Balancing Workload and Recovery Load on Distributed Fault-Tolerant VOD Systems

Ing-Jye Shyu and Shiuh-Pyng Shieh, *Member, IEEE*

***Abstract*— This letter presents a new code sequence for dispatching playback jobs among distributed video-on-demand servers. The dispatch accordingly has the property of evenly distributing workload among all servers and balancing recovery load on surviving servers if one of the servers fails. In this letter we give an $O((n-1)!)$ algorithm for efficiently finding a dispatch sequence for n servers.**

***Index Terms*—Fault tolerance, load balancing, VOD system.**

I. Introduction and Motivation

FAULT TOLERANCE is considered to be an important issue for the design of video-on-demand (VOD) systems. This letter proposes a VOD system equipped with distributed video servers which provide server-level fault-tolerant capability. Fig. 1 depicts a typical system consisting of distributed VOD servers. Based on the system model, we present a recovery scheme which can tolerate server failure [5]. The idea is to allocate new playbacks on the surviving servers to refresh the failed playbacks of a failed server. A *playback* stands for a delivered video stream. Allocating a playback consumes many system resources (including disk bandwidth, buffers, and network bandwidth), therefore a *merging* technique is used to minimize the recovery cost (the merging operation was proposed by Golubchik [3]). If there exists another playback which delivers an earlier frame of the same video on the surviving server, the merging technique alters the progress rates of both playbacks so that they can be merged within a short time, and consequently the resources occupied by one of the playbacks can be released. The time spent in merging the two playbacks is determined by their distance. Therefore, in terms of the recovery overhead and the merging latency, the best candidate to perform the recovery is the video server with a playback nearest to the failed one. The goal of our design is to dispatch the playbacks to the distributed VOD servers such that the workload and recovery load can be minimized and evenly distributed among servers.

In order to provide the VOD services, one of the common mechanisms is to deliver the video stream periodically by offering staggered video start times [4]. Viewers are thus guaranteed a dedicated video from the beginning within a waiting time no greater than the difference between two successive start times. With the consideration of workload balance, the problem of staggering video playbacks on a group of video servers must be dealt with. An intuitive solution is to dispatch playbacks with different start times to the video servers in a round-robin manner. Fig. 2(a) gives an example of the round-robin dispatch on a four-server system, in which a video starts every 5 min. In Fig. 1, P_i represents a playback and S_i a server. The round-round dispatch results in balanced workload among the video servers.

The round-robin dispatch policy entails a drawback when fault tolerance is of concern. When a server fails, all playbacks on the failed server will be recovered through the recovery schemes described in the previous context for reducing the overheads. For example, in Fig. 2(a), assume that S_2 fails, playbacks P_2, P_6,$\cdots$, P_{4N+2} will be recovered by allocating

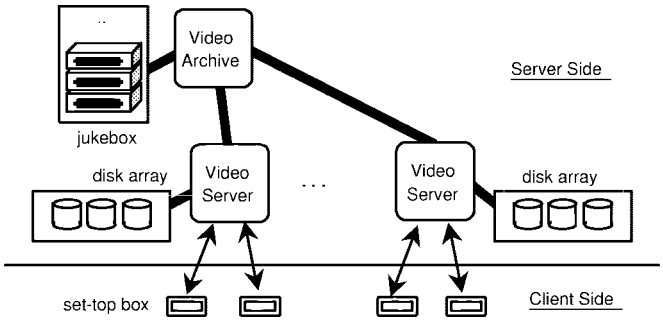

Fig. 1. The architecture of a multiserver distributed VOD system.

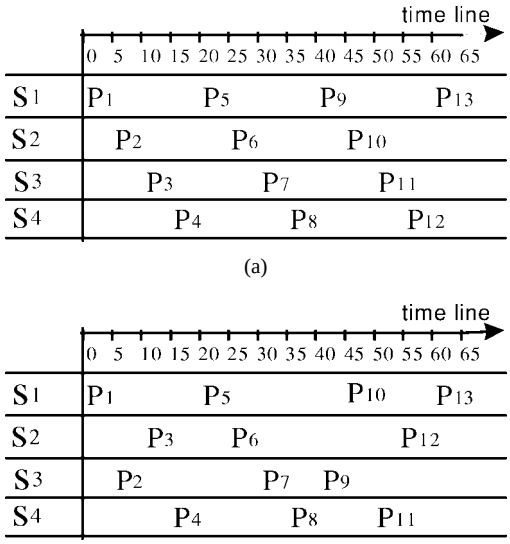

Fig. 2. (a) The round robin dispatch. (b) The balanced N-cyclic dispatch.

Manuscript received November 19, 1997. The associate editor coordinating the review of this letter and approving it for publication was Prof. V. S. Frost.

The authors are with the Department of Computer science and Information Engineering, National Chiao-Tung University, Hsinchu, Taiwan 30010, R.O.C.

Publisher Item Identifier S 1089-7798(98)08064-8.

temporary playbacks on S_3, because S_3 has playbacks nearest to these failed playbacks. This causes recovery load imbalance because the recovery process, including the allocation of the temporary playbacks and the computing powers to merge playbacks, is solely performed by S_3. However, if playbacks are dispatched to servers according to the sequence 1, 3, 2, 4, 1, 2, 3, 4, 3, 1, 4, 2 instead of the round-robin manner, as shown in Fig. 2(b); in the normal state, every server has the same workload distribution as in the round-robin dispatch. When a server failure occurs, e.g., S_2 fails, the temporary playbacks for recovering failed playbacks P_3, P_6, and P_{12} can be allocated on servers 4, 3, and 1, and merged with P_4, P_7, and P_{13}, respectively, such that the recovery load is minimum and evenly distributed. This kind of the dispatch sequence results in balanced workload both in the system normal state and in the failed state. We refer to this dispatch sequence as the *Balanced n-Cyclic Code* (BnC code).

In Section II we present a heuristic algorithm for efficiently finding a BnC code in time $O((n-1)!)$ for a given n. Section III shows the performance improvement with the algorithm. Section IV summaries the problems that merit further investigation.

II. Heuristic Algorithm

Balanced n-Cyclic code is defined as a sequence which is ordered to meet the following criteria: **C1**: be a cyclic code, **C2**: be divisible into $n-1$ equal-sized segments, SE, such that each segment consists of number 1 to n, and **C3**: for any two continuous elements in any two segments, i.e., SE_k = $(\cdots e_i e_{i+1} \cdots)$ and SE_l = $(\cdots e_j e_{j+1} \cdots)$, if $e_i = e_j$, then $e_{i+1} \neq e_{j+1}$, where $1 \leq l, k \leq n-1$, and $l \neq k$. We prove in the Appendix that finding a BnC code is a NP-complete problem. Thus, we need an efficient algorithm to produce BnC sequences. In this section we present the find-BnC-code algorithm which can generate a BnC code from a B($n-1$)C code. The technique is to insert the number n into some special positions in the B($n-1$)C code to form a BnC code. Here we use some terms from graph theory. The arc(v, w) is a directed edge from the vertex v to the vertex w. A *Hamilton cycle* is a path that passes through every vertex exactly once and returns to the start vertex.

Algorithm*:* Find-BnC-code (**Input:** B($n-1$)C code; **Output:** BnC code)

Begin

Step 1: Represent the input B($n-1$)C code as a series of connected arcs, (e_1, e_2), (e_2, e_3), $(e_{(n-1)\times(n-2)}, e_1)$, in which e_i is the ith number in the B($n-1$)C code.

Step 2: Partition the arcs in Step 1 into $n-1$ groups in which each group R_i consists of arcs from $(e_{(n-1)\times(i-1)+1}, e_{(n-1)\times(i-1)+2})$ to $(e_{(n-1)\times i+1}, e_{(n-1)\times i+2})$, where $1 \leq i \leq n-2$.

Step 3: Select one arcs (v_i, w_i) from every R_i such that all v_i are distinct, where $1 \leq i \leq n-2$.

Step 4: Step 3 produces $(n-1)!$ combinations. For each combination, two arcs (n, w_x) and (v_x, n) are generated, where $1 \leq v_x, w_x \leq n-1$, $v_x \neq w_x$, $v_x \notin \{v_i | 1 \leq i \leq n-2\}$ and $w_x \notin \{w_i | 1 \leq i \leq n-2\}$. Then check these n arcs to see whether they form a Hamilton cycle. If yes, go to Step 5. Otherwise, go to Step 3 and try the next combination. When all the combinations have been tested and no Hamilton cycle is found, it means that the algorithm cannot find a BnC code from the input B($n-1$)C code.

Step 5: Extend the B($n-1$)C code into a BnC code by

a) inserting number n at the place located at between v_i and w_i in the B($n-1$)C code, where (v_i, w_i) is the arcs selected in Step 3 and $1 \leq i \leq n-2$.

b) concatenating the formed Hamilton cycle (Step 4) to the tail of the B($n-1$)C code to form a BnC.

End

An example is given to illustrate the algorithm. Assume that a B5C code is {1, 2, 3, 4, 5, 2, 4, 1, 3, 5, 3, 1, 4, 2, 5, 4, 3, 2, 1, 5}. Partition it into four arc groups, R_1^5 =(1, 2), (2, 3), (3, 4), (4, 5), (5, 2), R_2^5 =(2, 4), (4, 1), (1, 3), (3, 5), (5, 3), R_3^5 =(3, 1), (1, 4), (4, 2), (2, 5), (5, 4), R_4^5 =(4, 3), (3, 2), (2, 1), (1, 5), (5, 1). By exploring the computations in Steps 3 and 4, we find that arcs (3, 4), (1, 3), (4, 2) and (5, 1) (respectively selected from groups R_1^5, R_2^5, R_3^5, and R_4^5) as well as two other generated arcs (6, 5) and (2, 6) can form a Hamilton cycle {1, 3, 4, 2, 6, 5}. We then insert number 6 into the B5C code at the positions between 3 and 4, 1 and 3, 4 and 2, and 5 and 1, then concatenate the formed Hamilton cycle {1, 3, 4, 2, 6, 5} to its tail. The B6C code thus obtained is {1, 2, 3, 6, 4, 5, 2, 4, 1, 6, 3, 5, 3, 1, 4, 6, 2, 5, 4, 3, 2, 1, 5, 6, 1, 3, 4, 2, 6, 5}.

Step 5a) extends the B($n-1$)C code by removing arc (v_i, w_i) and inserting arcs (v_i, n) and (n, w_i). Since the inserted arcs (v_i, n) and (n, w_i) are disjoint for all $1 \leq i \leq n-2$, and the removed arcs (v_i, w_i) are also disjoint from the inserted B($n-1$)C code, the code formed by Step 5b) is a BnC code. In Step 4, the worst case of finding a successful combination needs $(n-1)!$ iterations, so the complexity for finding a BnC code is $O((n-1)!)$. However, the heuristic algorithm may fail to find a BnC code from some B($n-1$)C code. In this case, we can generate another B($n-1$)C code as a seed to repeat the above procedures.

III. Performance Evaluation

We have developed a program that uses the above B5C code as a seed to find other BnC codes. The B5C seed is generated by exhausting all the combinations. Our experiments showed that our heuristic approach can find a BnC code much faster than the exhausting search. Table I shows the time needed for searching a BnC code by comparing the exhausting search with the find-BnC-code algorithm running on an Intel Pentium-90 machine.

IV. Conclusions

The playback dispatch according to a balanced n-cyclic code has two merits: 1) the balanced workload distribution among video servers and 2) the balanced recovery load distribution among surviving servers while some video server fails.

TABLE I
RESULTS OF THE COMPARISON

	Exhausting Search	Find-BnC-Code Algorithm
B6C	$\cong$ 4.5 hours	$\leq$ 0.1 s
B7C	$\cong$ 17.2 hours	$\leq$ 0.1 s
B8C	-	$\leq$ 0.1 s
B10C	-	$\leq$ 0.1 s
B20C	-	$\cong$ 0.1 s
B30C	-	$\cong$ 0.5 s
B40C	-	$\cong$ 295 s
B50C	-	$\cong$ 745 s

-: more than one day

However, the configuration after tolerating a server failure can no longer provide balanced recovery load distribution while a second server failure occurs. In the near future, we hope to find a special code sequence which makes the recovery load distribution near balanced while recovering the second or more server failures.

APPENDIX

Theorem 1: For any odd n, a BnC code exists and finding it is an NP-complete problem.

Proof: A complete graph with n vertices is a graph of degree $n-1$. According to related theorems [1], [2], [6], if there exists an algorithm that can effectively find $(n-1)/2$ disjoint Hamilton cycles for a complete graph with odd-n vertices, we can find $n-1$ disjoint Hamilton cycles for a complete symmetric digraph. These $n-1$ disjoint Hamilton cycles can be concatenated into a BnC code. Thus the problem of finding $(n-1)/2$ disjoint Hamilton cycles for a complete graph with minimal degree n is reducible to the problem of finding a BnC code when n is odd. We can conclude that a BnC code exists for any odd n, and also that finding a BnC code is an NP-complete problem. ■

REFERENCES

[1] B. Bollobas and A. M. Frieze, "On matchings and Hamiltonian cycles in random graphs," *Ann. Discrete Math.*, vol. 28, pp. 23–46, 1985.
[2] B. Bollobas, T. I. Fenner, and A. M. Frieze, "An algorithm for finding Hamilton paths and cycles in random graphs," *Combinatorica*, vol. 7, pp. 327–341, 1987.
[3] L. Golubchik, C. S. Lui, and R. Muntz, "Adaptive piggybacking: A novel technique for data sharing in video-on-demand storage servers," *Multimedia Syst.*, vol. 4, no. 3, pp. 140–155, 1996.
[4] T. D. C. Little and D. Venkatesh, "Popularity-based assignment of movies to storage devices in a video-on-demand system," *Multimedia Syst.*, vol. 2, no. 6, pp. 280–287, 1995.
[5] I. J. Shyu and S. P. Shieh, "Distributed fault-tolerant design for multiple-server VOD systems," *Multimedia Tools Appl.*, to be published.
[6] ——, "The load-balanced playback dispatch for fault-tolerant multi-server VOD systems," in *Proc. 3rd Workshop on Real-Time and Media Systems*, Taipei, Taiwan, R.O.C., pp. 165–170, 1997.