The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON)
Nov. 5-8, 2007, Taipei, Taiwan

NFuSA – Neuro-Fuzzy Algorithm for Sparing in RAID Systems

Guillermo Navarro
Hewlett-Packard
guillermo.navarro@hp.com

Milos Manic
University of Idaho
misko@ieee.org

***Abstract*— Sparing, the process of rebuilding data in case of disk failure, has been a target of research since early 1990's [1]. The problem that these specific hardware/software control systems typically face in sparing is the tradeoff between serving requests – user's versus internal [2]. 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 response times 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 *Q*ueue *S*ystem with *V*acations (QSV) [3]. 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.**

I. INTRODUCTION

The analysis and modeling of the RAID systems under failure has been of interest for researchers for almost two decades [1,2,4-9]. The control of queueing systems [11-14] is one area of research that is applicable to the problem of balancing between the response time for the customer and the time for the data to be made redundant in a RAID system.

Computationally intelligent techniques, like neural nets and fuzzy logic have been applied to the control of queueing systems. For example, fuzzy logic was used in the control of data flow in TCP communication networks to minimize congestions [15,16]. Fuzzy logic has been applied to the control of the sparing process [36]. Neural networks have been also applied to queueing systems. For example, neural networks were used to tune computer system performance [17], for modeling of queueing systems [18,19], for congestion control in communication networks [20-24], and for control of self-similar communication systems [26,27].

This paper presents an extension of the QSV where modeling of the sparing processes was based on a neuro-fuzzy controller (NFC). Such a NFC is used for both measuring of the input parameters and for the control of the queue. Two approaches to control the sparing process are compared: 1) the empty/no-empty QSV model, where the sparing process only takes place when the queue is empty, this is, when there are no external (users) requests; and 2) a neuro-fuzzy 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 be carried out or continue with the sparing.

This paper demonstrates a novel approach based on neuro-fuzzy control of queueing systems to the sparing process. Testing results illustrate a lower impact on the response time of user requests while completing the sparing at the same time as the empty/no-empty QSV.

The organization of this paper is as follows: Section 2 describes the sparing model based on a RAID1 system. Section 3 describes the neuro-fuzzy control implemented for the sparing process. Section 4 describes the simulation executed to test the performance of the neuro-fuzzy control. Section 5 presents conclusions.

II. QUEUEING AND SPARING MODELS

One of the first sparing models was proposed in 1990's [1]. Since then, other analysis of the sparing process in RAID systems have been published [1,2,4-9]. In this paper, 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 capacity of the disk C_d.

$$N_b = C_d / S_b \tag{1}$$

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 paper, a block size S_b=128KB will be used. This is the default size for the HP StorageWorks 1000/1500 MSA [28]. Each disk block is referred to as B_i, where i=1,2,3,…,N_b.

Fig. 1 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.

A workload with arrival rate λ is applied to the RAID1 system controller by the users. Instead of specifying an arrival rate λ to the disk controller, the throughput χ in IOs per second (IO/s), can be specified by

$$\chi = \frac{1}{\lambda} \tag{2}$$

The throughput is distributed across the disks. A balanced workload across the disks is used in this paper.

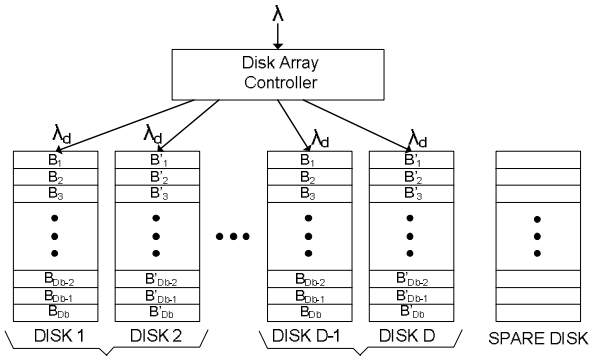

Fig. 1. RAID1 disk array data layout.

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. 2 shows an example of a failed disk; in this case, disk *D*-1 failed and the spare disk 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 newly disk *D*-1.

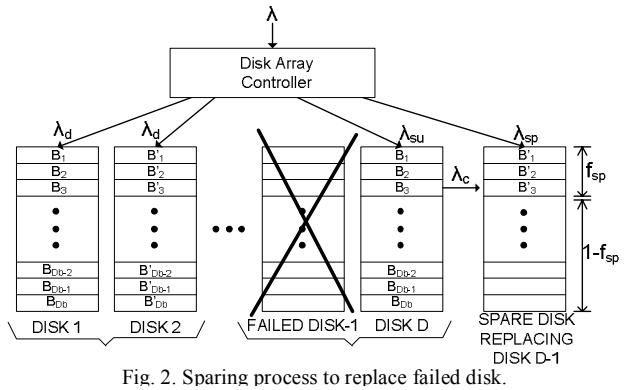

Fig. 2. Sparing process to replace failed disk.

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:

$$f_{sp} = B_c / N_b \tag{3}$$

When the RAID1 system is sparing, the combined throughput of the disks change 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. The other *D*-2 disks are serving requests as they would normally do.

A queueing system in which the server may be disconnected (turned off) or removed is said to be a queueing system with vacations [3]. Fig. 3 illustrates the concept of queueing systems with vacations (QSV).

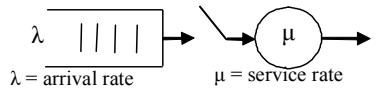

Fig. 3. Queueing System with Vacations (QSV).

The requests arrive at a rate λ to the queueing 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 paper as the empty/no-empty approach to control of the QSV.

The complete model of the RAID1 system is based on the central server model [29] with the addition of the QSV as shown in Fig. 4. The user requests to the queueing system arrive at a rate λ. The RAID controller (the server) processes requests at a μ service rate. Finding optimal policies for networks of queues is not a trivial problem. Some queue optimization problems are probably intractable [30]. This is one of the reasons why fuzzy logic can be used to control queueing systems. Fuzzy logic offers the possibility of easily modeling and controlling systems in which mathematical models can be hard to derive. A simulation of the queueing system in Fig. 4 was the approach used in this paper to show the improvements made by the neuro-fuzzy controller.

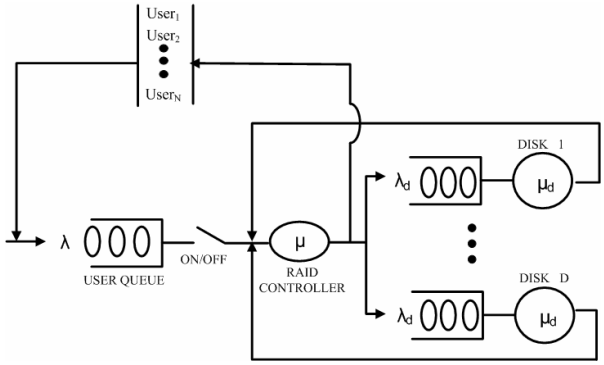

Fig. 4. Queueing system of disk array controller and disks.

The model of the disks used in the simulation is based on the ST373454FC Seagate disk [31]. 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) and transfer time, t_{xfer}.

$$T_d = t_{rot} + t_{seek} + t_{xfer} \tag{4}$$

The disk position time, *dpt*, is defined this way

$$dpt = t_{rot} + t_{seek} \tag{5}$$

The disks will be modeled using the following equation

$$T_d = dpt + t_{xfer} \tag{6}$$

The disk service times are difficult to estimate since some factors, like disk specifications, disk caching and scheduling policy are hard to determine [32]. The data used for this simulation came from measurements made on the ST373454FC Seagate disk. For random workloads the disk position time can be modeled by this equation in [32]:

$$dpt = a + \frac{b}{\sqrt{1 + disk\_queue}} \tag{7}$$

(8)

The values found for random reads were a=2 and b=5.8. The values for random writes were a=2.5 and b=6.85. Fig 5 shows the graph for the measured and modeled disk position times. The t_{xfer} for both 4KB and 128KB transfers were 0.06ms and 2.27ms, respectively.

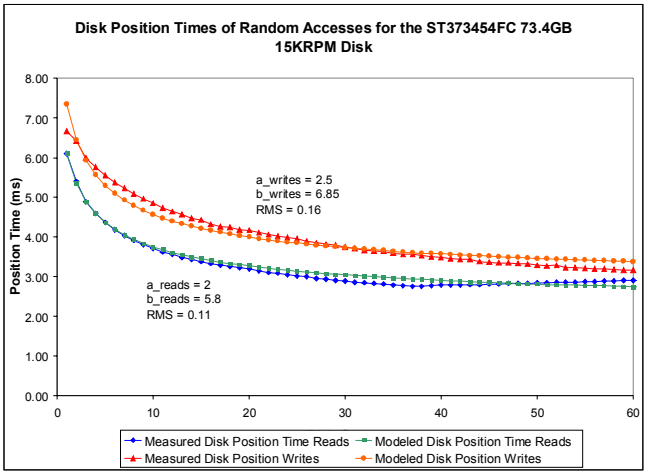

Fig. 5. Disk Position Times measured.

III. NFuSA (NEURO-FUZZY SPARING ALGORITHM)

The neuro-fuzzy solution proposed to balance the time needed to complete the sparing and the response time 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. 6 we show a graphical model of the proposed solution.

The input parameters of the fuzzy controller are three: 1) The fraction of sparing, f_{sp}; 2) the response time 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 neuro-fuzzy controller the three parameters were normalized. The first parameter f_{sp} from (3), is in the range [0,1]. The other two parameters are normalized by making two assumptions.

The first assumption is that the response time r_t can be normalized with respect to certain upper limits of the response time 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 [33]. For this paper, it was assumed that a delay of rt_{max}=50ms was the maximum that can be tolerated. That was the value used in the simulation of the sparing. The normalized response time rt_n used by the fuzzy controller is then:

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

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:

$$tsp_n = \frac{t_{sp}}{tsp_{\max}} \tag{9}$$

With the three input parameters normalized, now 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. [16]. Fig. 7 shows the triangular fuzzy membership functions for all three input parameters.

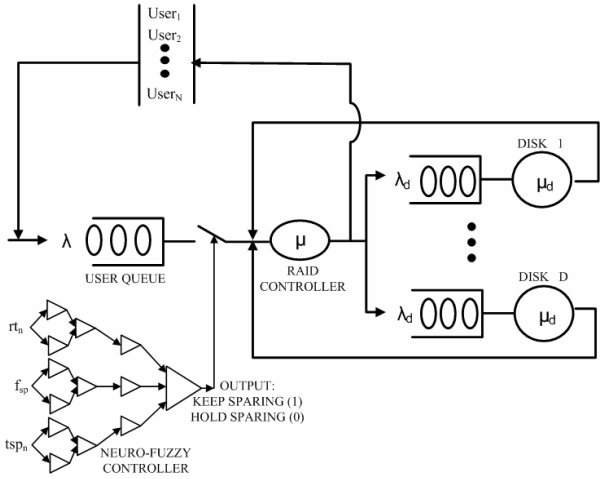

Fig. 6. Neuro-Fuzzy controller of the QSV for sparing

The three normalized parameters (f_{sp}, rt_n, tsp_n) in the range [0,1] are the input to the fuzzifier neural net. Fig. 8 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 controller part, which is based on the rule base.

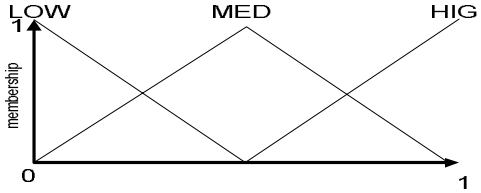

Fig. 7. Membership functions for the normalized parameters.

The rule base can be implemented according to the following linguistic criteria: 1) The response time of the RAID controller r_t, should be kept as low as possible. If the response time r_t, is LOW, the sparing can continue without

any risk of affecting the user response time. 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 response time.

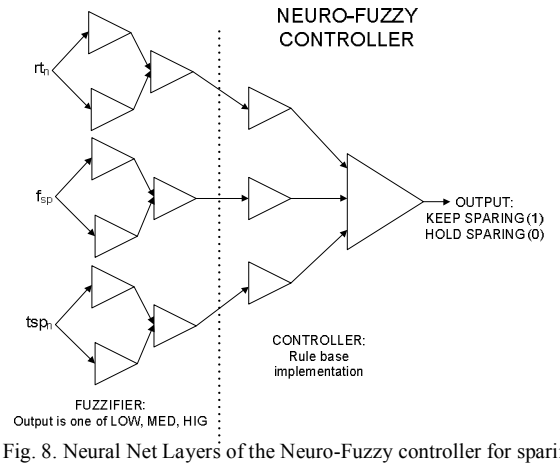

Fig. 8. Neural Net Layers of the Neuro-Fuzzy controller for sparing.

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 1. 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).

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 MED	KEEP	MED HIG MED	KEEP	HIG HIG MED	HOLD
LOW HIG HIG	KEEP	MED HIG HIG	KEEP	HIG HIG HIG	KEEP

Table1. Rule base for the Neuro-Fuzzy controller for sparing.

IV. SIMULATION AND RESULTS

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 [34]. The testing parameters were chosen to resemble a typical Exchange Server environment [35]: 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 applied for comparisons were 500, 1000, 2000, 3000 and 4000 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. On the vertical axis the graphs show the response time (latency) seen by the user requests. Fig. 9 shows the result for the 500 IO/s throughput applied to the RAID system. For a light workload, both sparing approaches produced the same results. Fig 10 shows the tridimensional plot of the three normalized variables during the sparing. In this case, since the neuro-fuzzy and the empty/no-empty sparing had the same results, the trajectories overlap in tridimensional graph of Fig. 10.

Fig. 11 shows the result for the 1000 IO/s throughput applied to the RAID system. For this workload, the neuro fuzzy sparing proved superior in terms of the response time seen during the sparing. The neuro-fuzzy could maintain a 10ms response time during the sparing. The empty/no-empty approach showed a bigger response time up to 20ms. Both sparing approaches finished the sparing in the same time. Fig 12 shows the tridimensional plot of the three normalized variables during the sparing. In this case, the neuro-fuzzy trajectory is a straight line since the response time was constant all along the sparing.

Fig. 13 shows the response time for a 2000 IO/s throughput. In this case the neuro-fuzzy sparing showed a better response time during the sparing process than the empty/no-empty approach even though the difference is not that big as in the 1000 IO/s case. As the throughput is increased, the difference between both approaches will be less. More investigation is needed to determine if a change in the fuzzifier neural net or the controller neural net can make the improvement at higher throughputs more indicative. Fig. 14 shows the trajectory of sparing for the 2000 IO/s throughput for both sparing approaches. Fig 15 and 16 show the comparison for 3000 IO/s. The results also show a slight improvement. This slight improvement is also shown in the 4000 IO/s case.

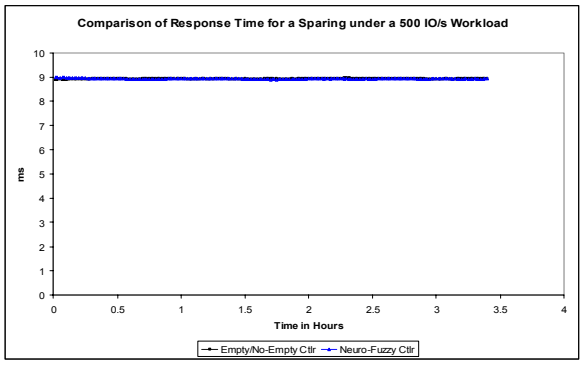

Fig. 9. Comparison of latency for 500 IO/s throughput.

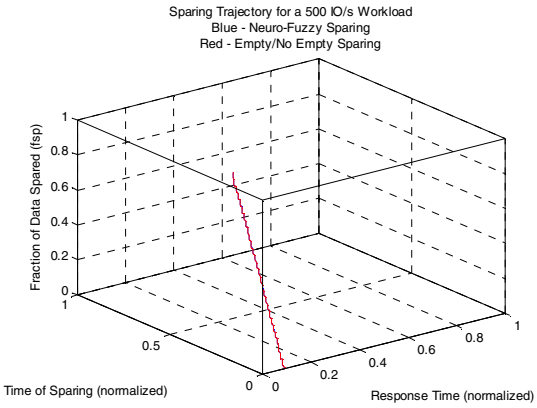

Fig. 10. Trajectory of the sparing process for 500 IO/s

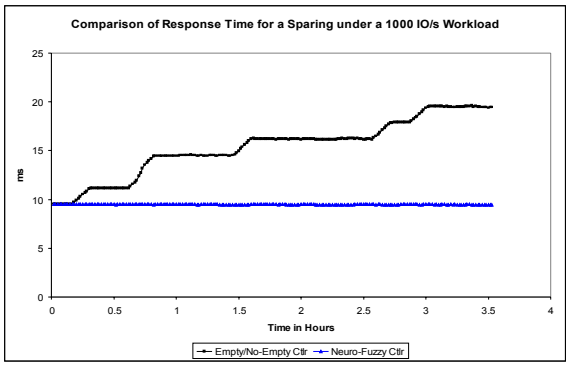

Fig. 11. Comparison of latency for 1000 IO/s throughput.

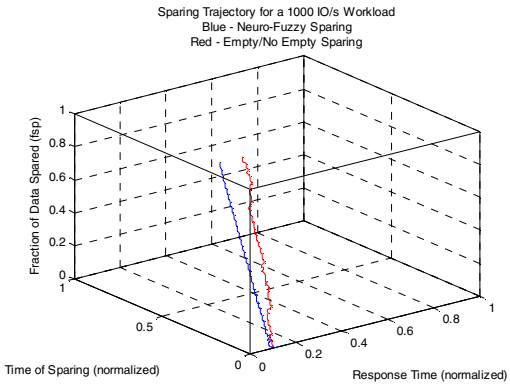

Fig. 12. Trajectory of the sparing process for 1000 IO/s throughput.

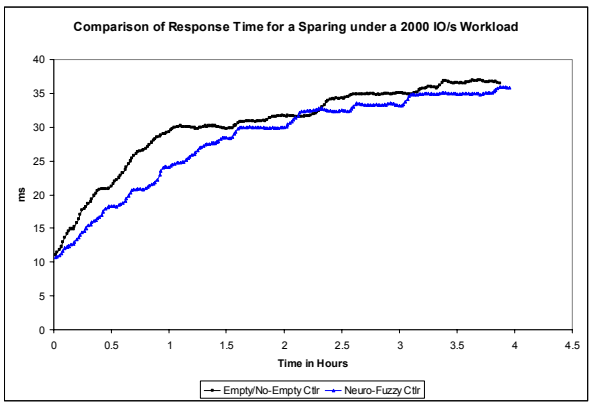

Fig. 13. Comparison of latency for 2000 IO/s throughput

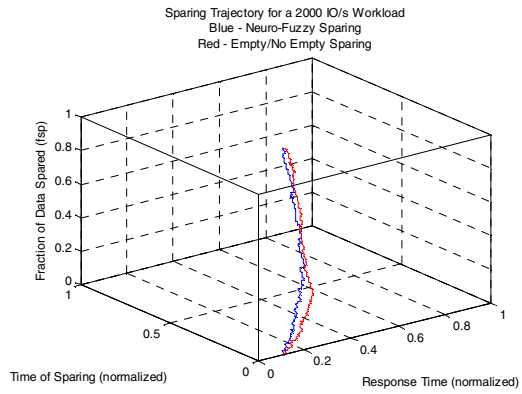

Fig. 14. Trajectory of the sparing process for 2000 IO/s throughput.

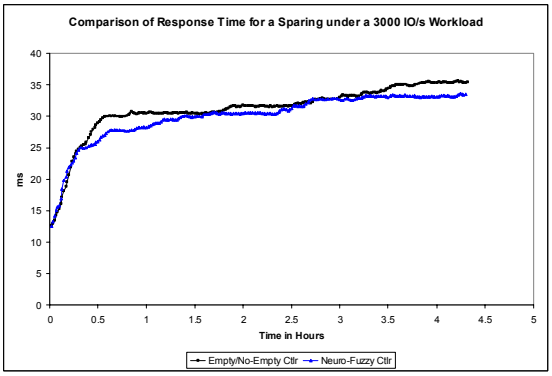

Fig. 15. Comparison of latency for 3000 IO/s throughput

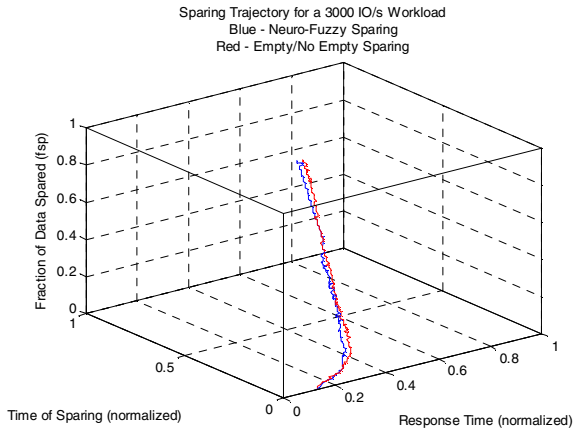

Fig. 16. Trajectory of the sparing process for 3000 IO/s throughput.

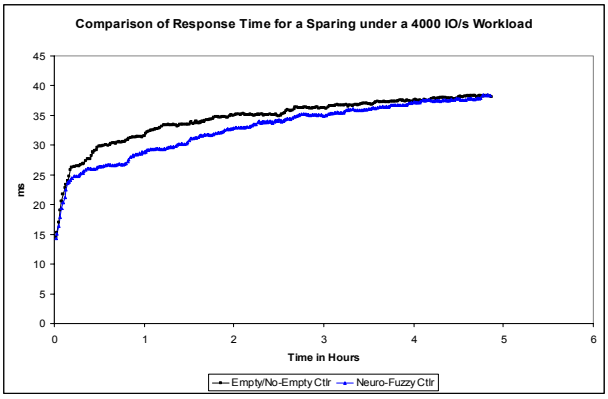

Fig. 17. Comparison of latency for 4000 IO/s throughput

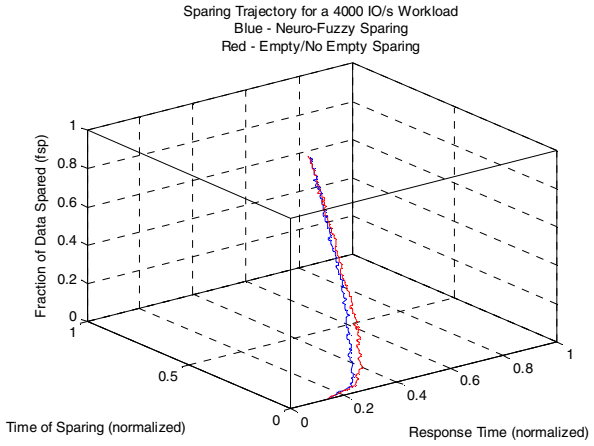

Fig. 18. Trajectory of the sparing process for 4000 IO/s throughput.

V. CONCLUSIONS

The approach presented in this paper demonstrated two distinct advantages. A traditional queueing system with vacations for sparing of RAID systems can be successfully extended via use of intelligent control based on neuro-fuzzy controller. Moreover, in the low throughput ranges, this intelligent controller proved to be superior to a simple yes/no empty approach of waiting for the queue to become vacant.

In regions of high throughput, the neuro-fuzzy controlled sparing approach demonstrated improvements in the response time (not as dramatic as in the case of low throughput range though). Overall, the neuro-fuzzy controller presented in this paper improved the response time for user requests. This proves that the neuro trained fuzzy logic controller for sparing can be a successful approach to meeting QoS requirements for RAID systems. Regions with high throughputs are worth further investigating (the presented approach may prove to be even more superior here).

REFERENCES

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

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

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

[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] J. Menon, D. Mattson, "Comparison of sparing alternatives for disk array", Int'l Symposium. on Computer Architecture, 1992.

[7] J. Menon, D. Mattson, "Distributed Sparing in Disk Arrays", IEEE, 1992.

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

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

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

[11] Heyman D.P., "Optimal operating policies for M/G/1 queueing systems", Operations Research, 1968.

[12] Zhang Z.G., Vickson R., Love E., "Optimal service polices in an M/G/1 queueing system with multiple vacation types"

[13] Altman E., Nain P., "Optimal control of the M/G/1 queue with repeated vacations of the server", IEEE, 1993.

[14] J. Ke; "The Optimal Control in Batch Arrival Queue with Server Vacations, Startup and Breakdowns", Yugoslav Journal of Operations Research, 2004.

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

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

[17] J.P. Bigus, "Applying Neural Networks to Computer System Performance Tuning",IEEE, 1994.

[18] D. Radev, S. Radeva, "Vector Quantization for Simulation Modelng of Queueing Systems", IEEE, 2003.

[19] K. Zatwarnicki, "Proposal of a neuron-fuzzy model of a WWW server", IEEE, 2005.

[20] H. C. Cho, S. Fadali, H. Lee, "Neural Network Control for TCP Network Congestion", IEEE, 2005.

[21] S. Le, C. Hou, "A Neural-Fuzzy System for Congestion Control in ATM Networks",IEEE,2000.

[22] L. Wang, S. Du, J. Lin, "An AQM Scheme Using Neural Network Based Predictive Control", IEEE, 2004.

[23] K. Rahnami, P. Arabshahi, A. Gray, "Neural Network Based Model Reference Controller for AQM of TCP Flows", IEEE, 2004

[24] Y. Yu, C. Cao, G. Yu, C. Li, "Design of Neural Model Predictive Controller for AQM Management", IEEE, 2005.

[26] H. Yousefi'zadeh, E. A. Jonckheere, "Dynamic Neural-Based Buffer Management for Queueing Systems with Self-Similar Characteristics", IEEE, 2005.

[27] H. Yousefi'zadeh, E. A. Jonckheere, J. A. Silvester, "Utilizing Neural Networks to Reduce Packet Loss in Self-Similar Teletraffic Patterns", IEEE, 2003.

[28] Hewlett-Packard, "HP StorageWorks 1000/1500 Modular Smart Array Command Line Interface", 3rd Edition (May 2006)
http://h20000.www2.hp.com/bc/docs/support/SupportManual/c00683579/c00683579.pdf

[29] Buzen, J.P., "Analysys of system bottlenecks using a queueing network model", ACM, 1971.

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

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

[32] E. Varki, A. Merchant, J. Xu, X. Qiu, "An Integrated Performance Model of Disk Arrays", IEEE, 2003.

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

[34] CSIM19, Mesquite Software, www.mesquite.com

[35] http://technet.microsoft.com/en-us/library/bb124226.aspx

[36] G. Navarro, M. Manic, "Fuzzy Control of Sparing in Disk Arrays", ETFA Conference, 2007.