STORAGE REBUILD FOR AUTOMATIC FAILURE RECOVERY IN VIDEO-ON-DEMAND SERVERS

Y.B. Lee and P.C. Wong

Department of Information Engineering
The Chinese University of Hong Kong, Hong Kong
{yblee, pcwong}@ie.cuhk.edu.hk

ABSTRACT

In a previous study [4], we proposed a Redundant Array of Inexpensive Servers (RAIS) architecture for designing scalable and fault-tolerant video-on-demand systems. Video data are striped across an array of autonomous servers, resulting in a scalable architecture where more concurrent video sessions can be supported by simply adding more servers. Moreover, by adding data redundancy among the servers, client recovery algorithms can be implemented to sustain server failure and provide non-stop video services. In this paper, we consider the failure recovery issue in RAIS. Specifically, we propose and analyze three algorithms for rebuilding data at a failed server to a spare server in order to restore the system back to normal operation. We derive the performance model and use numerical results to show that automatic rebuild can be done in reasonable time using the proposed rebuild algorithms.

1. INTRODUCTION

In [4], we proposed a Redundant Array of Inexpensive Servers (RAIS) architecture for designing scalable and fault-tolerant video-on-demand systems (Figure 1). Video units of each stream are striped across an array of servers and a fast packet switch is used to connect server and client stations. Each client contacts the server one by one to retrieve units of a particular video stream for playback and hence client load is uniformly shared by all servers irrespective of the skewness [3] of the video titles. This architecture allows one to build large VoD systems from smaller, less expensive server hardware (such as PCs). In addition, when scaling up the system to more concurrent users, the video storage need only be redistributed rather than replicated.

Besides scalability, the RAIS architecture also enables one to incorporate various levels of redundancy to implement server-level fault tolerance. The key is to reserve stripe units within a stripe to store redundant data as shown in Figure 2. Let there are N_S servers in the system. To sustain K simultaneous server failures, we would need to reserve K redundant units in every stripe. The example shown in Figure 2 uses a redundancy level of K=1. Therefore, when a server (say, server 2) fails, the client can recover the lost stripe unit of the failed server by retrieving the corresponding stripe units from the remaining servers, and performing erasure-correcting computation over the received units. Through proper client buffering as discussed in [4], video playback continuity can be maintained during the failure.

Although a RAIS system remains operational in failure mode, we still need to rebuild the data at the failed server into a spare server to protect the system from further server failures. This paper studies various algorithms for this rebuild process.

2. ANALYSIS OF REBUILD ALGORITHMS

Figure 2 shows the use of a spare server to store rebuilt data - *hot sparing*. Note that the spare server is not used under normal-mode operation. When a server fails, the lost data in the failed server are rebuilt into the spare server. When the rebuild process finishes, the spare server simply replaces the failed server.

We consider three rebuild algorithms in the following sections. For simplicity, we assume K=1 and consider only single-server failure in the following analysis, but the analysis can be extended to cover multiple-simultaneous server failures. When one server fails, the (N_S − 1) remaining active servers will cooperate to rebuild the data into the spare server. Let U bytes be the storage capacity of each server, and S_S be the effective server transfer capacity. For simplicity, we assume that S_S is a constant value equal to the sum of rates a server sends and receives data. That is, if a server has an effective transfer capacity of S_S = 6MB/s, and the server is receiving data at a rate of 2MB/s, the server will only be able to send data up to 4MB/s.

If all data in the servers are stored in a backup storage, we can reload the backup data into a spare server. The rebuild rate will then be equal to the transfer capacity of the backup device or the spare server, whichever is smaller. For comparison with other rebuild algorithms, we assume the backup device has infinite throughput and capacity. Therefore the rebuild rate is bounded by the server throughput $R_{reload} = S_S$, which is also the maximum rate achievable by any algorithm. In practice, maintaining a video library backup and keeping it up to date would likely be expensive, therefore the rebuild algorithms proposed in the following sections do not require extra backup data.

2.1 Baseline Rebuild

In this algorithm, the remaining active servers send their stripe units to the spare server and the spare server computes the lost data. This method is similar to the baseline rebuild scheme in RAID [1]. For every lost stripe unit, a total of (N_S − 1) stripe units will have to be transferred to the spare server. Since the servers continue service during rebuild, the average available transfer rate from each server will be $S_S(1-\rho)$, where ρ is the utilization of the remaining active servers. The total rate of data transfer from the remaining $(N_S - 1)$ active servers is therefore

$$r = S_S(1-\rho)(N_S - 1) \qquad (1)$$

It is possible that r may exceed the transfer capacity of the spare server if ρ is less than $(1-/(N_S - 1))$. We have the following theorem relating the upper limit on the baseline rebuild rate.

Theorem 1 *The data rebuild rate of baseline rebuild, denoted by $R_{baseline}$ is bounded by the capacity S_S of the spare server and is given by*

$$R_{baseline} = \begin{cases} S_S/(N_S-1) & \text{for } \rho < (1-1/(N_S-1)) \\ S_S(1-\rho) & \text{for } \rho \geq (1-1/(N_S-1)) \end{cases} \qquad (2)$$

Proof: Please refer to [1]. ■

The rebuild time for a server with storage U is then given by

$$T_{baseline} = \frac{(N_S-1)U}{\min(S_S, S_S(1-\rho)(N_S-1))} \qquad (3)$$

2.2 Distributed Rebuild

In baseline rebuild, the transfer capacity of the spare server becomes the bottleneck even if the remaining active servers have unused capacity available. Another approach is to compute the lost data before transferring to the spare server. In this way, only the reconstructed data is sent to the spare server and hence the capacity of the spare server can be better utilized. We consider a distributed rebuild scheme where rebuild computation is distributed over all the remaining active servers. We divide the lost data into $(N_S - 1)$ equal-sized subsets, with each subset rebuilt by one of the remaining $(N_S - 1)$ servers. The active server responsible for a subset retrieves stripe units from the other $(N_S - 2)$ servers, computes the lost unit, and transfers the unit to the spare server for storage.

Note that the sum of transfer rates in or out of the working (N_S-1) servers is given by $S_S(1-\rho)(N_S-1)$. For each stripe unit rebuilt by a particular server, we have $(N_S - 2)$ transmissions from the remaining servers, $(N_S - 2)$ receptions into this server, plus one transmission from this server to the spare server. Therefore the rebuild rate $R_{distributed}$ to the spare server is given by

$$R_{distributed} = \frac{S_S(1-\rho)(N_S-1)}{2(N_S-2)+1} = \frac{S_S(1-\rho)(N_S-1)}{2N_S-3} \qquad (4)$$

Note that $R_{distributed}$ is always smaller than the capacity of the spare server, proved in the following theorem:

Theorem 2 *Under distributed rebuild, the rate of data transfer from the active servers to the spare server will never exceed the capacity of the spare server.*

Proof:
We note that $(N_S - 1) \leq 2(N_S - 3)$ for all $N_S \geq 2$, hence $R_{distributed} \leq S_S(1-\rho) \leq S_S$. ■

The rebuild time is given by

$$T_{distributed} = \frac{U}{R_{distributed}} = \frac{U(2N_S-3)}{S_S(1-\rho)(N_S-1)} \qquad (5)$$

2.3 Mixed Distributed-Baseline Rebuild

In distributed rebuild, the spare server is never fully utilized. This is due to the fact that for every stripe unit of data reconstructed, a total of $(2N_S - 3)$ stripe units need to be transferred among the remaining active servers. In baseline rebuild, the ratio is only $(N_S - 1)$, albeit at the cost of $(N_S - 1)$ times more capacity required at the spare server. This suggests a mixed algorithm which part of the data are rebuilt using distributed rebuild, and the rest by baseline rebuild.

To analyze the mixed distributed-baseline rebuild, we let μ $(0\leq\mu\leq 1)$ be the proportion of server capacity allocated for distributed rebuild, and $(1-\mu)$ for baseline rebuild. The total rebuild rate into the spare server is just the sum of the two processes

$$R_{mixed} = \frac{S_S(1-\rho)(N_S-1)\mu}{2N_S-3} + S_S(1-\rho)(1-\mu) \qquad (6)$$

To find the effect of μ, we differentiate Equation (6) with respect to μ

$$\frac{dR_{mixed}}{d\mu} = S_S(1-\rho)\left(\frac{N_S-1}{2N_S-3} - 1\right) \leq 0 \quad \forall N_S \geq 3 \qquad (7)$$

Therefore reducing μ always increases the total rebuild rate for three or more servers. That is, baseline rebuild is more effective unless it is bounded by the spare server capacity. When $N_S=2$, the rebuild rate is independent of μ and just equal to $S_S(1-\rho)$. To find the lower bound for μ, we use the condition that the aggregate rate of rebuild traffic going into the spare server cannot exceed S_S:

$$\frac{S_S(1-\rho)(N_S-1)\mu}{2N_S-3} + S_S(1-\rho)(N_S-1)(1-\mu) \leq S_S \qquad (8)$$

Solving for μ gives the lower bound

$$\mu \geq \frac{(1-(1-\rho)(N_S-1))}{(1-\rho)(N_S-1)\left(\frac{1}{2N_S-3} - 1\right)} \qquad (9)$$

Note that we have $u = 0$ for $\rho \geq (1 - 1/(N_S - 1))$.

Using this lower bound, the maximum rebuild rate is then given by

$$R_{mixed} = \begin{cases} \frac{S_S[1+(1-\rho)(N_S-1)]}{2(N_S-1)} & \text{for } \rho \leq (1-1/(N_S-1)) \\ S_S(1-\rho) & \text{otherwise} \end{cases}$$

(10)

The rebuild time can then be found accordingly.

2.4 Optimal Rebuild Rate

Having derived the rebuild rate for the proposed algorithms, it is interesting to see how far these algorithms are from the optimum. It turns out that under the current assumptions, the mixed distributed-baseline rebuild algorithm is optimal. To prove this, we note that the computation of lost units is done either at the spare server, the remaining ($N_S - 1$) servers, or partially done at both. Therefore we have the next lemma.

Lemma 1 *For any stripe unit rebuilt and stored into the spare server, we need ($N_S - 1$) transmissions from the remaining active servers.*

Proof: Please refer to [1]. ■

Each server contributes in the rebuild process either in the transmission or reception mode, or both. Let φ {$0 \le \varphi \le 1$} be the proportion of capacity each server used for transmission. The sum of capacities of the remaining servers used for transmission is given by $\varphi S_S(1-\rho)(N_S-1)$. On the other hand, the sum of capacities available for reception is equal to the sum of receiving capacities of the remaining servers plus the capacity at the spare server. We have therefore

$$\varphi S_S(1-\rho)(N_S-1) \le (1-\varphi)S_S(1-\rho)(N_S-1)+S_S \qquad (11)$$

Lemma 1 shows that the rebuild rate is proportional to the sum of transmission rates by the remaining servers. Hence to maximize the rebuild rate implies maximizing the transmission rate, subjected to the constraint in Equation (11). We can solve for φ by rearranging Equation (11) as

$$\varphi \le \frac{1+(1-\rho)(N_S-1)}{2(1-\rho)(N_S-1)} \qquad (12)$$

Note that φ = 1 for $\rho \ge (1 - 1/(N-1))$, we can obtain the optimal rebuild rate as

$$R_{\max} = \begin{cases} \dfrac{S_S[1+(1-\rho)(N_S-1)]}{2(N_S-1)} & \text{for } \rho \le (1-1/(N_S-1)) \\ S_S(1-\rho) & \text{otherwise} \end{cases} \qquad (13)$$

which is exactly equal to Equation (10). We state our result in the following theorem.

Theorem 3 *The mixed distributed-baseline rebuild scheme achieves the optimal rebuild rate and hence requires the minimum amount of rebuild time.*

Proof:
This follows directly from Equation (10) and Equation (11). ■

2.5 Rebuild Time

The previous sections focus on deriving the rebuild rate and time. It is clear that the rebuild time increases with server loading (or utilization) ρ. To control the rebuild time, the server should limit its server loading ρ and make available some capacity for the storage rebuild process. First of all, we note that the rebuild time will be minimum if ρ is zero, given by

$$T_{\min} = \frac{2(N_S-1)U}{N_S S_S} \qquad (14)$$

To complete the rebuild process using the mixed distributed-baseline scheme by time t ($t \ge T_{min}$), we can derive the server loading ρ from Equation (9)

$$\rho \le \begin{cases} 1+\dfrac{1}{N_S-1}-\dfrac{2U}{tS_S} & \text{for } T_{\min} \le t \le \dfrac{U(N_S-1)}{S_S} \\ 1-\dfrac{U}{tS_S} & t > \dfrac{U(N_S-1)}{S_S} \end{cases} \qquad (15)$$

Therefore by limiting the server loading to ρ, we can control the rebuild process to finish by time t.

3. NUMERICAL RESULTS AND DISCUSSIONS

We consider a RAIS VoD system with 5 active servers and one spare server. Each server has 4GB storage, including the redundant units. We assume a server transfer capacity of 2MB/s (16Mb/s) in accordance with our experimental results in [4]. Results for other server capacity and storage size can be obtained by scaling the numbers accordingly.

Figure 3 shows the rebuild rate versus server utilization for all rebuild schemes. We observe that for baseline rebuild, the rebuild rate is constant at $S_S/(N_S - 1)$ for $\rho \le (1-1/(N_S - 1))$, even if the remaining active servers are lightly loaded and so have more capacity available for rebuild. As the active server utilization approaches one, the rebuild rate drops quickly. Distributed rebuild performs better than baseline rebuild when the server utilization is low, but it deteriorates earlier when server loading increases. Mixed distributed-baseline rebuild gives the best performance under all loading conditions. In fact it achieves the optimal rebuild rate. For comparison, we also show the maximum achievable rebuild rate when the lost data is loaded from a backup device directly. It sets an upper bound on the rebuild rate that can be achieved in an ideal situation.

Figure 4 shows the time required for rebuild. We note that the time difference between mixed and distributed schemes is less significant at light loading, and there is no difference at high loading between the mixed and the baseline scheme. So instead of using a mixed baseline-distributed scheme, one can switch from distributed rebuild for low server utilization to baseline rebuild for high server utilization.

Figure 5 shows the rebuild time versus the number of servers in the RAIS system under a server loading of ρ=0.5. We observe that the rebuild time for baseline rebuild increases with more servers. This is because the amount of data units required to rebuild a lost unit increases with more servers. On the other hand, the rebuild time for distributed rebuild and mixed distributed-baseline rebuild remains roughly the same for 10 or more servers. This is because the rebuild capacity also increases with the number of servers. Hence distributed rebuild and mixed distributed-baseline rebuild is more scalable to large RAIS systems with many servers.

4. CONCLUSION

In this paper, we analyzed and compared three algorithms for rebuilding lost data in a RAIS VoD system. We derived the optimal rebuild rate and showed that the mixed distributed-baseline algorithm is optimal. We presented numerical results obtained using real-world parameters, and showed that fully automatic rebuild can be performed in reasonable time if the server loading is controlled properly.

REFERENCES

[1] J. Chandy, A. L. Narasimha Reddy, "Failure Evaluation of Disk Array Organizations," *In Proceedings of the 13th International Conference on Distributed Computing Systems*, pp. 319-26, IEEE Computer Society Press, 1993.

[2] Y.B. Lee, *VIOLA - Video on Local Area Networks*, PhD Dissertation, Department of Information Engineering, CUHK, 1997.

[3] T.D.C. Little, and D. Venkatesh, "Popularity-Based Assignment of Movies to Storage Devices in a Video-on-Demand System," *ACM Multimedia Systems*, 1994.

[4] P.C.Wong and Y.B.Lee, "Redundant Array of Inexpensive Servers (RAIS) for On-demand Multimedia Services," *ICC'97*, Montreal, Canada, June 8-12, 1997.

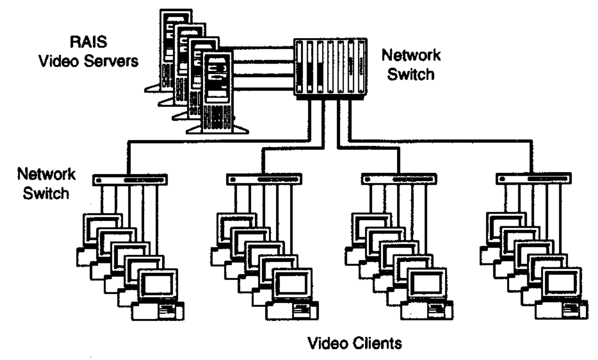
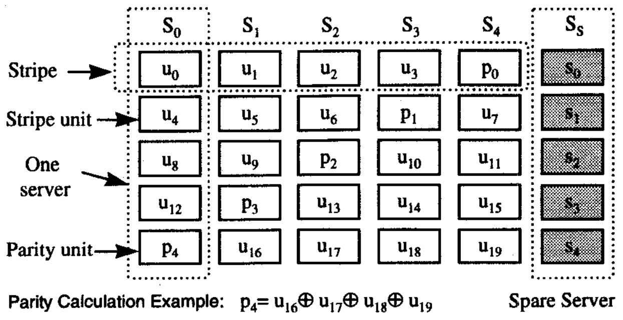


Figure 1. Architecture of a RAIS video-on-demand system.



S_0 S_1 S_2 S_3 S_4 S_S
Stripe
Stripe unit
One server
Parity unit
u_0 u_1 u_2 u_3 p_0 s_0
u_4 u_5 u_6 p_1 u_7 s_1
u_8 u_9 p_2 u_{10} u_{11} s_2
u_{12} p_3 u_{13} u_{14} u_{15} s_3
p_4 u_{16} u_{17} u_{18} u_{19} s_4
Parity Calculation Example: $p_4 = u_{16} \oplus u_{17} \oplus u_{18} \oplus u_{19}$
Spare Server

Figure 2. Storage configuration of a RAIS system.

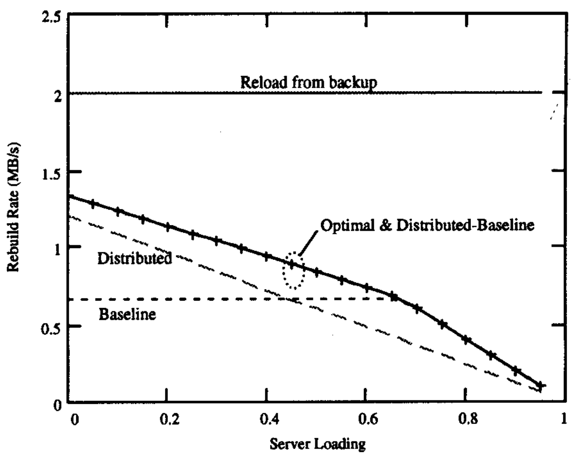


Figure 3. Rebuild rate versus server loading.

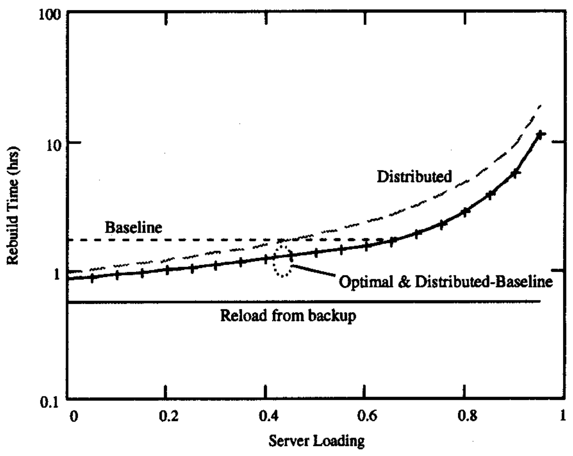


Figure 4. Rebuild time versus server loading.

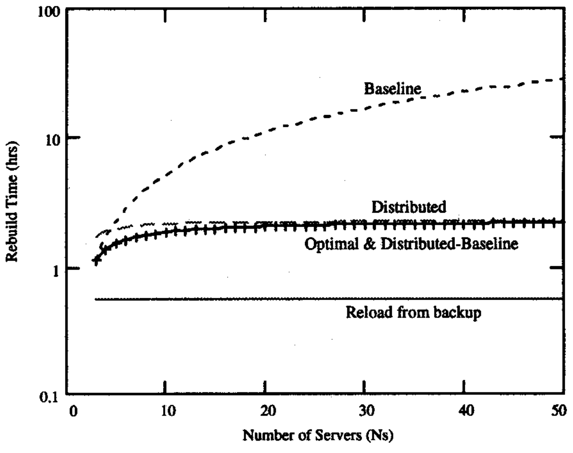


Figure 5. Rebuild time versus number of servers.