membership

functions, μ^{ZE}, μ^{NE} and μ^{PE}, based on the reference latency w_{rt}. The membership functions are described using a dummy variable error, ε, since these membership functions are the same for both e and Δe:

$$\mu_e^{ZE}(\varepsilon, w_{rt}) = \begin{cases} 1 - \dfrac{2\varepsilon}{w_{rt}} & if\ \varepsilon > 0 \\ 1 & if\ \varepsilon = 0 \\ 1 + \dfrac{2\varepsilon}{w_{rt}} & if\ \varepsilon < 0 \end{cases} \tag{5.37}$$

$$\mu_e^{PE}(\varepsilon, w_{rt}) = \begin{cases} 1 & if\ \varepsilon \geq w_{rt} \\ \dfrac{4\varepsilon}{3w_{rt}} - \dfrac{1}{3} & if\ \varepsilon\ in\ (\dfrac{1}{4}w_{rt}, w_{rt}) \\ 0 & if\ \varepsilon \leq \dfrac{1}{4}w_{rt} \end{cases} \tag{5.38}$$

$$\mu_e^{NE}(\varepsilon, w_{rt}) = \begin{cases} 0 & if\ \varepsilon \geq -\dfrac{1}{4}w_{rt} \\ -\dfrac{4\varepsilon}{3w_{rt}} - \dfrac{1}{3} & if\ \varepsilon\ in\ (-\dfrac{1}{4}w_{rt}, -w_{rt}) \\ 1 & if\ \varepsilon \leq -w_{rt} \end{cases} \tag{5.39}$$

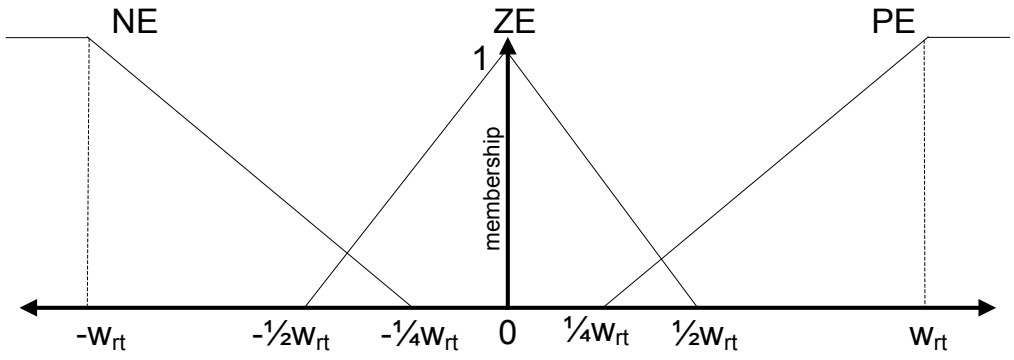

Fig. 5.10: Membership functions for e and Δe

The membership functions (5.37), (5.38) and (5.39) here shown are for the control error e (if $\varepsilon = e$), and for the change in control error Δe (if $\varepsilon = \Delta e$). The graphical representation of the membership functions is shown in Fig. 5.10.

To finish the fuzzification, the control error e and the change in control error Δe are mapped into one of the fuzzy descriptors (NE, ZE, or PE). This is accomplished by comparing the values obtained for the three membership functions (5.37), (5.38), and (5.39). Depending on which of the three has the maximum value the fuzzy value of the error F_e, and the fuzzy value of the change in error $F_{\Delta e}$, are mapped into one of the fuzzy descriptors NE, ZE or PE:

$$F_e = \max(\mu_e^{NE}, \mu_e^{ZE}, \mu_e^{PE}) \tag{5.44}$$

$$F_{\Delta e} = \max(\mu_{\Delta e}^{NE}, \mu_{\Delta e}^{ZE}, \mu_{\Delta e}^{PE}) \tag{5.45}$$

For example, if the output $y(t_1)$ is 45ms, then using (5.35) the error e is 15ms. The membership values, obtained by using (5.37), (5.38) and (5.39), are μ^{ZE}=0, μ^{NE} =0, and μ^{PE}=1. It is clear that the maximum value corresponds to μ^{PE}. Using (5.44), the fuzzy value of the error F_e will be mapped to Positive Error, PE. This same procedure is used for the change in error to map it into one of the fuzzy descriptors, NE, ZE or PE.

5.3.6 Rule Base to obtain U_{TH}

The rule base can now be built based on the following heuristic criteria: (1) if the user request latency is high, then the control error, e, is fuzzy positive error, PE, and the controller needs to reduce the number of copy-on-writes occurring. Therefore, the snap throttle factor u_{th} is reduced. (2) if the user request latency is low, then the controller can increase the number of copy-on-writes occurring. Therefore, the snap throttle factor u_{th} is increased. Otherwise, u_{th} stays the same. The probability of more copy-on-writes and the change in error are also taken into account.

Once the three fuzzified input variables e, Δe, and p_{snap}, are estimated, is the evaluation of the fuzzy rules. The output of the fuzzy rules is the *change in snap throttle*

Table 5.1: Rule base for Snapshot Fuzzy Controller

Rule Number	Rule Input Variables			Rule Output
	p_{snap}	e	Δe	Δu_{th}
R_1	HP	PE	PE	-0.2
R_2	HP	PE	ZE	-0.1
R_3	HP	PE	NE	-0.1
R_4	HP	ZE	PE	-0.1
R_5	HP	NE	ZE	+0.05
R_6	HP	NE	NE	+0.05
R_7	LP	PE	PE	-0.05
R_8	LP	PE	ZE	-0.05
R_9	LP	ZE	ZE	+0.05
R_{10}	LP	ZE	NE	+0.05
R_{11}	LP	NE	ZE	+0.05
R_{12}	LP	NE	NE	+0.05

factor $\Delta u_{th}(t_i)$. This value will denote the change in the snap throttle factor for the current iteration. The rule base is in Table 5.1. The rules are of the form:

$$if\ p_{snap} \in F_{snap}\ and\ e \in F_e\ and\ \Delta e \in F_{\Delta e}\ then\ \mu_{th}(t_i) = \mu_{th}(t_{i-1}) + \Delta\mu_{th}(t_i) \qquad (5.46)$$

where $\Delta u_{th}(t_i)$ can be in the [-1,1] range. Based on the chosen rule, an equation (5.46) is computed for the snapshot fuzzy controller. The snap throttle factor u_{th} value is in the [0.05,1] range. The value 0.05 as the minimum for u_{th} was based on empirical observations of actual snapshot processes. This value allows some copy-on-writes to proceed and make a little progress with the snapshot. The initial values when a snapshot volume is created are $u_{th}(0)$ =0.05 and $e(0) = 0$.

5.3.7 Stability of the Fuzzy Controller

The fuzzy system presented here is globally asymptotically stable based on the fact that it meets the stability condition for the state variables, which according to [Michels 06a]

shows that state variables converge to a reference vector as time goes to infinity. In the case of the snapshot fuzz controller, the probability of a snap, p_{snap} and therefore the $\lambda_{row\text{-}cow}$ arrival rate (5.33) converges to zero as user writes access more source volume data blocks as time goes by. The exponential decrease of the probability of a snap decreases the possibility of CoWs and therefore the probability of processing the user writes with no delay is greater, which makes the snapshot fuzzy controller less likely to intervene and cause instability. The stability of the fuzzy controller is then guaranteed by the condition:

$$\lim_{t\to\infty} p_{snap}(t) \to 0 \tag{5.47}$$

5.4 Experimental Results

The snapshot fuzzy controller was tested with a setup that consisted of an HP 7640 Itanium workstation with 48GiB of memory and with RedHat Linux 6.2 installed. The disk setup consisted of 118 BF1465A477 15K RPM disks. The traditional copy-on-write and the

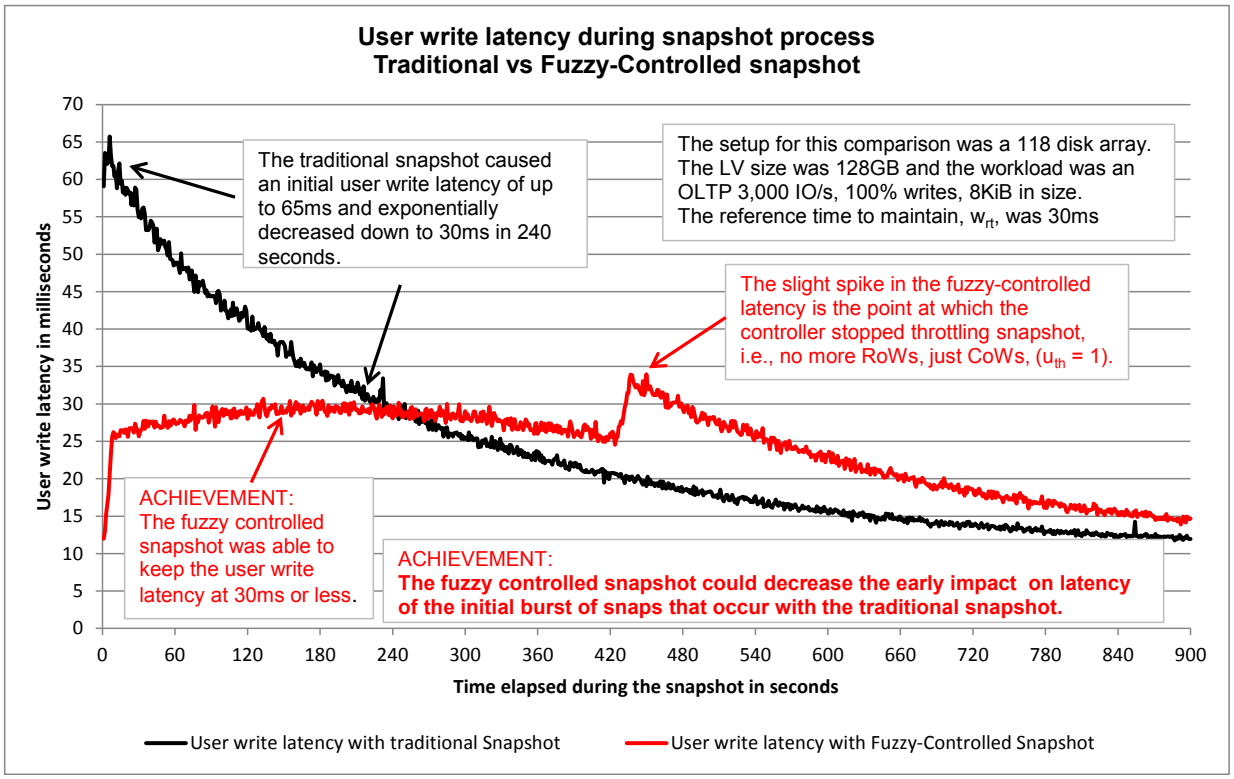

Fig. 5.11: Comparison of latency at 3,000 IO/s, 100% User writes

snapshot fuzzy controller were implemented in C language and compiled with GCC 4.4.6. The implementation was executed as a parent process in the user space and not as a part of the kernel. The parent process performed the following functions: 1) spawned user requests at a constant rate using the fork() Unix function; 2) kept track of the data blocks written, snapped and or with a redirect-on-write. The data block table was in shared memory so it could be updated by the spawned user requests; 3) monitored the latency of the user requests; 4) implemented the snapshot fuzzy control logic.

A comparison was run with an 8KiB workload, 0% reads at 3,000 IO/s. The source volume was a RAID1 128GiB in size using data blocks of 256KiB laid out in an evenly fashion over all the 118 disks. The results in Fig. 5.11 show the traditional copy-on-write implementation delivering initial latencies for user writes (black line) in the 65 ms range. The snapshot fuzzy controller implementation proved superior since it could keep the initial latency for user writes (red line) in the low 30 ms range.

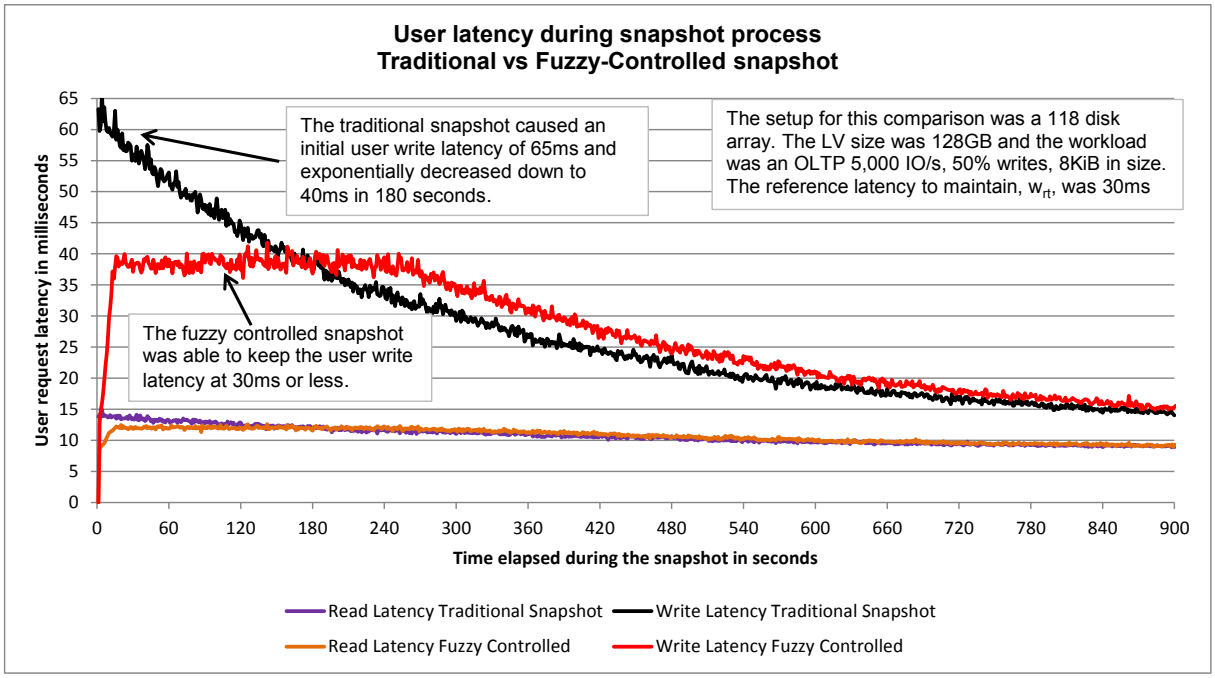

Fig. 5.12: Comparison of latency at 5,000 IO/s, 50% User writes

Another comparison was run with an 8KiB workload, 50% reads at 5,000 IO/s. The source volume was a RAID1 128GiB in size using data blocks of 256KiB laid out in an evenly fashion over all the 118 disks. The results in Fig. 5.12 show the traditional copy-on-write implementation delivering initial latencies for user writes (black line) in the 65 ms range. For user reads (purple line) the traditional copy-on-write delivered a latency in the 10-

15ms range. The snapshot fuzzy controller implementation proved superior since it could keep the initial latency for user writes (red line) in the low 35-40ms range. For the user reads (orange line), the latency delivered by snapshot fuzzy controller was in the 10-12ms range.

5.5 CONCLUSIONS

The greatest benefit the snapshot fuzzy controller delivers is to avoid the high latency peak at the beginning of a snapshot process as predicted by the equations (5.13) and (5.14) developed for the traditional copy-on-write snapshot. These equations can provide a guide for the snapshot behavior even for different disks speeds and disk arrays if the snapshot process is the traditional copy-on-write. The improvements in latency the snapshot fuzzy controller delivers show how computationally intelligent techniques, namely fuzzy logic, 1) can be applied to the data backup management for disk arrays; 2) can outperform traditional techniques like copy-on-write; 3) can be used to control the nonlinear response of disks. The reduction of the impact caused by the traditional copy-on-write is the accomplishment that meets the performability challenge imposed by the copy-on-writes.

CHAPTER 6: T2 FUZZY CONTROL OF LOGICAL VOLUME CLONING REPLICATION

The logical volume cloning replication feature offered by disk arrays provides point-in-time copies of the logical volumes to guarantee data protection to users if an event like data corruption or if accidental deletion occurs. Disk array manufacturers offer this option as part of their set of data replication features. The SnapClone feature offered by Hewlett Packard [HP 08a] and the EMC TimeFinder/Clone [EMC 05a] and the IBM Flashcopy [Garimella 06a] provide full-volume point-in-time copies of logical source volumes. The logical volume cloning replication features mentioned provide some form of background copy of the logical source volume. The details of the algorithms used for the replication are kept confidential by the vendors. This kind of logical volume replication is referred to as background because it occurs in the background while the disk array is servicing user requests (reads and writes). The term local is used to refer to the fact that the copy of the logical volume is stored in the same disk array that holds the original logical volume and the copy. In this chapter the term *cloning* or *cloning replication* will be used to refer to the local background copy of all the data in an LV (source) to another LV (clone or replica).

Interval Type-2 Fuzzy Logic Controllers (IT2 FLCs) have been proposed as a better alternative to Type-1 Fuzzy Logic Controllers (T1 FLCs) [Hagras 07a]. As stated by Mendel in [Mendel 10a], the question of establishing when and by how much Type-2 Fuzzy System (T2 FS) or Interval Type-2 Fuzzy Systems (IT2 FS) will outperform a Type-1 Fuzzy System (T1 FS) may be the most important unanswered question in the T2 field. Wu in [Wu 12b] presents a comparison between IT2 FLCs and T1 FLCs and the results show that IT2 FLCs are more adaptive and can implement more complex control surfaces than a T1 FLC with the same rule base.

The purpose of the type-2 fuzzy based control is to balance the impact on the latency of the user requests with the speed (or rate) of the replication. On one hand, the users want very little (or no) impact on the latency of the requests (read and writes) and on the other hand, the users want the replication of the data to take the shortest time possible to protect as much data as possible in the shortest time possible.

This chapter presents a type-2 fuzzy based control of background local cloning replication. Section 6.1 explains the basic theory of type-2 fuzzy logic. Section 6.2 describes the fundamental blocks of a type-2 fuzzy controller. Section 6.3 explains the process of local

replication and the copy-on-writes requests that impact the user request latency. Section 6.4 shows the queuing scheme for the local replication. Section 6.5 presents a mathematical description of the cloning replication process and a new formula. Section 6.6 presents the type-2 fuzzy controller used for local replication. Section 6.7 will present the experimental results and section 6.8 will present the conclusions.

6.1 Interval Type 2 Fuzzy Sets

Interval Type-2 fuzzy sets (IT2 FS) are an extension of type-1 fuzzy sets [Karnik 99a]. The Interval Type-2 Fuzzy Set in Fig. 6.1 shows the graph of the membership function of a triangular IT2 FS. The horizontal axis denotes the values x in the domain X, e.g., real or integer numbers. The vertical axis denotes the membership function $u(x) \in [0,1]$ for each value of x in the domain X. An IT2 FS denoted $\tilde{X}$ can be characterized by a type-2 membership function $\mu_{\tilde{X}}(x,u)$ where $x \in X$ and $u(x) \in J_x \subseteq [0,1]$ in which $0 \leq \mu_{\tilde{X}}(x,u) \leq 1$ and can be expressed as [Karnik 99a]:

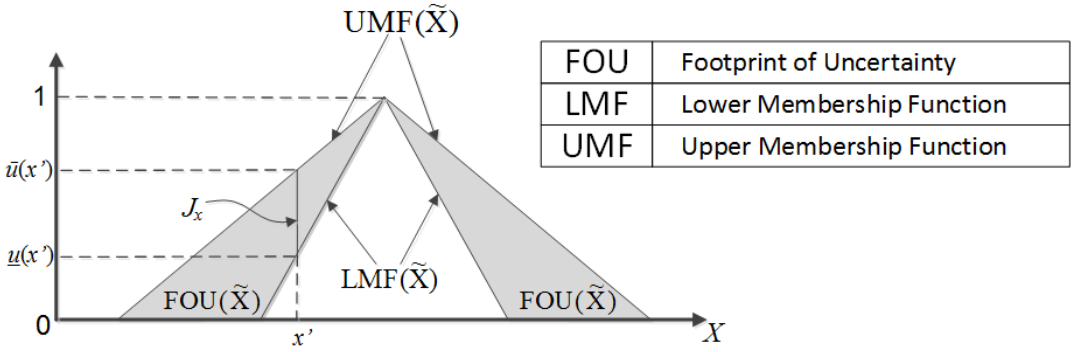

Fig. 6.1: Example of Type-2 Fuzzy Set

$$\tilde{X} = \int_{x \in X} \int_{u(x) \in J_x \subseteq [0,1]} \mu_{\tilde{X}}(x,u)/(x,u(x)) \tag{6.1}$$

where $\int$ denotes the fuzzy cardinality operator [Hanss 10a]. The $\mu_{\tilde{X}}(x,u)/(x,u(x)$ denotes the tuple *(x,u(x))* with membership function $\mu_{\tilde{X}}(x,u)$. The x and u are called the primary and secondary variables. Here, J_x is the primary membership of x and $\mu_{\tilde{X}}(x,u)$ is the secondary grade of $\tilde{X}$. For the IT2 FS the secondary grade equals 1 for $\forall x \in X$ and $\forall u \in J_x \subseteq [0,1]$. Assuming $J_x \subseteq [0,1]$, an IT2 FS $\tilde{X}$can be characterized by:

$$\tilde{X} = \int_{x\in X} \int_{u(x)\in J_x} 1/(x,u(x)) = \int_{x\in X} \left[\int_{u(x)\in J_x} 1/u\right]/x \tag{6.2}$$

The fuzzy cardinality operation inside the square brackets in (6.2) is a vertical slice of the IT2 FS. As shown in Fig. 6.1, the vertical slice for the specific value of *x*' the vertical slice is:

$$\mu_{\tilde{X}}(x',u) = \int_{u(x)\in J_x} 1/u \qquad J_x \subseteq [0,1] \tag{6.3}$$

An IT2 FS is completely determined by the union of all primary memberships, J_x, called the footprint of uncertainty (FOU):

$$FOU(\tilde{X}) = \cup_{\forall x\in X} J_x = \{(x,u(x)) : u \in J_x \subseteq [0,1] \tag{6.4}$$

An IT2 FS has an FOU that is bounded by two T1 MFs: the upper membership function (UMF) and the lower membership function (LMF). Fig.6.1 shows the FOU, UMF and LMF of a IT2 FS. With the UMF and LMF another definition of the FOU is:

$$FOU(\tilde{X}) = \cup_{\forall x\in X} (\underline{\mu}_{\tilde{X}}(x), \overline{\mu}_{\tilde{X}}(x)) \tag{6.5}$$

where $\underline{\mu}_{\tilde{X}}(x)$ is the LMF and $\overline{\mu}_{\tilde{X}}(x)$ is the UMF. These two functions are important because an IT2 FS is fully determined by if they are known.

6.2 Type 2 Fuzzy Logic Controllers (T2 FLCs)

Fig. 6.2 shows the block diagram of an IT2 Proportional-Integral-Derivative (PID) FLC. The fuzzifier maps the crisp inputs into IT2 FLC.

The rule base is composed of implicative rules of the following form:

$$\text{IF } x_1 \text{ is } \tilde{X}_1^n \text{ AND } x_2 \text{ is } \tilde{X}_2^n \text{ ... AND } x_I \text{ is } \tilde{X}_I^n \text{ THEN } y \text{ is } \tilde{Y}^n \tag{6.6}$$

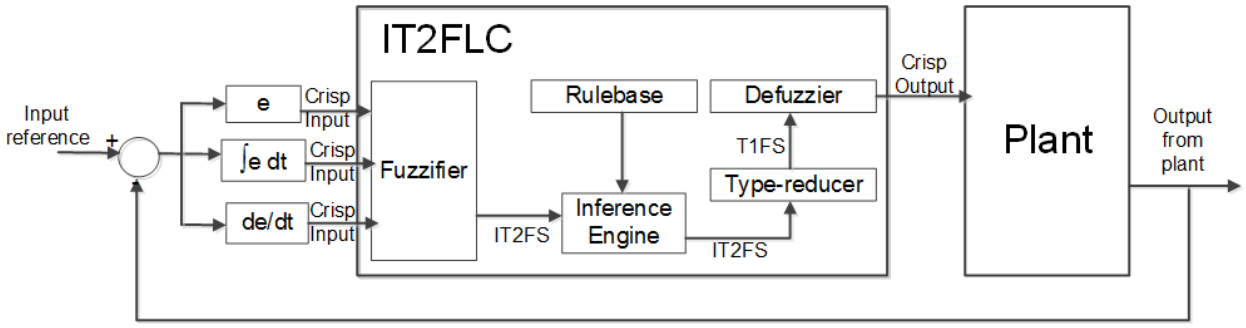

Fig. 6.2: Block diagram of an interval type-2 fuzzy controller (IT2FLC)

where I is the number of inputs (i=1,2,…,I) and N is the number of rules (n=1,2,…,N). The $\tilde{Y}^n$ are the consequent IT2 FS that are replaced by an interval $\tilde{Y}^n = [\overline{y}^n, \underline{y}^n]$ when the popular center-of-sets type reduction is used [Wu 12a]. The typical procedure performed by an IT2 FLC is the following [Wu 12a][Mendel 12a]:

1) For each input of the input vector $\mathbf{x}' = (x_1', x_2', \ldots, x_I')$ obtain the interval of membership $\left[\underline{\mu}_{\tilde{X}}(x_i'), \overline{\mu}_{\tilde{X}}(x_i')\right]$ (LMF and UMF) on each X_i^n where (i=1,2,…,I) and (n=1,2,…,N).

2) Now the firing interval of each rule is computed, e.g., for the n^{th} rule

$$F^n(\mathbf{x}') = \left[\underline{f}^n(\mathbf{x}'), \overline{f}^n(\mathbf{x}')\right] = \left[\underline{f}^n, \overline{f}^n\right] \tag{6.7}$$

where:

$$\underline{f}^n(\mathbf{x}') = \left[\underline{\mu}_{\tilde{X}_1^n}(x_1') \times \underline{\mu}_{\tilde{X}_2^n}(x_2') \times \ldots \times \underline{\mu}_{\tilde{X}_I^n}(x_I')\right] \tag{6.8}$$

$$\overline{f}^n(\mathbf{x}') = \left[\overline{\mu}_{\tilde{X}_1^n}(x_1') \times \overline{\mu}_{\tilde{X}_2^n}(x_2') \times \ldots \times \overline{\mu}_{\tilde{X}_I^n}(x_I')\right] \tag{6.9}$$

The product t-norm was used in (6.8) and (6.9) but the minimum t-norm can be used instead.

3) The type reduction is the next step. After computing the firing intervals then the centroids of all consequent sets $\tilde{Y}^n$are computed. The result is a set of R interval fuzzy sets:

$$Y_{\tilde{Y}^n}(y) = \left[y_l(\tilde{Y}^n), y_r(\tilde{Y}^n)\right] = \left[\underline{y}^n, \overline{y}^n\right] \tag{6.10}$$

The centroid is an interval T1 fuzzy set. The centroids are computed using Karnik-Mendel (KM) [Mendel 12a][Wu 12a] algorithms.

4) The firing intervals (6.8), (6.9) and their respective centroids (6.10) are combined by means of the center-of-sets (cos) type reduction [Karnik 99a]. There are other methods [Karnik 99a] but the center-of-sets is the most common.

$$Y_{cos}(y|\mathbf{x}') = \frac{\sum_{n=1}^{N} F^n(x')\, Y_{\bar{Y}^n}(y)}{\sum_{n=1}^{N} F^n(x')} = [y_l(\mathbf{x}'), y_r(\mathbf{x}')] \tag{6.11}$$

where $y_l(\mathbf{x}')$ and $y_r(\mathbf{x}')$ are computed using KM algorithms. The $y_l(\mathbf{x}')$ and $y_r(\mathbf{x}')$ can be computed by:

$$y_l(\mathbf{x}') = \frac{\sum_{n=1}^{L} \overline{f}^n \underline{y}^n + \sum_{n=L+1}^{N} \underline{f}^n \underline{y}^n}{\sum_{n=1}^{L} \overline{f}^n + \sum_{n=L+1}^{N} \underline{f}^n} \tag{6.12}$$

$$y_r(\mathbf{x}') = \frac{\sum_{n=1}^{R} \underline{f}^n \overline{y}^n + \sum_{n=R+1}^{N} \overline{f}^n \overline{y}^n}{\sum_{n=1}^{R} \overline{f}^n + \sum_{n=R+1}^{N} \underline{f}^n} \tag{6.13}$$

where L and R are the switch points determined by the KM algorithms [Mendel 12a].

5) Finally, the crisp (defuzzified) output is computed by using the average value of the two end-points [Karnik 99a]:

$$y(\mathbf{x}') = \frac{[y_l(\mathbf{x}'), y_r(\mathbf{x}')]}{2} \tag{6.14}$$

6.3 Logical Volume (LV) Cloning Replication

There are features offered by the disk array manufactures to create a point-in-time replication the data in a logical volume. The background local cloning replication copies all the data present in a logical volume, therefore creating a mirror copy (*clone* or replica) of the original (*source*) logical volume on the same disk array, thus, the clones (replicas) provide a high-availability, disaster recovery of the data. In case of a complete data loss of the source logical volume, the replica can replace the source logical volume instantaneously. Unlike the

local snapshot replication, there is no need to reconcile the source data and the updated sections. This is one of the disadvantages of the snapshot method [Preston 02a]. The drawback of the background cloning replication is that it requires the same space as the source logical volume, therefore doubling the space needed to have the source data and its clone on the disk array. But with the advent of new fast and high capacity disks, like 1 TiB magnetic drives, that is becoming less and less of a drawback. Therefore, the method of background local cloning replication of logical volumes is still a good solution for data protection.

Logical volumes can be created and deleted by users. The size of the logical volume is determined by the user at creation time. This size can be big, for example, 500 GiB. But the disk array manages the logical volume in units named *data blocks* of 128KiB, 256KiB, or other sizes depending on the manufacturer and model of the disk array. For the purpose of giving an example of the procedure used to replicate a logical volume, it is assumed in this paper that the logical volumes are managed in data blocks of 256KiB each. The first case to present is when the source logical volume is replicated (cloned) and there is no user workload (reads or writes) applied to the source logical volume. In this case, the disk array copies block by block sequentially in incremental order until all the blocks that make up the source, B_V, have been copied to the clone (replica) logical volume. This case is shown in Fig. 6.3. The circled numbers indicate the sequence of events. At the end, every single block that makes up the source logical volume is replicated into the clone logical volume. It is clear from this

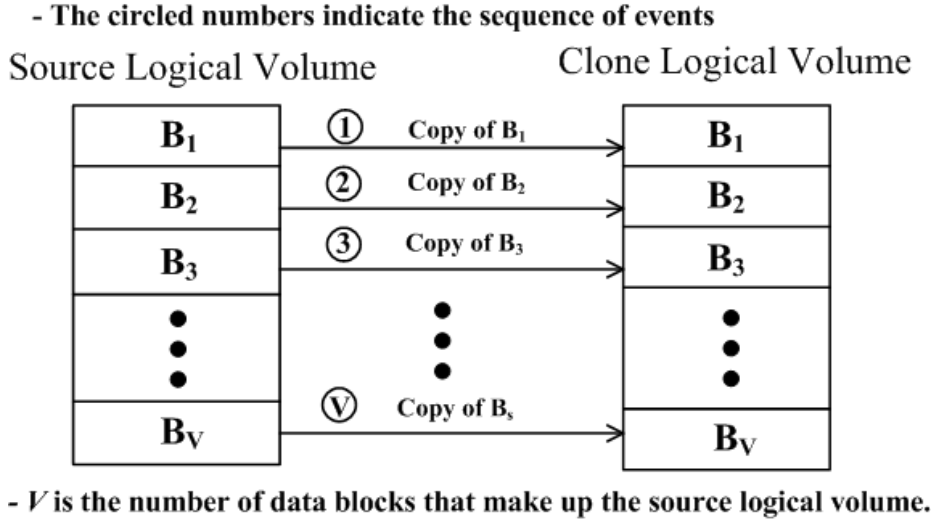

Fig. 6.3: Example of replication of a source volume

example why the clone logical volume takes up as much space as the source logical volume

The source logical volume (or source volume, for short) can be available during the cloning replication process. This means that the source volume can still be read or written to during the replication. Reading from the source volume does not disrupt the sequence of copying (cloning) the data blocks. Writing to the source volume after the replication process started is potentially disrupting to the cloning replication. To understand why a write can disrupt the cloning replication process it is important to remember that the clone volume is a point-in-time copy of the data in the source volume. Therefore, once the user decides to start replicating a source volume, the data at that particular point in time has to be preserved. Any further updates (writes) to the source volume should not be reflected in the replicated (cloned) volume. There are two possible results of a user write to the source volume during cloning replication: 1) if the user writes to a data block that has been already cloned then the write can proceed normally. There is no problem with the write since the original data block is already copied in the clone volume; 2) if the user writes to a data block that has not been cloned, then the incremental sequence of data block cloning has to be interrupted and that data block has to take priority and be cloned before the user write can proceed. The copy (replica) of a data block before the user write can be served is referred to as the copy-on-write (CoW) problem. Therefore, there is the possibility of generating copy-on-writes during the replication process by writing to the source volume.

Copy-on-writes cause a user write to have a high latency since the write has to wait for the copy (replication) of the data block before the write can be executed in the source volume. The CoW problem has been studied before [Navarro 11a] in the context of the logical volume *snapshot* replication. In that context, each snapshot replication is called a *snap*, and each snap causes a CoW to occur. That is why the terms *snap* and CoW are used interchangeably. A data block is said to be *snapped* if a user write to that data block caused a CoW to occur. An example of how a copy-on-write occurs during replication is shown in Fig. 6.4. The cloning replication is proceeding in sequential order and has copied data blocks B_1 and B_2. Before replicating the next data block, B_3, there is a user write to B_4. Now the replication process is forced to skip temporarily the replication of data block B_3 and clone data block B_4 to ensure it is a point-in-time copy. After the data block B_4 is cloned then the user write can proceed. After the replicated of B_4 the cloning replication process can resume where it left off, data

block B_3, and clone it. As expected, after cloning data block B_3, then data block B_4 is skipped since it is already cloned and the cloning replication proceeds sequentially with next un-cloned data block, B_5. This example shows that the logical volume cloning replication can occur interacting with logical volume snapshot replication.

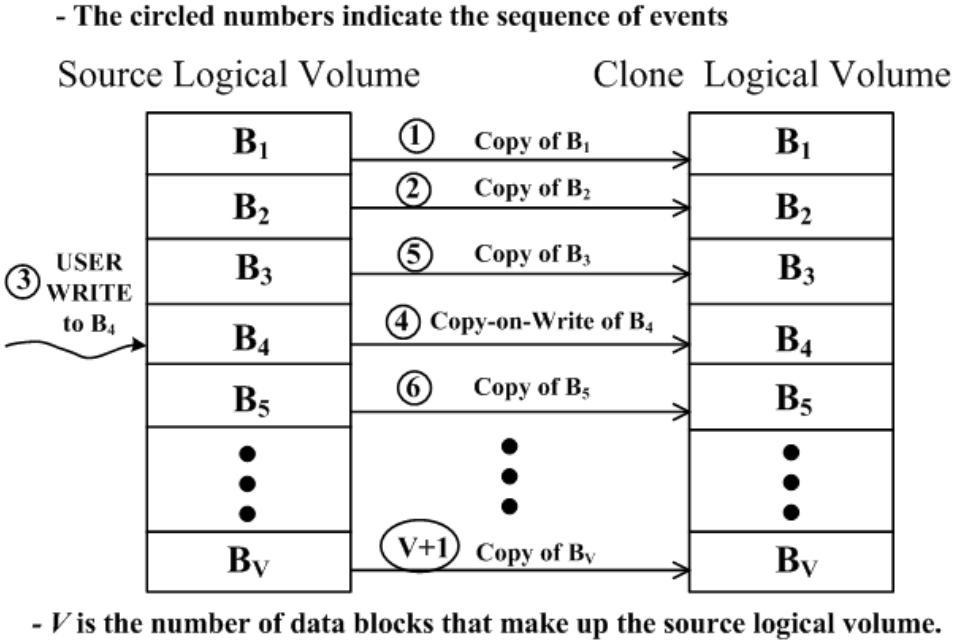

Fig. 6.4: Example of a CoW during the replication of a source volume

6.4 Queuing Description of the LV Cloning Replication

There are three queues directly involved in the Logical Volume (LV) cloning replication process: 1) the LV clone queue; 2) the LV user writes queue; and 3) LV the snapshot (CoW) queue. The LV user reads queue increases the utilization of the CPU and disks; but the user reads queue does not alter the cloning replication process in any way. Fig. 6.5 shows the queueing scheme considered for the analysis of the cloning process. The reason the LV user reads queue is included in the queueing scheme is because the type-2 fuzzy controller presented in this chapter takes the average of the LV user reads and writes as an input parameter.

The disk array controller sends the requests for cloning all the data blocks of an LV. The disk array keeps track at all times of which data blocks have been already been cloned and which ones are still pending. The cloning algorithm sends a request to clone a data block every z_c seconds. The z_c is the cloning interarrival time. Typical times for z_c can be in the

milliseconds or microseconds range. The z_c can also be considered a *think time* between requests for the LV Clone queue. The z_c is the parameter that controls the arrival rate of clone requests, χ_c, to the disks. Typical values for χ_c can be hundreds or thousands of IO requests per second (IO/s). The type-2 fuzzy controller will regulate the arrival rate of clone requests to the disk, χ_c, by regulating z_c.

The LV user writes queue sends the user writes to through the disk array controller onto the disk. For the purpose of this analysis, an Online Transaction Processing (OLTP) workload is considered. In OLTP workloads user writes and reads are randomly spread over the LV. The arrival rate of user writes, λ_w, on an LV in replication causes snaps to occur. The arrival rate of snaps caused by user writes was studied in the previous chapter and in [Navarro 11a] and shown in (5.13), where $p_{snap}(t)$ is the probability of a snapshot caused by the arrival rate of user writes, λ_w. This is the p_{snap} term studied in the previous in section 5.2.2 of this dissertation.

The rate arrival of user reads, λ_r, is considered as part of the estimation of the average latency of user requests (reads and writes) for the purpose of the fuzzy control.

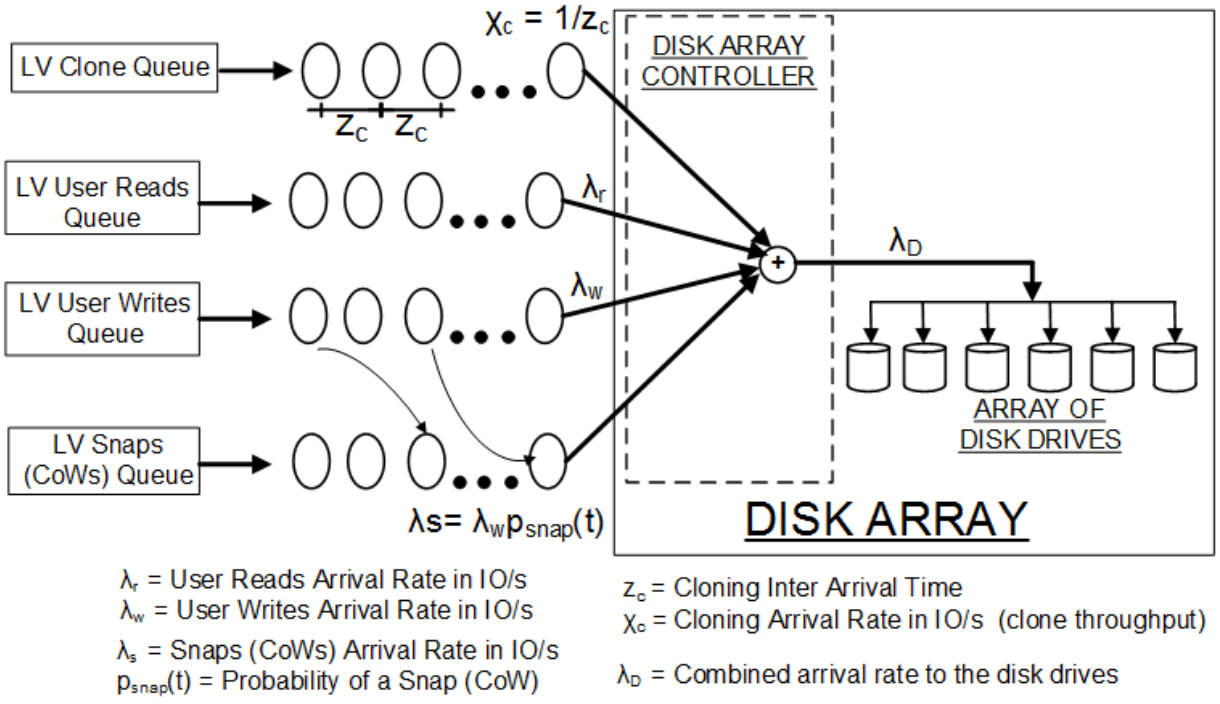

Fig. 6.5: Queueing scheme of LV Cloning with Snapshot

The combined arrival rate to the disks, λ_D, is the combination of the four arrival rates (χ_c, λ_w, , λ_s, λ_r) but considering the transformation of writes according to RAID level. For example, if the RAID level used for the data redundancy of the LV is RAID1, then the writes doubles because the data is written to separate disks.

6.5 Mathematical Description of the LV Cloning Replication

The cloning replication process is a deterministic process. The cloning replication algorithm keeps track of the replicated vs. the un-replicated data blocks. By doing this, the cloning replication always copies the un-replicated data blocks and never accesses again any already-replicated data block. This is an important distinction with respect to the snapshot replication process. The snapshot process is an on-demand process based on user writes to a logical volume. The user accesses to databases (SQL, Oracle) are typically randomly spread over a logical volume in what is referred to as the OLTP workload. This means that a user write may or may not cause a snap (CoW) to occur depending on whether the user write will write to an already-replicated or to a to-be-replicated data block. The understanding of the impact of randomly distributed user writes on logical volume replication through snapshots was studied in the previous chapter and in [Navarro 11a]. The formulas proposed in those two cited sources will be applied here for the purpose of showing the interaction of snapshot and cloning.

The estimation of the fraction of data blocks cloned at time t, $f_c(t)$, during the logical volume cloning replication, can be computed by:

$$f_c(t) = \frac{\chi_c}{B_v} t \tag{6.15}$$

where the B_v is the number of data blocks that make up the Logical volume. Fig. 6.6 shows a measured fraction of data blocks cloned during a logical volume cloning replication vs. the estimated fraction of data blocks using (6.15). It can be seen that the deterministic behavior of the cloning algorithm translates into a linear progress of the logical volume replication

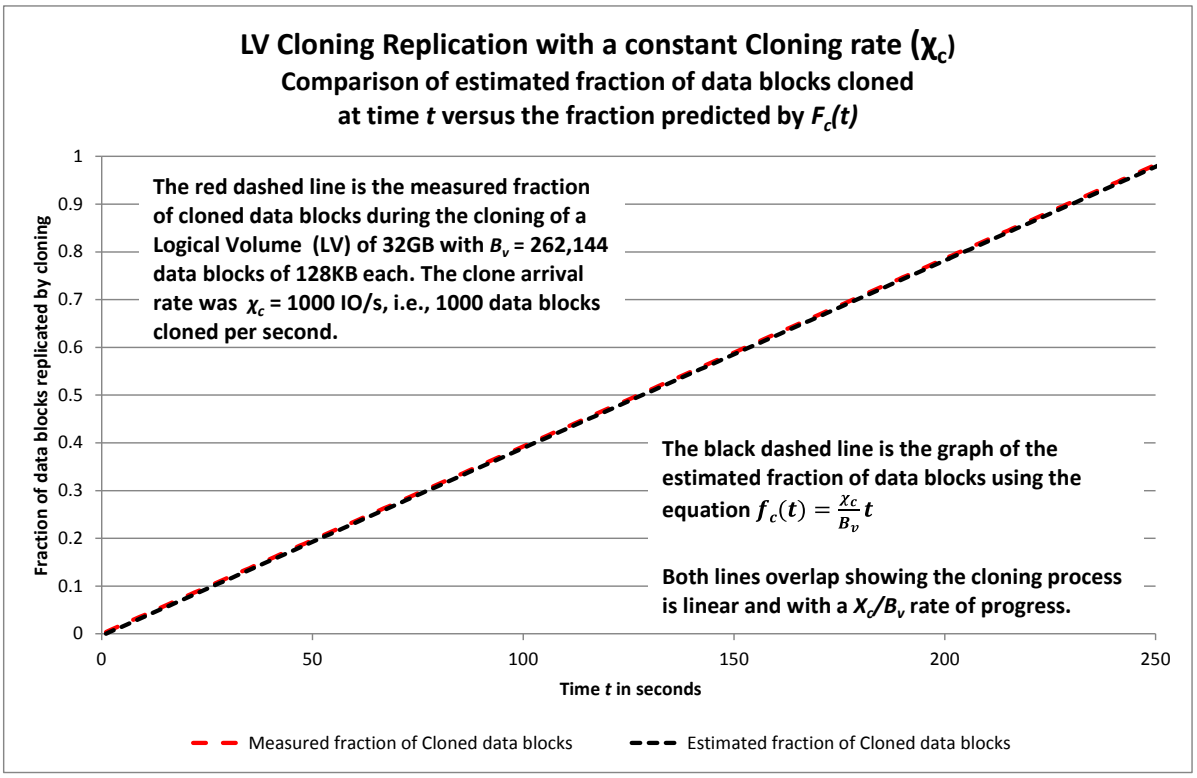

Fig. 6.6: Graph of the $f_c(t)$ equation predicting the fraction of cloned blocks

The estimation of the fraction of data blocks snapped at time t, $f_s(t)$, is a problem that was approached in the previous chapter and in [Navarro 11a].]. The main conclusion in [Navarro 11a] is that the snapshot replication under a randomly distributed user writes workload, like the on-line transaction processing (OLTP) workload, behaves in an exponential manner. The formula in presented in the previous chapter and in [Navarro 11a] is:

$$f_s(t) = 1 - e^{-\frac{\lambda_w}{B_v} t} \tag{6.16}$$

where λ_w is the user write arrival rate in IO/s and $f_s(t)$ is the estimated fraction of snapped data blocks. Fig. 6.7 shows the fraction of snapped data blocks, i.e., the fraction of data blocks replicated by snapshot vs. the estimated fraction of snapped data blocks using (6.16). In this example the logical volume is only being subjected to user writes so the replication shown in this figure is a snapshot replication, not a cloning replication.

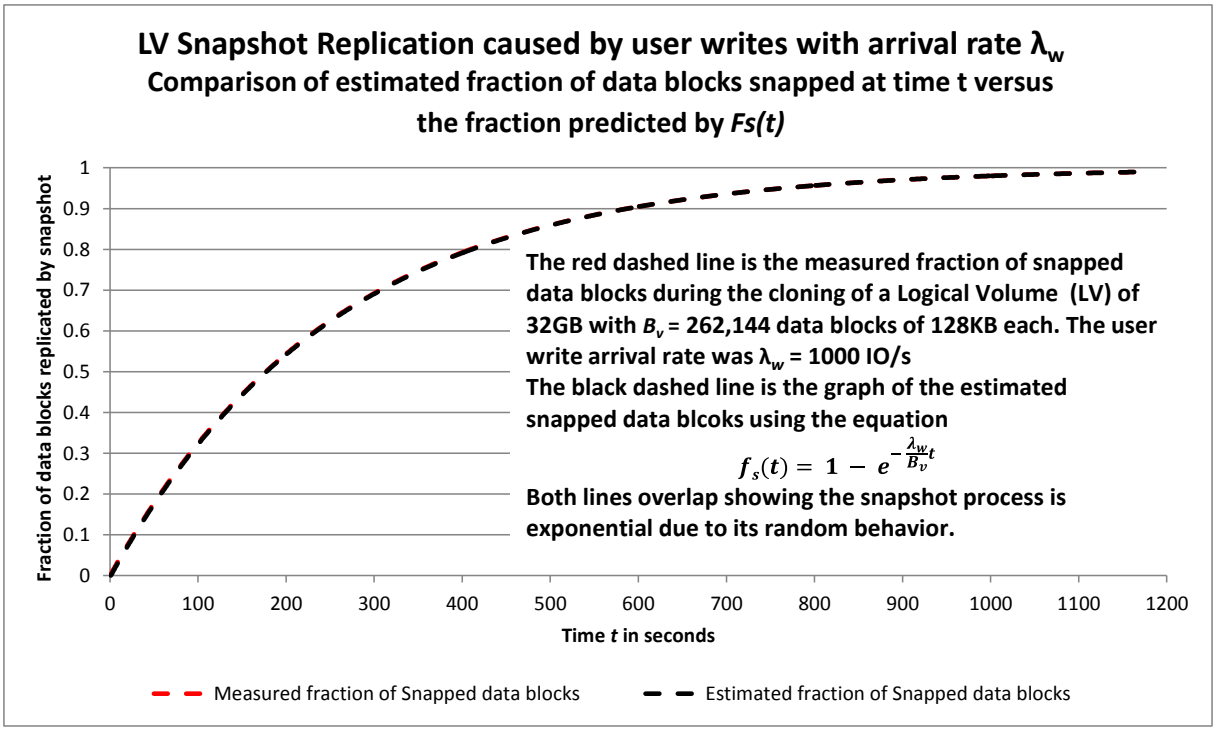

Fig. 6.7: Graph of the $f_s(t)$ equation predicting the fraction of snapped blocks

The equation for a logical volume combined replication, both cloning and snapshot, is derived here and shown to agree with a measured combined replication. We derive the equation for the fraction of replicated data blocks for a combined replication, using the equations and knowledge of the behavior of the both cloning and snapshot. The fraction of combined replicated data blocks, $f_r(t)$, is the sum of the cloned and the snapped data blocks

$$f_r(t) = f_c(t) + f_s(t) \tag{6.17}$$

The derivative of f_r is:

$$\frac{df_r}{dt} = \frac{df_c}{dt} + \frac{df_s}{dt} \tag{6.18}$$

The derivative of f_c is:

$$\frac{df_c}{dt} = \frac{\chi_c}{B_v} \tag{6.19}$$

We know that the probability of a snap is greater if there are more non-replicated data blocks. The fraction of non-replicated data blocks, $f_n(t)$, is:

$$f_n(t) = 1 - f_r(t) \tag{6.20}$$

We also know that the probability of a snap is an exponential function that depends on the ratio of the user write arrival rate, λ_w,, and the number of data blocks in a logical volume, B_v. The differential equation for the derivative of the fraction of data blocks snapped at time t, $f_s(t)$, is then:

$$\frac{df_s}{dt} = \frac{\lambda_w}{B_v} f_n = \frac{\lambda_w}{B_v}(1 - f_r) \tag{6.21}$$

The differential equation for f_r is:

$$\frac{df_r}{dt} = \frac{\chi_c}{B_v} + \frac{\lambda_w}{B_v}(1 - f_r) \tag{6.22}$$

And the fraction of replicated data blocks in a combined replication, $f_r(t)$ at time t is:

$$f_r(t) = (1 + \frac{\chi_c}{\lambda_w})\left(1 - e^{-\frac{\lambda_w}{B_v}t}\right) \tag{6.23}$$

The equation is compared against the measured fraction of replicated data blocks in a combined (cloning and snapshot) replication of a logical volume as shown in Fig. 6.8. The graph shows that (6.23) estimates $f_r(t)$ very accurately.

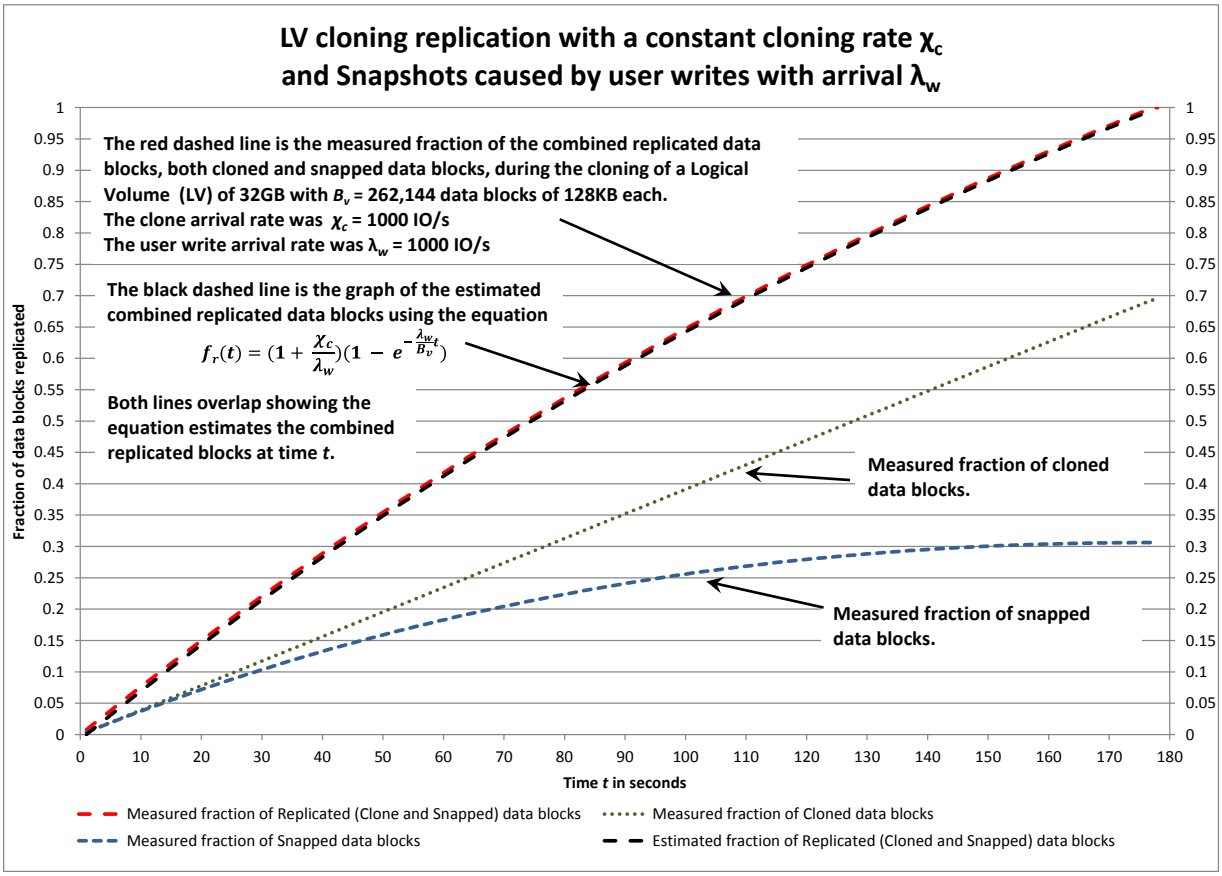

Fig. 6.8: Graph of the $f_r(t)$ equation predicting the fraction of replicated blocks

6.6 Local LV Cloning Replication Type-2 Fuzzy Logic Controller

6.6.1 Purpose of the LV Cloning T2 Fuzzy Controller

The primary goal of this controller is to regulate the rate at which the cloning process occurs so the latency of customer reads and writes is at or below a target latency. If the primary goal cannot be accomplished, then the secondary goal is to minimize the impact to the latency of customer reads and writes. The way the controller achieves the goal of regulating the cloning process is by adjusting the cloning interarrival value, z_c, therefore, regulating the time between each clone request to the . Every time sample iteration, t_i, the controller estimates the z_c to be used in the next time sample, t_{i+1}.

The logical volume cloning replication Type-2 (T2) fuzzy controller is a Takagi-Sugeno Type-2 fuzzy-logic based controller. The T2 fuzzy PI controllers are more robust than

their Type-1 counterparts [Wu 2010a]. The T2 fuzzy PI controllers can cope better with disturbances, uncertainties and eliminate oscillations better than their T1 counterparts.

6.6.2 Description of the LV Cloning T2 Fuzzy Controller

The block diagram of the logical volume cloning type-2 fuzzy controller is illustrated in Fig. 6.9. From a control standpoint, the disk array is the controlled system. The controlled system has one input: λ_D, which is the combined arrival rate of user writes, λ_w, user reads, λ_r, snapshots, λ_{cow}, and cloning arrival rate χ_c. The combined arrival rate on the disk array, λ_D, is:

$$\lambda_D = \lambda_r + \alpha_{RL}\lambda_w + \lambda_{cow} + \chi_c \tag{6.24}$$

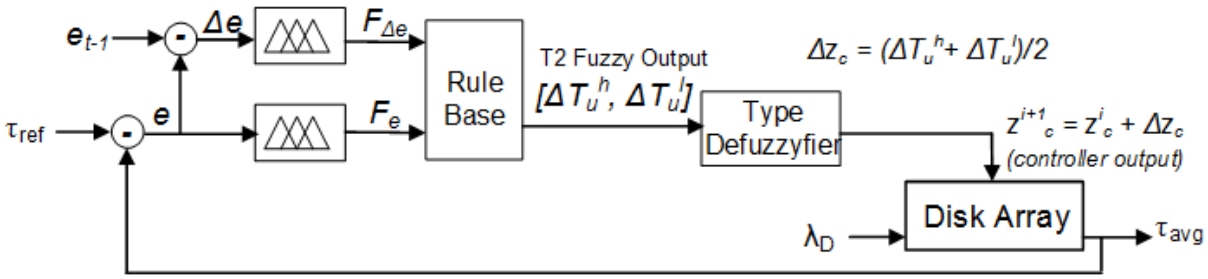

Fig. 6.9: Cloning type-2 fuzzy controller

The cloning arrival rate, χ_c, depends on the cloning interarrival time, z_c:

$$\chi_c = \frac{1}{z_c} \tag{6.25}$$

The number of data blocks writes needed for a snapshot (CoW) is dependent on the RAID level of the snapshot volume. The number of data block writes is defined by the α_{RL} factor. For RAID1 the $\alpha_{RL} = 2$, which is the number of writes needed for each data write. This was explained in section 5.2.3.

The total extra arrival rate on the disk array generated by the copy-on-writes, λ_{cow}, is already shown in (5.17) but show here again:

$$\lambda_{cow} = p_{snap}\lambda_w(1 + \alpha_{RL}) \tag{6.26}$$

The output of the system to be controlled (disk array) is the average latency experienced by the user accesses (reads or writes), τ_{avg}.

The logical volume cloning type-2 fuzzy controller makes use of a reference variable: the reference latency, τ_{ref}. The reference latency represents the maximum acceptable average latency during the cloning process. The maximum latency used in this section was 30ms, same as in chapter 5.

The output τ_{avg} is compared with the reference latency τ_{ref} to compute the control error, e:

$$e(t_i) = \tau_{avg}(t_i) - \tau_{ref} \tag{6.27}$$

The change in the control error, Δe, is also computed:

$$\Delta e(t_i) = e(t_i) - e(t_{i-1}) \tag{6.28}$$

The cloning rate is kept within limits by setting two variables also used by the logical volume cloning type-2 fuzzy controller: 1) the minimum cloning interarrival time z_c^{min}, which determines the maximum cloning arrival rate, i.e., the cloning throughput, χ_c^{max}; and 2) the maximum cloning interarrival time, z_c^{max}, which determines the minimum cloning arrival rate (cloning throughput) χ_c^{min}.

In order to control the outputs, they have to be periodically monitored every t_m seconds. The decision on how often to monitor can be based on the maximum acceptable latency and the performance of the disk array controller. The sampling of the outputs is performed at intervals of time t_m. Each sample is denoted by (t_i), where i is the *i-th* sample of the output that occurred at a time t_i, as in:

$$t_i = it_m \qquad \text{where } i = 0,1,2,\ldots \tag{6.29}$$

6.6.3 LV CLONING CONTROLLER FUZZIFICATION

This section shows how the step 1 of the typical procedure performed by an IT2 FLC as shown in section 6.2, is implemented for the logical volume cloning T2 fuzzy controller. The goal of the fuzzification is to map the control error *e* into a fuzzy value F_e and the change in the control error, Δe, into a fuzzy value $F_{\Delta e}$. This stage corresponds to the blocks shown in Fig. 6.9 with the symbols F_e and $F_{\Delta e}$. Both fuzzy values, F_e and $F_{\Delta e}$, can be mapped into one of three T2 fuzzy descriptors: Zero (ZE), Positive Error (PE), and Negative Error (NE). The first step is to normalize *e* and Δe with respect to the reference latency, τ_{ref}.

$$e_n = \frac{e}{\tau_{ref}} \tag{6.30}$$

$$\Delta e_n = \frac{\Delta e}{\tau_{ref}} \tag{6.31}$$

The normalized control error, e_n, and normalized change in the control error, Δe_n, are limited in their values to make the T2 fuzzification possible to the interval [-1,1]:

$$e_n \in [-1,1] \quad and \quad \Delta e_n \in [-1,1] \tag{6.32}$$

The mapping for the e_n and Δe_n values into the Z_{en} and to $Z_{\Delta en}$ fuzzy values is done by defining the following equations:

$$UMF_{Zen}(e_n) = \overline{\mu}_{Zen} = \begin{cases} -e_n + \left(1 + \frac{DOU}{2}\right) & if \ e_n \geq 0 \\ e_n + \left(1 + \frac{DOU}{2}\right) & if \ e_n < 0 \end{cases} \tag{6.33}$$

$$LMF_{Zen}(e_n) = \underline{\mu}_{Zen} = \begin{cases} -e_n + \left(1 - \frac{DOU}{2}\right) & if \ e_n \geq 0 \\ e_n + \left(1 - \frac{DOU}{2}\right) & if \ e_n < 0 \end{cases} \tag{6.34}$$

The intervals for UMF_{Zen} and LMF_{Zen} are:

$$UMF_{Zen} \in [-1,1] \quad and \quad LMF_{Zen} \in [-1,1] \tag{6.35}$$

Equations (6.33) and (6.34) map the e_n value to the T2 fuzzy value Z_{en}. The equations for mapping Δe_n to $Z_{\Delta en}$ are the same as (6.33) and (6.34) just with the Δe_n variable instead of the e_n as the input variable. The intervals for the $UMF_{Z\Delta en}$ and $LMF_{Z\Delta en}$ are the same as the UMF_{Zen} and LMF_{Zen} shown in (6.35).

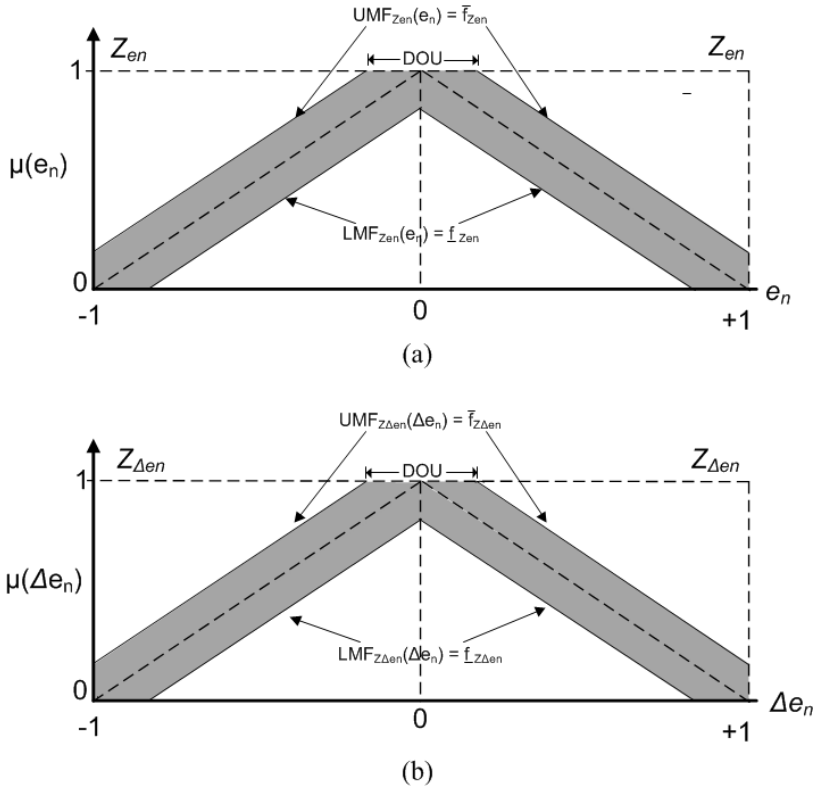

Fig. 6.10: T2 Fuzzy values Z_{en} and $Z_{\Delta en}$

The mapping for the e_n and Δe_n values into the N_{en} and to $N_{\Delta en}$ fuzzy values is done by defining the following equations:

$$UMF_{Nen}(e_n) = \overline{\mu}_{Nen} = -e_n + \frac{DOU}{2} \tag{6.36}$$

$$LMF_{Nen}(e_n) = \underline{\mu}_{Nen} = -e_n - \frac{DOU}{2} \tag{6.37}$$

The intervals for UMF_{Nen} and LMF_{Nen} are:

$$UMF_{Nen} \in [-1,1] \quad and \quad LMF_{Nen} \in [-1,1] \tag{6.38}$$

Equations (6.36) and (6.37) map the e_n value to the T2 fuzzy value N_{en}. The equations for mapping Δe_n to $N_{\Delta en}$ are the same as (6.36) and (6.37) just with the Δe_n variable instead of the e_n as the input variable. The intervals for the $UMF_{N\Delta en}$ and $LMF_{N\Delta en}$ are the same as the UMF_{Nen} and LMF_{Nen} shown in equation (6.38).

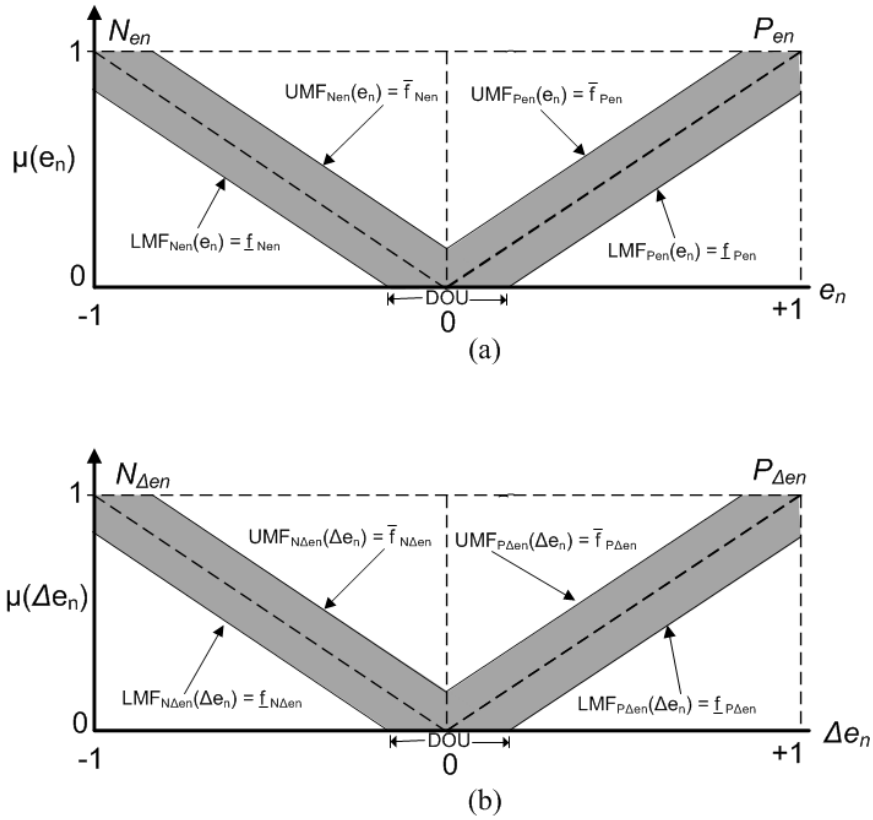

Fig. 6.11: T2 Fuzzy values (a) N_{en}, P_{en} and (b) $N_{\Delta en}$. $P_{\Delta en}$

The mapping for the e_n and Δe_n values into the P_{en} and to $P_{\Delta en}$ fuzzy values is done by defining the following equations:

$$UMF_{Pen}(e_n) = \overline{\mu}_{Pen} = e_n + \frac{DOU}{2} \tag{6.39}$$

$$LMF_{Pen}(e_n) = \underline{\mu}_{Pen} = e_n - \frac{DOU}{2} \quad (6.40)$$

The intervals for UMF_{Pen} and LMF_{Pen} are:

$$UMF_{Pen} \in [-1,1] \quad and \quad LMF_{Pen} \in [-1,1] \quad (6.41)$$

Equations (6.39) and (6.40) map the e_n value to the T2 fuzzy value P_{en}. The equations for mapping Δe_n to $P_{\Delta en}$ are the same as (6.39) and (6.40) just with the Δe_n variable instead of the e_n as the input variable. The intervals for the $UMF_{P\Delta en}$ and $LMF_{P\Delta en}$ are the same as the UMF_{Pen} and LMF_{Pen} shown in equation (6.41).

6.6.4 Rule Base to obtain T_U

This section shows how the step 2 of the typical procedure performed by an IT2 FLC as shown in section 6.2, is implemented for the logical volume cloning T2 fuzzy controller. The rule base applies the logic to determine how to alter the cloning interarrival time, z_c at every time iteration t_i. In order to alter z_c incrementally, the change in z_c has to be in some range that modifies z_c in a way that does not change the cloning arrival rate, χ_c erratically (up and down) and causes the controller to oscillate. The *throttling unit*, T_u, is a quantity of time used by this controller as a unit of change of the cloning interarrival time, z_c. The outputs of the rules, i.e., the subsequent sets, are expressed in terms of the throttling units.

The rule base can now be built based on the following heuristic criteria. The first criterion is that if the user request latency is high, then the control error, e, is fuzzy positive error, PE, and the controller needs to reduce the cloning rate. Therefore, the cloning interarrival time z_c is increased. The second criterion is that if the user request latency is low, then the controller can increase the cloning rate. Therefore, the cloning interarrival time z_c is reduced. Based on those two heuristics criteria, the rules are developed of the form:

$$if\ e \in F_e\ and\ \Delta e \in F_{\Delta e}\ then\ \ [\Delta T_u^h\ , \Delta T_u^l] \quad (6.42)$$

where F_e and $F_{\Delta e}$ can take the fuzzy values shown in section 6.6.3:

$$F_e \in \{N_{en}, Z_{en}, P_{en}\} \tag{6.43}$$

$$F_{\Delta en} \in \{N_{\Delta en}, Z_{\Delta en}, P_{\Delta en}\} \tag{6.44}$$

And the consequent sets are of the form [ΔT_u, ΔT_u] where Δ is a multiplier of the throttling unit T_u as shown in Table 6.1.

Table 6.1: Rule base for LV Cloning Type-2 Fuzzy Controller

Rule Number	Rule Input Variables		Rule Output	Comments
	e	Δe	T_u Range	
R_1	Ne_n	$N\Delta e_n$	[$-4T_u$, $-2T_u$]	Reduce z_c, increase χ_c heavily
R_2	Ne_n	$Z\Delta e_n$	[$-2T_u$, $0T_u$]	Reduce z_c, increase χ_c lightly
R_3	Ne_n	$P\Delta e_n$	[$-3T_u$, $-1T_u$]	Reduce z_c, increase χ_c
R_4	Ze_n	$N\Delta e_n$	[$0T_u$, $2T_u$]	Increase z_c, reduce χ_c lightly
R_5	Ze_n	$Z\Delta e_n$	[$-T_u$, $+T_u$]	Keep z_c, thus, χ_c
R_6	Ze_n	$P\Delta e_n$	[$-2T_u$, $0T_u$]	Reduce z_c, increase χ_c lightly
R_7	Pe_n	$N\Delta e_n$	[$+T_u$, $3T_u$]	Increase z_c, reduce χ_c
R_8	Pe_n	$Z\Delta e_n$	[$0T_u$, $2T_u$]	Increase z_c, reduce χ_c lightly
R_9	Pe_n	$P\Delta e_n$	[$2T_u$, $4T_u$]	Increase z_c, reduce χ_c heavily

6.6.5 TYPE REDUCTION (DEFUZZIFICATION)

The steps 3 and 4 of the typical procedure performed by an IT2 FLC as shown in section 6.2, are implemented for the LV cloning T2 fuzzy controller are implemented using the Karnik-Mendel (KM) algorithms. References [Mendel 12][Wu 12a] are recommended to learn about the KM algorithm. The output of the type reduction is the pair:

$$[\ \Delta z_c^l\ ,\Delta z_c^r\ ] \tag{6.45}$$

where Δz_c^l and Δz_c^r are the left and right values of the Type 1 output fuzzy set produced by the KM algorithm.

6.6.6 CRISP DELTA OF THE CLONING INTERARRIVAL TIME

The step 5 of the typical procedure performed by an IT2 FLC as shown in section 6.2, is the calculation of the crisp value of the change in the cloning interarrival time. The calculation is the middle point of the two values in the T1 fuzzy set produced in the previous step:

$$\Delta z_c = \frac{\Delta z_c^l + \Delta z_c^r}{2} \tag{6.46}$$

where Δz_c is the delta to be added to the current z_c for the next time sample of the controller:

$$z_c(t_{i+1}) = z_c(t_i) + \Delta z_c(t_i) \tag{6.47}$$

where $\Delta z_c(t_i)$ is the delta of the cloning interarrival time obtained by (6.45); $z_c(t_i)$ is the current cloning interarrival time and $z_c(t_{i+1})$ is the cloning interarrival time computed to be used in the next time sample by the logical volume cloning T2 fuzzy controller.

This is the final step of the controlling process and it is repeated at the next time iteration starting from the steps shown in section 6.6.3.

6.7 EXPERIMENTAL RESULTS

The type-2 fuzzy logical volume replication controller was tested with a setup that consisted of an HP 7640 Itanium workstation with 48GiB of memory and with RedHat Linux 6.2 installed. The disk setup consisted of 118 BF1465A477 15K RPM disks. The logical volume replication and the type-2 fuzzy control was implemented in C language and compiled with GCC 4.4.6. The implementation was executed as a parent process in the user space and not as a part of the kernel. The parent process performed the following functions: 1) generated (forked) a process that acted as foreground user request process generator for 8KiB user reads and writes; 2) generated a process that acted as the background logical volume replication by generating a process for 256KiB data block that had to be cloned; 3) kept track of the data blocks replicated and the data blocks that required a replication by doing a copy-on-write. The logical volume data block table was in shared memory so it was visible to all processes; 4)

monitored the latency of the user requests and 5) implemented the IT2 FLC logic of the LV cloning controller.

A comparison of an LV cloning replication process with and without the T2 fuzzy control was run. For the comparison a RAID1 128GiB source volume was used. The source volume was comprised of data blocks of 256KiB in size and laid out in an even fashion over all the 118 disks.

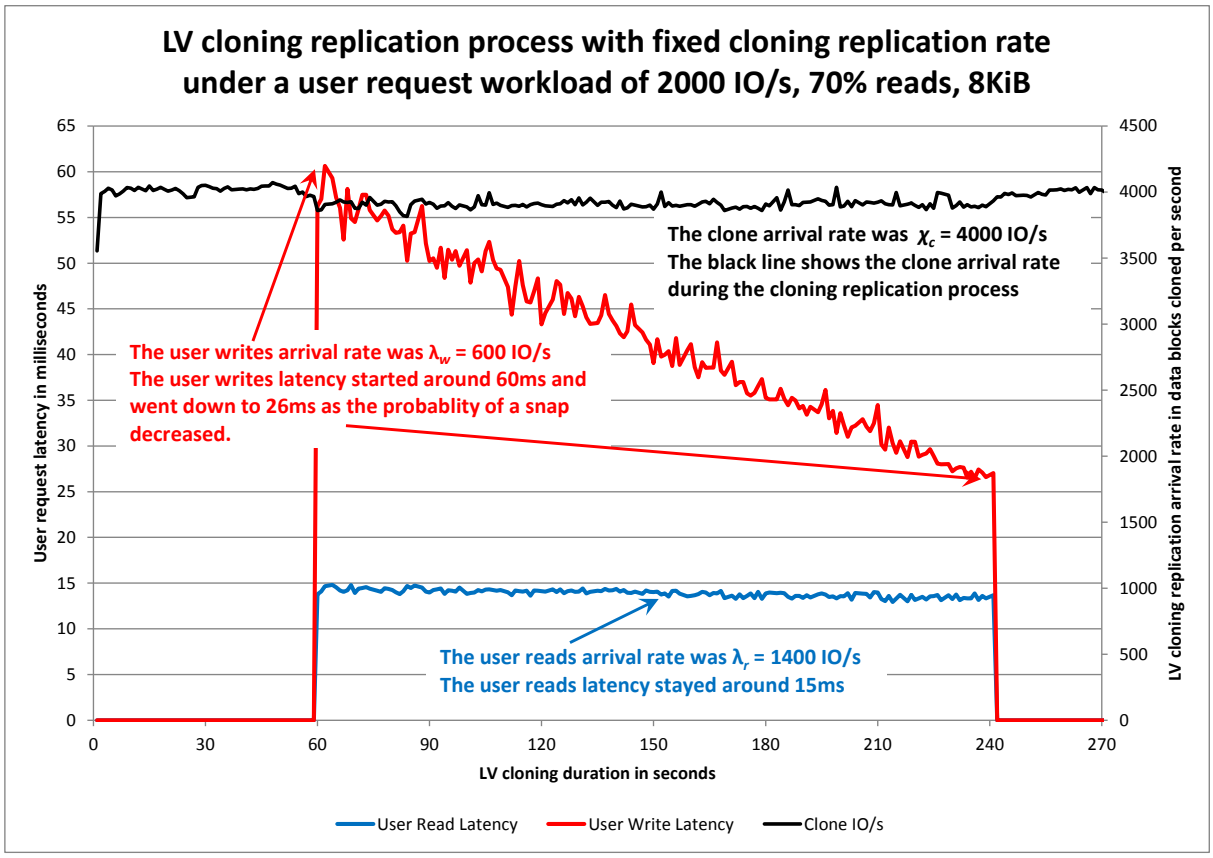

Fig. 6.12: Cloning of an LV with no fuzzy control

First, the LV cloning process is shown with no fuzzy control in Fig. 6.12. The left vertical axis shows the latency values in milliseconds. The right vertical axis shows the IO/s values. A cloning process is replicating the 128GiB at a constant rate of χ_c = 4,000 IO/s as shown by the black line. After 60 seconds into the replication process, a user workload of 2,000 IO/s, 70% reads, (1,400 IO/s for reads and 600 for IO/s writes) was applied during 3 minutes. The blue line shows the latency of user reads during the 3 minutes. It can be seen that the read latency (blue line) is around the 15ms range. The user reads do not show a significant change during the duration of the user workload (3 minutes). The user writes, on the other hand, show how significantly can be affected by the cloning process. At first, the

user writes show a latency of 60ms. This high latency was caused by the combined effects of the copy-on-writes generated by the user writes themselves, and the background cloning replication in progress. As the probability of generating copy-on-writes lowers, then the user writes show less impact on their latency. At the end of the 3 minute run, the user write latency is around 25ms.

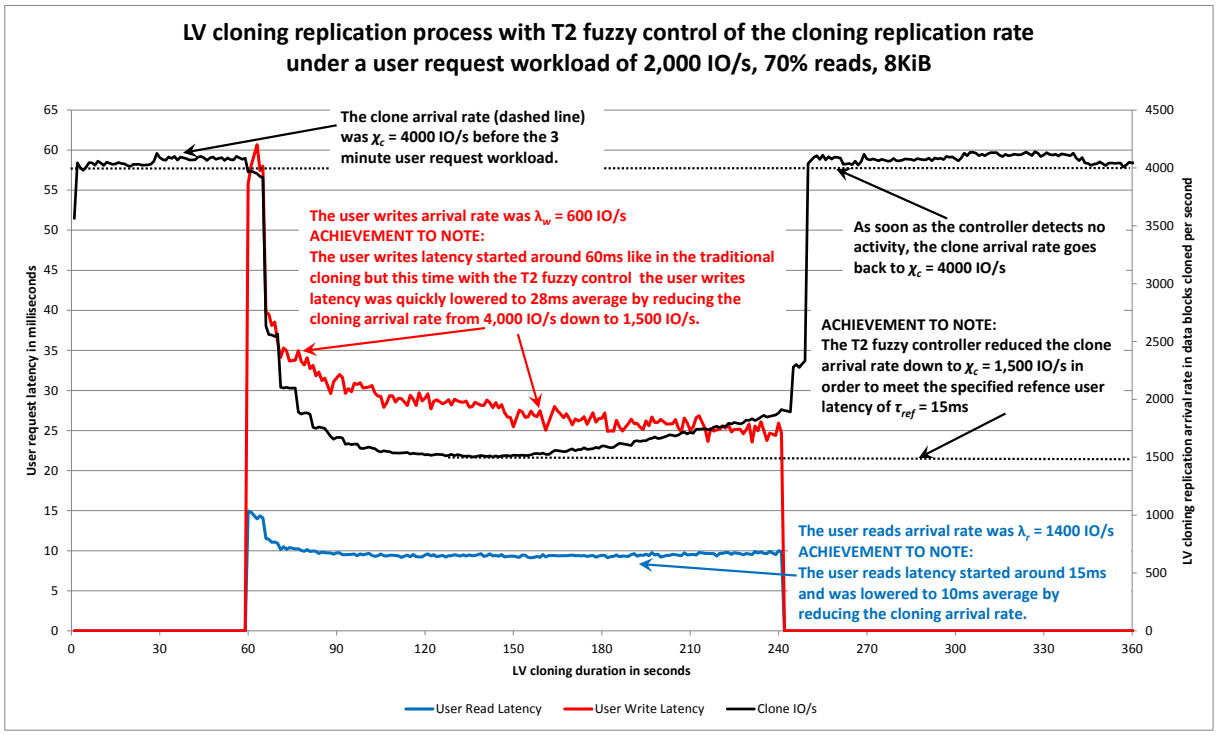

Fig. 6.13: Cloning of an LV with T2 fuzzy control

Second, The LV cloning process was run with the T2 fuzzy controller. The reference latency used was τ_{ref} = 15ms; a throttling unit, T_u = 200us and a delta of uncertainty, DOU = 0.2. Fig. 6.13 shows how the T2 fuzzy controller could achieve a reduction in the user request latency for both writes and reads. The controller is cloning in the background at a rate of χ_c = 4,000 IO/s when 60 seconds into the cloning a user workload of 2,000 IO/s, 70% reads, (1,400 IO/s for reads and 600 for IO/s writes) was applied during 3 minutes just like in the case with no fuzzy control. But in this case the T2 fuzzy controller detects an error because the user write latency is in the 60 millisecond range and responds by increasing the cloning interarrival time which in turns reduces the cloning rate from χ_c = 4,000 IO/s down to χ_c = 1,500 IO/s. This brings down the user write latency down from 60 to 25ms, which in conjunction with the reduction in the user read latency from 15 to 10ms, brings the user average (reads and writes)

latency down to the reference latency of τ_{ref} = 15ms. The T2 fuzzy controller achieved the purpose of reducing the cloning rate so the reference latency could be achieved.

6.7 CONCLUSIONS

The Type-2 LV cloning fuzzy controller accomplished the goal of reducing the impact on the user reads and writes latency caused by an LV cloning background process in a disk array. The improvements in latency the Type-2 LV cloning fuzzy controller delivers show how fuzzy logic can be applied to improve the performability of data backup management for disk arrays. The Type-2 LV cloning fuzzy controller can be used to control a disk array with complex components such as disks, for which we don't have any knowledge of their internal logic and are hard to model mathematically. Another contribution of this chapter is the equation (6.23) that predicts the fraction of replicated data blocks in a combined (clone and snapshot) replication.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

Chapter 3 presented a numerical method in the form of an extensible algorithm that can be used to estimate the reliability and performability of a disk array. This method can be used to achieve high performability based on the reliability of a RAID group.

Chapters 4, 5 and 6 present improvements in disk array performability based on fuzzy control schemes.

Chapter 4 showed that fuzzy logic can be applied to improve the sparing process in disk arrays. The patented fuzzy-controlled sparing process outperformed the traditional QSV sparing process by finishing in half the time and without impacting the user request latency.

Chapter 5 showed the benefit of the proposed snapshot fuzzy controller, which is to avoid the high latency peak at the beginning of a snapshot process. Chapter 5 also presented a Markov Model and equations for the snapshot process. These equations can provide a guide for the snapshot behavior even for different disks speeds and disk arrays if the snapshot process is the traditional copy-on-write.

Chapter 6 showed the benefit of a cloning fuzzy controller, which is to reduce the impact of the background cloning process on the user request latency but ensuring the cloning occurs in the background at the maximum possible rate. Chapter 6 also presented an equation for the combined cloning and snapshot of a logical volume.

7.2 FUTURE WORK

The first area of proposed future work is the development of probabilistic models for the performability evaluation of the background-jobs-based performability for sparing, snapshot and cloning. These models can leverage off the probabilistic equations already presented for snapshot and cloning in chapters 5 and 6. The second area of future work is the stability analysis of the fuzzy controllers presented in chapters 4, 5 and 6.

REFERENCES

[Barnett 98a] S. A. Barnett, G. J. Anido, "Performability of disk-array-based video servers", Multimedia Systems, 1998.

[Bolch 06a] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi, "Queueuing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications", John Wiley & Sons, 2006.

[Burkhard 93a] W.A. Burkhard, J. Menon; "Disk array storage system reliability", Fault-Tolerant Computing, 1993. FTCS-23.

[Catania 95a] V. Catania, A. Puliafito, S. Riccobene, L. Vita, "Design and performance analysis of a disk array system", IEEE Transactions on Computers, 1995.

[EMC 05a] EMC Corporation, "Ensuring Data Availability with TimeFinder Family Local Replicas", http://www.emc.com/collateral/software /white-papers/h1618-ensuring-data.pdf , 2005.

[Ganger 94a] G.R Ganger; B.L. Worthington; R.Y. Hou; Y.N. Patt, "Disk arrays: high-performance, high-reliability storage subsystems", Computer, Volume 27, Issue 3, March 1994.

[Garimella 06a] N. Garimella, "Understanding and exploiting snapshot technology for data protection, Part 1: Snapshot technology overview", http://www-128.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html, IBM Tivoli Software Group, 2006.

[Gartner 13a] J. Rivera, R. van der Meulen; "Garnet Says Worlwide External Controller-Based Disk Storage Market Grew 1.9 Percent in Fourth Quarter of 2012", http://www.gartner.com/newsroom/id/2380815, March 21, 2013.

[Hagras 07a] H. Hagras, "Type-2 FLCs: A New Generation of Fuzzy Controllers", IEEE Computational Inteligence Magazine, Feb. 2007.

[Hanss 10a] M. Hanss, "Applied Fuzzy Arithmetic; An Introduction with Engineering Applications", Springer, 2010.

[Haverkort 01a] B. R. Haverkort, R. Marie, G. Rubino, K. Trivedi, "Performability Modeling: Technique and Tools", John Wiley & Sons, Ltd, 2001.

[Hanss 05a] M. Hanss, "Applied Fuzzy Arithmetic", Springer-Verlag, 2005.

[Hou 93a] R.Y. Hou, J. Menon, Y.N. Patt, "Balancing I/O response time and disk rebuild time in a RAID5 disk array", HICSS, 1993.

[HP 06a] Hewlett-Packard, “HP StorageWorks 1000/1500 Modular Smart Array Command Line Interface”, May 2006, http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00683579/c00683579.pdf

[HP 08a] Hewlett-Packard, “HP StoreageWorks Business Copy EVA”, http://h18000.www1.hp.com/products/quickspecs/11616_div/11616_div.pdf, 2008.

[Islam 93a] S.M. Rezaul Islam, “Performability Analysis of Disk Arrays”, IEEE Circuits and Systems, 1993.

[Karnik 99a] N. Karnik, J. M. Mendel, Q. Liang, “Type-2 Fuzzy Logic Systems”, IEEE Transactions on Fuzzy Systems, Vol. 7, No. 6, Dec. 1999.

[Kleinrock 75a] L. Kleinrock, “Queuing Systems, Vol. 1: Theory”, John Wiley & Sons, 1975.

[Klir 95a] G.J. Klir, B. Yuan, “Fuzzy Sets and Fuzzy Logic – Theory and Applications”, Prentice Hall, 1995.

[Lee 93a] Edward K. Lee, Randy H. Katz, “An analytic performance model of disk arrays”, ACM SIGMETRICS '93.

[Mamdani 75a] E. H. Mamdami, S. Assilian, “An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller”, International Journal of Man Machine Studies, 7:1-13, 1975.

[Medhi 03a] J. Medhi, “Stochastic Models in Queueing Theory”, Academic Press, 2003.

[Mendel 10a] J. M. Mendel, “A Quantitive Comparison of Interval Type-2 and Type-1 Fuzzy Logic Systems: First Results”, Proceedings of World Congress on Computational Intelligence, Barcelona, Spain, July 2010.

[Mendel 12a] J. M. Mendel, “On KM algorithms for Solving Type-2 Fuzzy Sets Problens”, IEEE Transactions on Fuzzy Logic, 2012.

[Menon 93a] R.Y. Hou, J. Menon, Y.N. Patt, “Balancing I/O response time and disk rebuild time in a RAID5 disk array”, Proc. Hawaii Int’l Conf. on System Science, Jan. 1993.

[Menon 94a] J. Menon, A. Thomasian, “Performance Analysis of RAID5 Disk Arrays with a Vacationing Server Model for Rebuild Mode Operation”, Proc. IEEE 10th International Conference on Data Engineering, 1994.

[Mesquite 07a] CSIM19, Mesquite Software, www.mesquite.com

[Meyer 78a] J. F. Meyer, “On Evaluating the Performability of Degradable Computing Systems”, Proceedings of FTCS-8 pp. 44-49 IEEE, 1978.

[Meyer 80a] J. F. Meyer, “Performability Evaluation of the SIFT Computer”, IEEE Transaction on Computers, vol C-29, pp. 501-509, June 1980.

[Michels 06a] K. Michels, F. Klawonn, R. Kruse, A. Nurnbeger, “Fuzzy Control: Fundamentals, Stability and Design of Fuzzy Controllers”, Springer-Verlag, 2006.

[Microsoft 04a] Microsoft, “Disk Subsystem Performance Analysis for Windows”, http://download.microsoft.com/download/e/b/a/eba1050f-a31d-436b-9281-92cdfeae4b45/subsys_perf.doc, March 2004.

[Microsoft 06a] Microsoft, “Disk Bottleneck Detected”, 2006, http://technet.microsoft.com/en-us/library/29f01985-7b44-47cb-96f7-d7c92fd8e867.aspx

[Microsoft 06b] Microsoft, “Calculate Your Server Size”, 2006, http://technet.microsoft.com/en-us/library/bb124226.aspx

[Microsoft 07a] Microsoft, “Exchange Server 2003”, http://technet.microsoft.com/en-us/library/bb123872.aspx

[Microsoft 07b] Microsoft, “How to Calculate Your Disk I/O Requirements”, http://technet.microsoft.com/en-us/library/bb125019.aspx

[Muntz 90a] R. R. Muntz, John C.S. Lui, “Performance Analysis of Disk Arrays under failure”, 16th VLDB Conference, 1990.

[Navarro 06a] G. Navarro, M. Manic, “Fuzzy Performability Analysis of Disk Arrays”, IEEE ISIE, 2006.

[Navarro 07a] G. Navarro, M. Manic, “Predictive E-Mail Server Performability Analysis Based on Fuzzy Arithmetic”, IJCNN, 2007.

[Navarro 07b] G. Navarro, M. Manic, “Fuzzy Control of Sparing in Disk Arrays”, 12th IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2007.

[Navarro 07c] G. Navarro, M. Manic, “NFuSA – Neuro-Fuzzy Algorithm for Sparing in RAID Systems”, IEEE IECON, 2007.

[Navarro 11a] G. Navarro, M. Manic, “FuSnap: Fuzzy Control of Logical Volume Snapshot Replication for Disk Arrays’, IEEE Transactions on Industrial Electronics, Vol. 58, No. 9, Sep. 2011.

[Papadimitriou 94a] C.H. Papadimitriou. J. H. Tsitsiklis, “The Complexity of Optimal Queuing Network Control”, SCTC, 1994.

[Patterson 88a] D. A. Patterson, G. Gibson, R. H. Katz, “A case for redundant arrays of inexpensive disks (RAID)”, ACM SIGMOD, 1988.

[Patterson 94a] D. A Patterson, P. M. Chen, E. K. Lee, G. A. Gibson, R. H Katz, “RAID: High-Performance, Reliable Secondary Storage”, ACM Computing Surveys, 1994.

[Patterson 07a] Patterson David A., Hennessy John L., “Computer Architecture, a quantitative approach”, Morgan Kaufman Publishers, 2003.

[Phillips 99a] Y. A. Phillips, R. Zhang, “Fuzzy Service Control of Queuing Systems”, IEEE Transactions on Systems, Man and Cybernetics, Vol. 29, No. 4, August 1999.

[Preston 02a] W. C. Preston, “Using SANs and NAS”, O’Reilly, 2002.

[Reddy 91a] Reddy, A.L.N.; Banerjee, P, “Gracefully degradable disk arrays”, Fault-Tolerant Computing, 1991. FTCS-21.

[Rezaul 93a] Islam, S.M. Rezaul, “Performability Analysis of Disk Arrays”, IEEE Circuits and Systems, 1993.

[Schulze 89a] M. Schulze; G. Gibson; R. Katz; D. A. Patterson, “How reliable is a RAID?”, COMPCON Spring '89.

[Schwarz 92a] Schwarz, T.J.E.; Buckhard, W.A.; “RAID organization and performance”, Distributed Computing Systems, 1992.

[Seagate 05a] “Cheetah 15K.4 FC Product Manual”, 2005, http://www.seagate.com/staticfiles/support/disc/manuals/enterprise/cheetah/15K.4/FC/100220449c.pdf

[Smith 04a] Smith, N.; Clark, T.; “An Exploration of C2 Effectiveness: A Holistic Approach”, 2004 Command and Control Research and Technology Symposium, June 2004.

[SNIA 13a] SNIA Dictionary, http://www.snia.org/education/dictionary, Storage Networking Industry Association, 2013.

[Shooman 02a] Martin L. Shooman, “Reliability of Computer Systems and Networks”, John Wiley & Sons, 2002.

[Tai 96a] A. Tai, J. F. Meyer, A. Avizienis, “Software Performability: From Concepts to Applications”, Kluwer Academic Publishers, 1996.

[Takagi 85a] T. Takagi, M. Sugeno, “Fuzzy Identification of Systems and its Applicatons to Modeling and Control”, IEEE Transactions on Systems, Man and Cybernetics, vol. 15, Jan.-Feb. 1985..

[Thomasian 97a] A. Thomasian, J. Menon, “RAID5 Performance with Distributed Sparing”, IEEE Transactions on Parallel and Distributed Systems, 1997.

[Varki 03a] Varki, E.; Merchant, A.; Xu, J.; Qiu, X.; “An Integrated Performance Model of Disk Arrays” ; MASCOTS 2003.

[Weber 94a] D.P. Weber, “Fuzzy Fault Tree Analysis”, Fuzzy Systems, 1994, IEEE World Congress on Computational Intelligence.

[Wu 12a] D. Wu, “On the Fundamental Differences Between Interval Type-2 and Type-1 Fuzzy Logic Controllers”, IEEE Transactions on Fuzzy Systems, Vol. 20, No. 5, pp 832-848, Oct. 2012.

[Zhang 05a] R. Zhang, Y. A. Phillis, V. S. Kouikoglou, “Fuzzy Control of Queuing Systems”, Springer-Verlag, 2005.

[Zhang 06a] Q. Zhang, A. Riska, N. Mi, E. Riedel, E. Smirni, “Evaluating the Performability of Systems with Background Jobs”, Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN’06), IEEE, 2006.

LIST OF PUBLICATIONS

This section presents a list of the author's published publications and work in progress.

PUBLICATIONS IN CONFERENCES

[1] G. Navarro, M. Manic, "Fuzzy Performability Analysis of Disk Arrays", *IEEE ISIE*, 2006.

Abstract: The performability of disk arrays systems has been studied before [Islam 93a][Barnett 98a]. However, in the case of imprecise data, a fuzzy model can be the base for the performability analysis. In this paper a performability analysis of a disk array using a Markov Reward Model (MRM) is presented. The model considers the repair as the reconstruction (rebuild) of the redundancy, not as a hard drive replacement. With traditional, crisp arithmetic, for each change in a single model parameter the model would need to be run again, resulting in a family of curves difficult to interpret. In the approach presented in this paper, the rewards for each of the states of the MRM, as well as other disk array parameters are expressed through fuzzy numbers. The use of fuzzy arithmetic for the performability estimation of a disk array proved significant advantages. First, the model was able to capture the uncertainty variance of each of the model parameters. Secondly, as opposed to traditional, crisp arithmetic approach, the presented model provides the estimation of the lower and upper boundary of the system performability with a single run of the model.

[2] G. Navarro, M. Manic, "Predictive E-Mail Server Performability Analysis Based on Fuzzy Aritmetic", *IEEE IJCNN*, 2007.

Abstract: The performability of disk arrays systems has been studied before. However, in the case of imprecise data, a fuzzy model can be the base for the performability analysis. This paper presents a performability analysis of an MSExchange-like e-mail server. The analysis is based on a Markov Reward model. The performability analysis is accomplished through the use of fuzzy arithmetic. Unlike traditional Markov Chains, Fuzzy Markov Chains can successfully handle uncertain, imprecise probabilities. In cases where the failure rates, repair rates, or the workload parameters are uncertain, Markov Chains enhanced with fuzzy arithmetic provide means for comprehensive predictive performability analysis of a system. This performability analysis provides a valuable guideline regarding required resources such as the number of mailboxes, and therefore, the number of users the mail server can support with regards to the reliability and performance of the disk array used by the mail server. The fuzzy arithmetic helps in better visualization and estimation of the range of number of users the mail server is capable of servicing over long periods of time.

[3] G. Navarro, M. Manic, "Fuzzy Control of Sparing in Disk Arrays", *IEEE ETFA,* 2007.

Abstract: The redundancy regeneration (sparing or rebuild) algorithms in disk arrays face the problem of balancing between the data recovery activity within the array and the user workload acting upon the array at

the same time [Hou 93a]. If the algorithm favors the user workload so the user requests can always preempt the internal data recovery, then the data sparing can stall in the presence of a sustained workload. But on the contrary, if the data recovery is favored over the user requests, the latency of the user requests can be so high to reach unacceptable levels for the data transactions.

Using computationally intelligent techniques, like fuzzy logic, better algorithms to balance the level of user requests and the internal data recovery can be achieved. The disk array and data recovery process are modeled using the queue systems with vacations (QSV) [Medhi 03a]. A fuzzy algorithm to control the sparing is presented in this paper. The results indicate that by using fuzzy logic, a better balancing is achieved between the need to have an acceptable response time for the user requests and the data recovered as soon as possible.

[4] G. Navarro, M. Manic, "NFuSA – Neuro-Fuzzy Algorithm for Sparing in RAID Systems", *IEEE IECON,* 2007.

Abstract: Sparing, the process of rebuilding data in case of disk failure, has been a target of research since early 1990's [Muntz 90a]. The problem that these specific hardware/software control systems typically face in sparing is the tradeoff between serving requests – user's versus internal [Hou 93a]. If the algorithm favors user requests, in the presence of heavy workloads, the internal data recovery gets preempted resulting in risky delay of the data sparing. On the other hand, favoring internal data recovery requests over the user requests can result in high latencies per transaction that are unacceptable for the users of the RAID system. Intelligent, neuro-fuzzy controllers (NFCs) offer a way to improve the control process and enhance the ability of a system to achieve faster system response, while serving the internal requests at the same time. This paper presents the neuro-fuzzy enhancement of the traditional data recovery of a RAID system modeled with a Queue System with Vacations (QSV) [Medhi 03a]. Experimental results demonstrated better balancing between an acceptable response time for the user requests and the time for the data to be redundant again, resulting in both higher user satisfaction and better system reliability.

PUBLICATIONS IN JOURNALS

[5] G. Navarro, M. Manic, "FuSnap: Fuzzy Control of Logical Volume Snapshot Replication for Disk Arrays", *IEEE Transactions on Industrial Engineering,* 2011.

Abstract: This manuscript presents FuSnap, a fuzzy logic based controller that monitors and controls the snapshot process of a logical storage volume in a disk array. As disks do not linearly respond to the arrival rate of user accesses, FuSnap makes use of fuzzy logic as the means to achieve better control of their response time. The goal of the FuSnap controller is to reduce the response time caused by the copy-on-writes that occur during the snapping of a storage logical volume. The FuSnap controller, based on the response time of user accesses, makes the decision on whether to proceed with a copy-on-write or a redirect-on-write when a source logical volume is being copied to a snapshot logical volume. The benefits of FuSnap approach are twofold. Firstly, significant reductions in response time of user requests are obtained with the FuSnap approach over the

traditional Copy-on-Write snap approach. Secondly, these reductions in response time make the point-in-time copy of data a process less disruptive for database users. FuSnap was verified with two setups using HPUX workstations, one setup with 8 and the other with 32 disks.

PUBLICATIONS IN PROGRESS

[6] G. Navarro, M. Manic, D. Umberger, "Virtual Disk Replication on Disk Arrays using a Type-2 Fuzzy Controller", *work in progress*, 2015.

Abstract: Virtual Disk (VD) cloning is a data protection technique used by disk arrays to replicate the data in a VD. A typical consequence of the cloning on the disk array is the increase in response time of user reads and writes. A method for VD cloning using Type-2 fuzzy control is presented. This method is capable of balancing data replication on one side and impact on the response time of user reads and writes on the other. The method we present here can significantly reduce the throughput of the VD replication to reduce the impact on user response time. On the other hand, when user response time is below the maximum allowed, the background VD cloning can increase the cloning rate. The first contribution of this manuscript is: 1) a formula for data backup planning. This formula predicts the fraction of replicated data blocks in a combined (clone and snap) replication. 2) a novel FT2 control scheme that reduces latencies of user reads/writes response time in half; This control scheme was tested on an Itanium workstation with 120 disks and proved shortening user latencies in half (high response time of 60ms in half within 30 seconds only) as an average case.

PATENTS

[7] G. Navarro, M. Manic, David K. Umberger, inventors; Hewlett Packard, assignee; "Control of Sparing in a Storage System", US Patent number 8,201,018; issued June 12, 2012.

Abstract: Embodiments include methods, apparatus, and systems for controlling of sparing in a storage system. In one embodiment, a method compares a first amount of time to complete sparing of data from a failed disk in a storage system with a second amount of time to complete a user request to the storage system in order to determine when to create a copy of the data from the failed disk.

[8] G. Navarro, David K. Umberger, inventors; Hewlett Packard, assignee; "Creating Snapshots of Data Using a Selected One of Different Snapshot Algorithms", US Patent 8,650,145; February 11, 2014.

Abstract: Embodiments include methods, apparatus, and systems for controlling of sparing in a storage system. In one embodiment, a method compares a first amount of time to complete sparing of data from a failed

disk in a storage system with a second amount of time to complete a user request to the storage system in order to determine when to create a copy of the data from the failed disk.

[9] G. Navarro, M. Manic, David K. Umberger, inventors, "Managing Processing of User Requests and Data Replication for a Mass Storage System", US Patent 9,063,835; June 23, 2015.

Abstract: A technique includes determining a workload on mass storage system that is associated will user requests during a time in which mass storage system is replicating data from a source data unit of the mass storage system to a replica storage unit of the mass storage system. The technique includes determining a progress rate associated with the replication and managing processing of the user requests and the data replication for the mass storage system, including initiating corrective action in response to determining that the workload is near a predetermined maximum workload threshold and the progress rate is near a predetermined minimum threshold.

PATENT APPLICATIONS

[10] G. Navarro, M. Manic, David K. Umberger, inventors, "Regulating Power Consumption of a Mass Storage System", US Patent Application 20130326249; December 5, 2013.

Abstract: A technique includes receiving first work requests that are associated with a user workload. The technique includes using a machine to transform the first work requests into second work requests that are provided to components of a mass storage system to cause the components to perform work associated with a workload of the mass storage system; and regulating a power consumption of the mass storage system, including regulating a rate at which the second work requests are provided to the components of the mass storage system.