Fuzzy Control of Sparing in Disk Arrays

Guillermo Navarro
Hewlett Packard
11311 Chinden Blvd.
Boise, ID 83714, USA
guillermo.navarro@hp.com

Milos Manic
University of Idaho at Idaho Falls
1776 Science Center Dr., Ste. 306
Idaho Falls, ID 83402, USA
misko@uidaho.edu

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 [1]. 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) [2]. 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.

1. Introduction

One of the first sparing models was proposed in [3]. Since then, other analysis of the sparing process in disk arrays have been published [1,3-7]. The process of reconstruction of the data redundancy has been referred to as rebuild [4] or sparing [5]. In papers [4,5], the sparing process for a RAID5 disk array is modeled using a queue system with vacations [2].

The goal of the optimal design and control of queueing systems has been an area of research since the 1950 s [8]. The goal of the optimal design and control of queues is to determine the optimal system parameters such as optimal service rate or number of servers [8]. The use of computationally intelligent techniques for the control of queue systems has been researched before. Fuzzy logic has been used for the control of queues [9,10] .

In this paper a queue system with vacations is used to model the sparing process as in [4,5]. A fuzzy controller is used for the measurement of the input parameters and the control of the queues as it is proposed in [9,10]. In this paper the authors make a comparison between two approaches to control the sparing process: 1) the traditional 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; and 2) a fuzzy-logic controller that uses three parameters (response time of user requests, fraction spared and time of sparing) to make the decision whether to allow user requests to proceed or proceed with the sparing.

The first contribution of this paper is to show how the fuzzy control of queueing systems can be applied to the sparing process. The second contribution is that the results show that fuzzy logic is a technique that improves the sparing process for high demands on the disk array. The improvement shows by achieving a faster sparing than the traditional empty/no-empty control model, but without impacting the latency of user request.

2. Disk Array Queueing Model

For this paper, a RAID5 disk array will be used for the analysis. The disk array consists of D disks. The D disks are divided up in RAID5 disk groups of G disks. The data on the disks is divided up in data blocks. 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 number of N_b blocks per disk depends on the capacity of the disk C_d.

The size of each data block $B_{i,j}$ is B_s. For this paper, a block size B_s=128KB will be used. References [1,4-10] can be consulted for the different cases for reads and writes during *optimal* (no disk failed) conditions and during *degraded* (one disk failed) conditions.

A queueing system in which the server may be turned off is said to be a queueing system with vacations [2]. The requests to the queueing system

1-4244-0826-1/07/$20.00 © 2007 IEEE

arrive at a rate A. The server processes the requests at a S rate. When the server is idle (queue is empty) it turns itself off and goes on vacation to do some process in the background. After some time V, the server comes back and rechecks the queue. If not empty, the server turns itself on. Otherwise, it keeps itself off and proceeds to another vacation. One of the advantages of the fuzzy logic is the ability to easily model and control systems in which precise, crisp, fully tractable mathematical models can be difficult to derive. The problem of finding optimal policies for networks of queues is not trivial. Some queue optimization problems are often intractable [11]. Fig. 1 shows the model used for the disk array controller and the disks. This model combines the QSV with the queueing network formed by the disk array controller and the disks. The user requests to the queueing system arrive at a rate A. The disk array controller processes requests with S service rate. For modeling the disk array workload, all writes will be assumed to be Read-Modify-Writes (RMW) [1,3-7]. The disks receive requests at a rate A_d and process requests at a rate S_d.

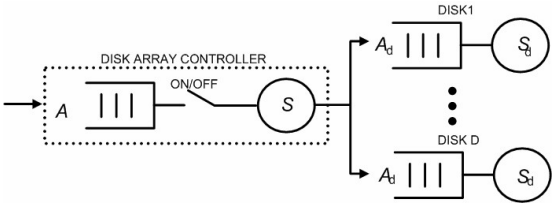

Figure 1. Queueing System of disk array controller and disks using a QSV.

The disk array controller will be modeled with a constant service rate of S = 12,500 requests per second. The service rate S_d is the inverse of the service time of the disk T_d . The T_d is the sum of the rotational latency, seek time and transfer time [13]. The sum of the rotational latency and the seek time is referred to as the disk position time (*dpt*) [13]. The equation that relates the *dpt* to the disk queue is in [13]:

$$dpt = a + \sqrt{1 + disk\_queue}^{\,b} \qquad \text{Eq.1}$$

The parameters a and b are constants depending on the specific disk modeled and the type of access (read or write). The T_d was modeled using data obtained from the ST373454FC Seagate disk [12]. The workload used for modeling and simulation was random, for both 4KB (small transfers) and 128KB (large transfers). Measurements of the response time and IOs per second (IO/s) from this Seagate disk under varying queue sizes (from 1 to 60), were used to determine the *dpt* for each queue size. The values found for random reads were a=2 and b=4.6. The values for random writes were a=2.9 and b=4.1.

3. 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 based on an approach of fuzzification of controller. It is important to note that this solution will be more sophisticated than the traditional QSV model, where the sparing process only occurs when the queue is empty.

The proposed solution involves an extended set of parameters. The input parameters will be fuzzified and the decisions will be based on fuzzy values. In Fig. 2 we show a graphical model of the proposed solution.

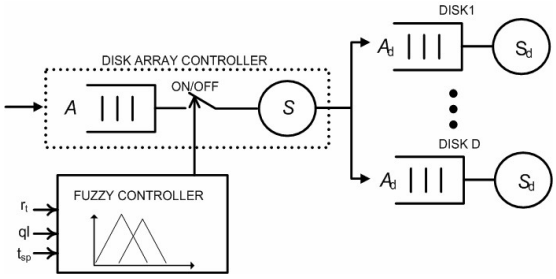

Figure 2. Fuzzy Control of the QSV for Sparing.

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

The queue length of the controller ql is included to allow the sparing process to execute even if there are requests in the queue waiting to be served. The key question is what the biggest size of the queue is before it is decided to stop the sparing process and serve the requests. If we assume an average response time of RT_{avg} = 10ms for the users and an average throughput (arrival) of A_{avg} = 1,000 IO/s, then we can use Little s theorem [10], and estimate the average queue length at ql = 10. Now we can set a maximum queue length, ql_{max} to some value. For the modeling and simulation, ql_{max} = 20, the normalized ql_n was then obtained by dividing ql by ql_{max}.

The response time r_t can be normalized if we consider an upper limit to the response time which user applications can withstand without causing long response times for the users. One example is the Microsoft Exchange Servers. Latencies above 50ms are not acceptable [14],. For the simulation in this paper, it was assumed that a delay of rt_{max}=50ms was the maximum that can be tolerated by the users. The normalized response time rt_n, was then obtained by the division of r_t by rt_{max}.

The time elapsed for the sparing t_{sp} is also being normalized. The assumption made here is that there is a maximum time acceptable for the user without the

redundancy of the data restored. 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 obtained by dividing the elapsed sparing time t_{sp} by tsp_{max}.

With the three input parameters normalized to a range between [0,1], the fuzzy triangular membership functions can be defined. Three linguistic values were assigned (ZRO, MID, and ONE), which stand for zero , middle value , and one , respectively. This is following the same technique shown in [10]. The purpose of these three membership functions is to have a measure of how close the input parameter is to the numerical value zero (ZRO), 0.5 (MID), or 1 (ONE).

Now the next step is the specification of the rules for the rule base. The linguistic criteria can be summarized as follows. Firstly, the response time of the disk array controller r_t, should be kept as low as possible. If the response time r_t, is low, we can proceed with the sparing. Secondly, the sparing process should be finished as soon as possible. The closer we are to tsp_{max}, the more priority should be given to the sparing process. Thirdly, the lower the queue length is, the more we can spare since just few processes will be delayed.

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 is given in Table 1.

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 MID YES	MID ONE MID YES	ONE ONE MID NO
ZRO ONE ONE YES	MID ONE ONE YES	ONE ONE ONE YES

Table 1. Rule base.

4. Simulation and Results

CSIM19 [15] was used for the simulation. CSIM19 has the capability to model a system as a collection of processes and servers with queues. The workload applied was 75% reads (3:1 ratio, as typical for Exchange Server environments [16]). A disk array with 80 disks was simulated. The throughputs applied for comparison were 1000, 5000, 9000, and 10,000 IO/s. The throughputs were maintained constant during the entire duration of the simulation. The intention was to measure the variations in response time 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. The closer to zero the sparing process took, the better, since this indicates the sparing process finished sooner. On the vertical axis the graphs show the response time seen by the user requests. The closer to zero the response time is, the better, since this indicates the user requests were processed faster and users saw better response from the disk array.

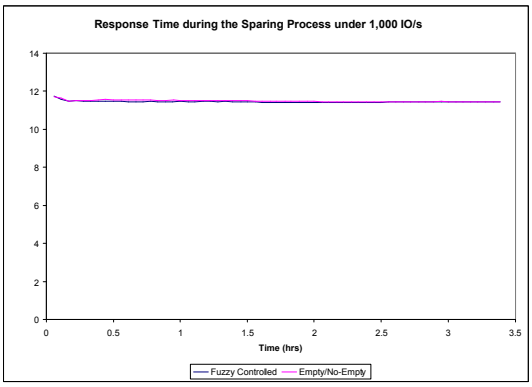

Figure 3. Comparison of response time during the sparing for 1,000 IO/s.

Fig. 3 shows the result for the 1,000 IO/s throughput applied to the disk array. This result shows the fuzzy controller performing at the same level as the the traditional empty/no-empty controlled sparing. The graph shows both sparing processes finishing in the same time (around 3.5 hours after the disk failed). For both cases, the response time was around 12ms. So, in a low throughput, both the fuzzy controlled sparing and the traditional empty performed the same

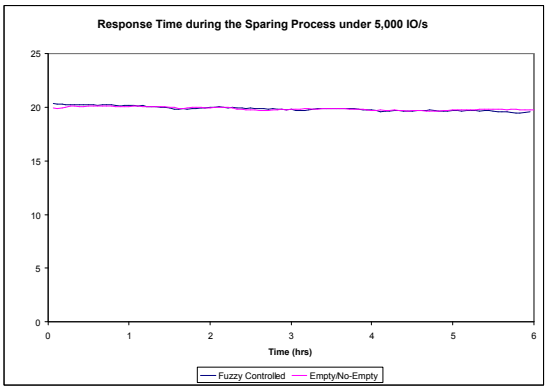

Figure 4. Comparison of response time during the sparing for 5,000 IO/s.

Fig. 4 shows the result for the 5,000 IO/s throughput applied to the disk array. This result shows the fuzzy controller performing the same as the traditional empty/no-empty controlled sparing. In the low throughput ranges, both schemes of control seem to

perform well. This can be explained by the fact that at low throughputs, the empty/no-empty controlled sparing has a lot of opportunities to execute in between requests.

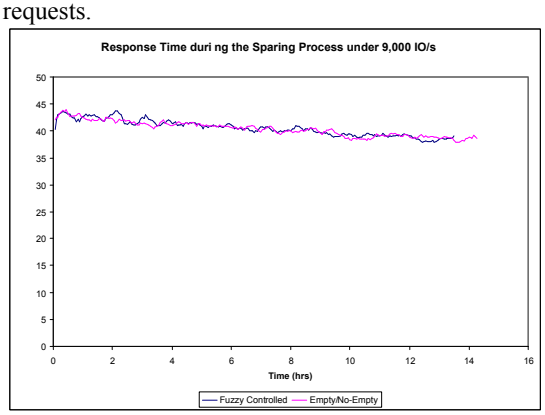

Figure 5. Comparison of response time during the sparing for a 9,000 IO/s.

Fig. 5 shows the result for the 9,000 IO/s throughput applied to the disk array. At high throughputs we can see that the fuzzy controlled sparing process outperforms the empty/no-empty slightly by finishing the sparing shortly before. Both have the same response time in the order of 40~50ms. The fuzzy controlled sparing could finish before since it executed the sparing even in the presence of some requests in the queue. The traditional sparing preempted itself more since the controller queue was busy most of the time. This is the result that is encouraging since it shows that the fuzzy controlled sparing can be tuned in future experiments so it can improve its performance over the traditional empty/no-empty controlled sparing.

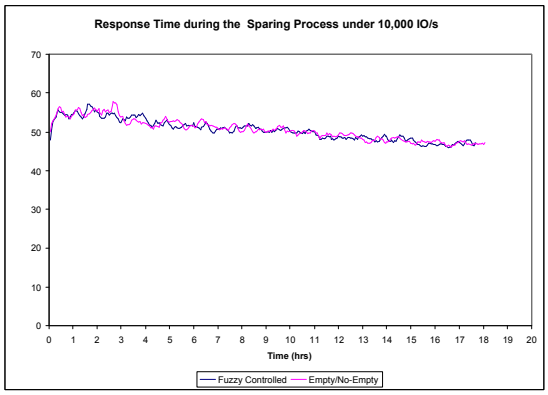

Figure 6. Comparison of response time during the sparing for 10,000 IO/s.

Fig. 6 shows the result for the 10,000 IO/s throughput applied to the disk array. This result shows also a little improvement over the traditional empty/no-empty controlled sparing. It should be noted that the fuzzy controlled sparing finishes a little before without impacting the user response time. With this in mind, we can say there is a benefit provided by the fuzzy controller, even if it is in the high throughput ranges.

5. Conclusions

First and foremost, this paper shows that fuzzy logic can be applied to improve the sparing process in disk arrays. The fuzzy controller for control of the sparing process proved to be better in the high throughput ranges than the simple QSV approach that waits for the queue to be empty. For the high throughput regions, the fuzzy-controlled sparing process slightly outperformed the traditional QSV sparing process by finishing a little earlier. More importantly, it finished earlier without impacting the user response time. This proves that the fuzzy logic control for sparing can be an alternative to meet QoS requirements for disk arrays. More investigation can be done with high throughputs (greater than 5,000) to ascertain if the fuzzy controller can outperform the traditional QSV in low throughputs or finish earlier in high throughputs.

References

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

[2] J. Medhi, Stochastic Models in Queueing Theory , Academic Press, 2003.

[3] R.R. Muntz, John C.S. Lui, Performance Analysis of Disk Arrays Under Failure , IEEE, 1990.

[4] J. Menon, A. Thomasian, Performance Analysis of RAID5 Disk Arrays with a Vacationing Server Model for Rebuild Mode Operation , IEEE, 1994.

[5] A. Thomasian, J. Menon, RAID5 Performance with Distributed Sparing , IEEE, 1997.

[6] A. Thomasian, Rebuild Options in RAID5 Disk Arrays , IEEE , 1995.

[7] H. Kari, H. Saikkonen, S. Kim, F. Lombardi, Repair Algorithms for Mirrored Disk Systems , IEEE, 1995.

[8] L, Tadj, G. Choudhury, Optimal Design and Control of Queues , TOP, Vol. 13, December 2005.

[9] Y. A. Phillips, R. Zhang, Fuzzy Service Control of Queuing Systems , IEEE, 1999.

[10] Y. A. Phillips, R. Zhang, V. S. Kouikoglou, Fuzzy Control of Queuing Systems , Springer-Verlag, 2005.

[11] C.H. Papadimitriou. J. H. Tsitsiklis, The Complexity of Optimal Queuing Network Control , SCTC, 1994.

[12] www.seagate.com/staticfiles/support/disc/manuals/ enterprise/ cheetah/ 15K.4/ FC/ 100220449c.pdf

[13] E. Varki, A. Merchant, J. Xu, X. Qiu, An integrated performance model of disk arrays , IEEE, 2003.

[14] technet.microsoft.com /en-us/ library/ 29f01985-7b44-47cb-96f7-d7c92fd8e867.aspx

[15] H. Schwetman, CSIM19: A powerful tool for building system models , 2001 Winter Simulation Conf.

[16] technet.microsoft.com /enus/ library/ bb124226.aspx