中央研究院
資訊科學研究所
Institute of Information Science, Academia Sinica • Taipei, Taiwan, ROC

TR-IIS-05-023

Real-Time Scheduling by Cascading

Ray-I Chang, Ruei-Chuan Chang, Jan-Ming Ho

December 2005 || Technical Report No. TR-IIS-05-023

http://www.iis.sinica.edu.tw/LIB/TechReport/tr2005/tr05.html

Real-Time Scheduling by Cascading

Ray-I Chang
Department of Engineering Science
National Taiwan University
Taipei, Taiwan, ROC

Ruei-Chuan Chang[1]
Institute of Inform Science
Academia Sinica
Taipei, Taiwan, ROC

Jan-Ming Ho
Institute of Inform Science
Academia Sinica
Taipei, Taiwan, ROC

Abstract*

Real-time scheduling is important for modern computer systems to support increasingly popular multimedia applications. In this paper, we consider a simple video server that receives the commands from clients and sends the video data without error occurred. In the video server, there is one parent thread for dealing with the arrival video tasks. After receiving a new video task from a client, a new child thread is created to determinate the real-time requirements for disk access. Based on the results of traffic smoothing and the sizes of disk blocks, the child thread sends real-time disk requests to the system. Then, a multi-segment cascade scheme is applied to implement the real-time disk scheduling routine on UnixWare operating systems. Both the simulation results and the implementation results are presented for comparisons. The same idea may be extended to handle the real-time scheduling problem with both video server and video proxy [30].

---

[1] *He is also with Dept of Computer Science, National Chiao Tung University, HsinChu, Taiwan, ROC.*

1. Introduction

In this paper, we consider a simple video server that receives the commands from clients and sends the video data without error occurred. As shown in *Fig.* 1, given a media stream, a suitable network transmission schedule [5] can be determined to minimize the required buffer and bandwidth.

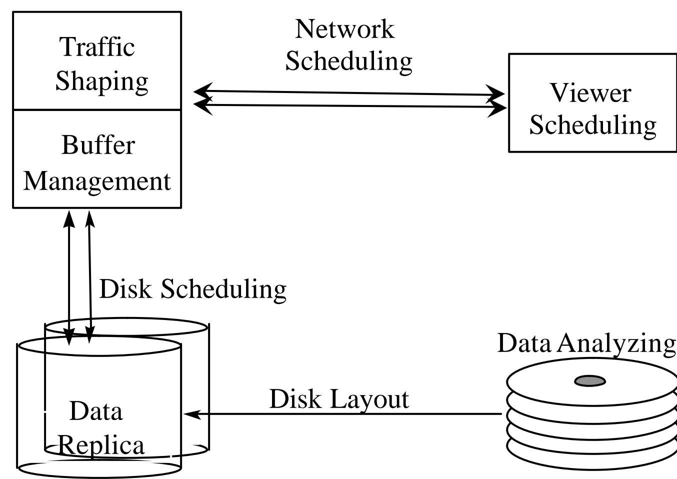

Fig. 1. *The workflow of multimedia system design.*

In the video server, there is one parent thread for dealing with the arrival video tasks. After receiving a new video task from a client, a new child thread is created to determinate the real-time requirements for disk access. A real-time disk task T_i can be denoted as (r_i, d_i, a_i, b_i) where r_i is the release time, d_i is the deadline, a_i is the track location and b_i is the data capacity [28]. The release time and deadline are decided from the network transmission schedule to prevent buffer overflow and underflow [5]. In this paper, based on the results of traffic smoothing [5] and the sizes of disk blocks [8], a sequence of real-time disk tasks is generated as follows.

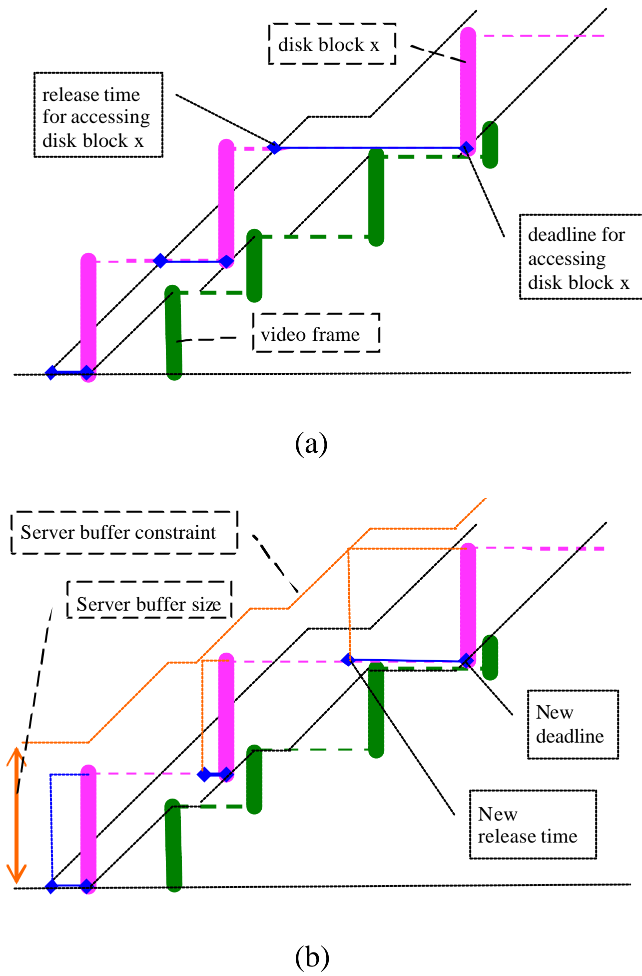

Fig. 2. A sequence of real-time disk tasks is generated.

(a) Without server buffer constraint. (b) With server buffer constraint.

In real-time disk scheduling (RTDS), we consider a set of real-time disk tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$ where n is the number of tasks. A RTDS algorithm is defined as a process Z to construct a schedule $T_{Z(0)}T_{Z(1)}\ldots T_{Z(n)}$ to maximize disk throughput under real-time constraints [15]. Notably, in RTDS, tasks need to be served not only with high throughput but also with guaranteed timing requirements [2]. To satisfy real-time constraints, the task's start-time should not be earlier than its release time and the task's fulfill-time should not be later than its deadline.

Although SCAN [7] is used in many operating systems for disk scheduling [1], it does not consider timing constraints [9] and is not suitable for serving real-time disk tasks [6]. The EDF (earliest-deadline-first) method is good for scheduling real-time tasks under the assumption of i.i.d. and exponential service times [10-11]. However, the service time for each task in RTDS depends on the initial read/write head position as well as the scheduling results. When T_i is just served after task T_j, the disk-head should move from the track location a_j to track location a_i to retrieve the data block with size b_i. The start-time is $e_i = \max\{ r_i, f_j \}$ and the fulfill-time is $f_i = e_i + c_{j,i}$. Note that the execution time $c_{j,i}$ of task T_i depends not only on the disk parameters [26-27] but also on the schedule result -- the pre-specified sequence T_jT_i. Since the independent assumption no longer holds, the employment of EDF results in poor data throughput.

In these years, different RTDS methods are proposed to combine the seek-optimizing scheme and the real-time scheduling scheme [18-23]. In this paper, we focus on the implementation of RTDS functions on a UnixWare 2.01 operating system by a multi-segment cascade scheme. The RTDS algorithm considered for implementation is based on BFI (best-fit-insertion) with some modifications.

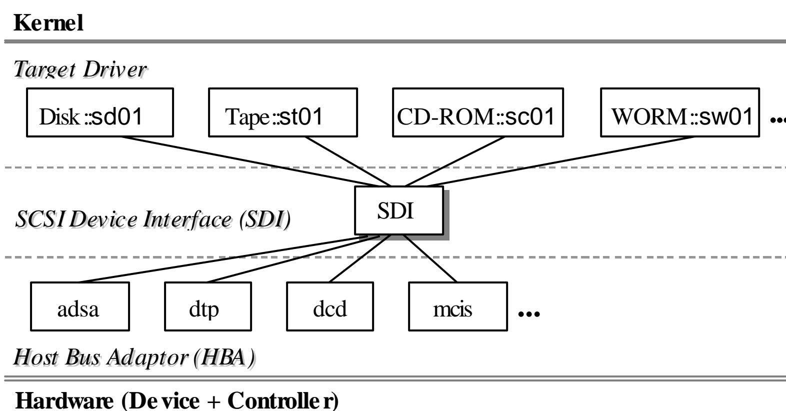

Fig. 3. The basic architecture of the portable device interface.

2. Disk Service Routines on UnixWare

In this paper, our implementations are worked on portable device interface (PDI) [25] of a SCSI hard disk driver. It takes the advantage that our implementation can be easily ported to other systems. In this section, we provide a high-level description of PDI. As shown in *Fig.* 3, the PDI architecture consists of three logical layers (from kernel to hardware units): *the target drivers*, *the SCSI device interface (SDI)*, and *the host bus adapter (HBA) drivers*. Generally, a hardware unit contains a physical device and a controller card. PDI bases on this architecture to break a device driver into a device-specific portion (said the *target driver*) and a controller-specific portion (said the *HBA driver*). Connections between these two portions are formed by SDI to provide a consistent and complete interface.

Target Drivers: The major functions of target drivers are to accept I/O tasks from kernel, and construct jobs to be sent to the HBA controllers. It is the interface between kernel and PDI. *Fig.* 3 shows four types of target drivers: sd01, st01, sc01, and sw01. They are related to hard disks, tapes, CD-ROM devices, and WORM (write-once-read-many) devices, respectively. Note that target drivers deal with only the device-specific aspects of the related hardware. They tackle the details of operating the physical device, and control the device behavior to decide what actions are needed for passed tasks. For example, the hard disk sd01 driver does not only understand how to perform operations onto a hard disk. Besides, it also imposes the data layout standard of hard disk used for various operating systems. On UnixWare, the main dominant data structure of the sd01 target driver is the "disk" structure. It contains information pertaining to a specific hard disk drive, such as the job queue, the state information, the VTOC (volume-table-of-content), the partition data, and the bad block information. For each physical disk configured in the system, there should be one disk structure.

SCSI Device Interface (SDI): SDI can be viewed as a pipeline to connect the target driver and the related HBA driver as shown in *Fig.* 3. (SDI is originated from an interface for supporting a variety of SCSI devices. However, the current SDI can support both SCSI and non-SCSI devices.) The major controlling data structures of SDI are the table of HBA entries (HBA_tbl) and the table of target driver entries (equipped device table, EDT). These two tables provide a set of interfaces that enable target drivers to issue commands to related HBA drivers. On the other hand, HBA drivers are allowed to respond to target drivers by SDI.

Host Bus Adapter (HBA): The HBA drivers handle the controller specific aspects of the device operations. Through the HBA_tbl table with well-defined interfaces, entry points into the related HBA driver can be easily accessed from SDI. *Fig.* 3 presents four different types of controllers, adsa, dcd, mcis, and dpt. They are corresponding to Adaptec SCSI-2742AT, directly-coupled device, IBM SCSI, and distributed processing technology, respectively. The basic functions of HBA are to send a set of commands to the controllers to monitor the processing flow of job. For example, considering the reading of a data block from a hard disk, the target driver will determine which actual physical sector on the hard disk will be read. When the actual physical sector provided by the target driver is applied, the HBA driver can program the disk controller registers and issues the read operation to generate the I/O interrupt. Whenever the completion of the operation is signaled, the operation result (whether successful or not) is passed back. Note that, in HBA, each task job is addressed to one of the connected hardware devices. The addressing scheme is different for different types of controllers specified. For example, in a SCSI controller, a single adapter can control up to 7 SCSI targets. Each of the SCSI targets may consist of up to 8 logical unit devices. Thus, a complete SCSI address would consist of the controller, the target and the logical unit number (LUN). This three-dimensional address (controller, target, LUN) is applied for determining the correct logic unit of a SCSI controller.

As our algorithms are designed for real-time scheduling of hard disks, our descriptions focus on the target driver sd01. Through sd01, the most common path is the sd01strategy() routine as shown in Fig. 3. Based on the related disk structure of the input task, the size and position of the input task is validated by the sd01strat0() routine. At first, the bad blocks are re-mapped by the sd01ck_badsec() routine. Then, by calling the sd01gen_sjq() routine, a data structure "job" for this input task job is allocated. For the creation of bus task, the sd01strat1() routine is called to take the buffer header to this job structure. At last, the job structure is linked onto the job queue of the disk structure and sent to the SDI driver by the sd01send() routine.

By the sdi_send() routine, the target drivers pass the tasks linked onto the job queue to SDI. Note that, although the target drivers need only the type of operation to be performed (for example, *read* a block from a hard disk), the HBA drivers will need to know the specific details of how a command is issued to the related controller (for example, the format of SCSI commands). SDI serves PDI in one critical role to act as a "router" to ensure that each disk task from the target driver is passed to the correct HBA driver. Furthermore, responses of the HBA drivers are also ensured to return to the correct target drivers. While a task is sent to SDI, the controller specific data structure is created. In this paper, the applied HBA driver is a representative controller adsa for the Adaptec SCSI-2742AT driver. In the case of the adsa HBA driver, the adsagetblk() routine is called to allocate a SCSI task block (srb). This routine is necessary for passing the controller dependent arguments to the related controller. With this srb structure, SDI initializes a pointer to link back to the job structure and passes the task to the related HBA driver.

The most common path through the adsa HBA driver is the adsasend() routine. In adsasend(), the HBA driver first queues the received srb's on the logical unit queue scsi_lu by adsa_putq(). Then, the adsa_next() routine is called to operate the disk scheduling algorithm on scsi_lu and get

the first task in queue. On UnixWare, there are three types of srb's: SCB (the SCSI command block, such as the read/write commands), ISCB (the immediate SCB, such as the resuming and the suspending of the hard disks), and SFB (the SCSI function block, such as the block reassigning of the hard disks). They have the priorities: SFB > ISCB > SCB. As shown in *Fig.* 4, different type of srb is put to different section of the scsi_lu queue. With the highest priority, SFB's are immediately executed in the order they are received. If there is no SFB, the earliest ISCB is selected for executing. They are presented with the FIFO order.

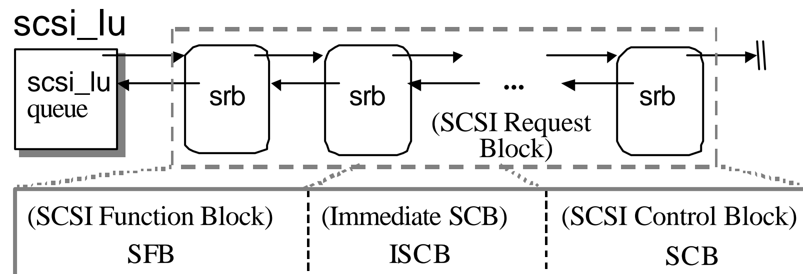

Fig. 4. *Different types of SCSI task blocks are put into different sections.*

3. Multi-Segment Cascading

In this paper, a multi-segment cascade scheme is proposed to support RTDS on UnixWare. Our implementation is worked on the PDI architecture. It is portable for different operating systems. In our implementation, only the adsa_putq(), adsa_schedule() and adsa_getq() routines are modified to consider two additional srb types: time-constrained SCB (TSCB) and rejected SCB (RSCB). Excepting the extension of the job data structure with additional real-time parameters (release time and deadline), no data structure is changed. Descriptions about our adsa_putq(), adsa_schedule() and adsa_getq() routines shown as follows.

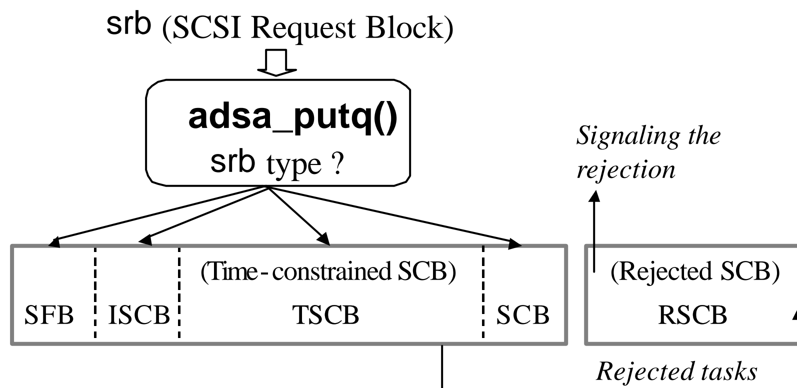

Fig. 5. The new adsa_putq() routine.

adsa_putq() -- Our adsa_putq() routine follows the priorities SFB > ISCB > TSCB > SCB to put srb's into different sections of scsi_lu (see *Fig.* 5). Note that ISCB and SFB are used only for the reconfiguration of hard disks (i.e. disk failure, mount or unmount). Whenever ISCB and SFB are presented, the disk configuration is changed and all the disk requests based on the old disk configuration are invalid. On UnixWare, an exception handler routine is provided to serve these special tasks. Since ISCB and SFB will not happen in normal applications, we can focus on the scheduling of normal disk tasks such as TSCB (real-time) and SCB (non-real-time).

adsa_schedule() -- Based on the *multi-segment cascade scheme*, we implement the BFI approach on UnixWare with some modifications to schedule both real-time and non-real-time disk tasks. As shown in *Fig.* 6, there are three segments in the TSCB section: the RTDS (real-time disk scheduling) segment, the EDF segment, and the FIFO segment. Whenever a new real-time disk task is presented, Our adsa_putq() routine first append it to the end of the FIFO segment. As the original processing flow of adsa_schedule(), tasks in the FIFO segment are selected for scheduling. In our adsa_schedule() routine, tasks in the FIFO segment are scheduled into the EDF segment by the adsa_schedule2e() routine. Then, tasks in the EDF segment are scheduled into the RTDS segment

by the adsa_schedule2r() routine. The processing flow is like a cascade as shown in *Fig*. 6. Note that, under a hard real-time constraint, the input task may not be successfully scheduled. If a request is rejected, we need to send a message to the client. However, this signaling task may take some system resources that are critical for the task schedule routine and the disk access routine. In this paper, we mark these rejected real-time tasks as non-real-time and use a low-priority RSCB queue to store them (see *Fig*. 5). Note that, although the non-real-time tasks have a lower priority than real-time tasks, they still need to be served as soon as possible (if the system resources are available). This problem is considered in our adsa_getq() routine.

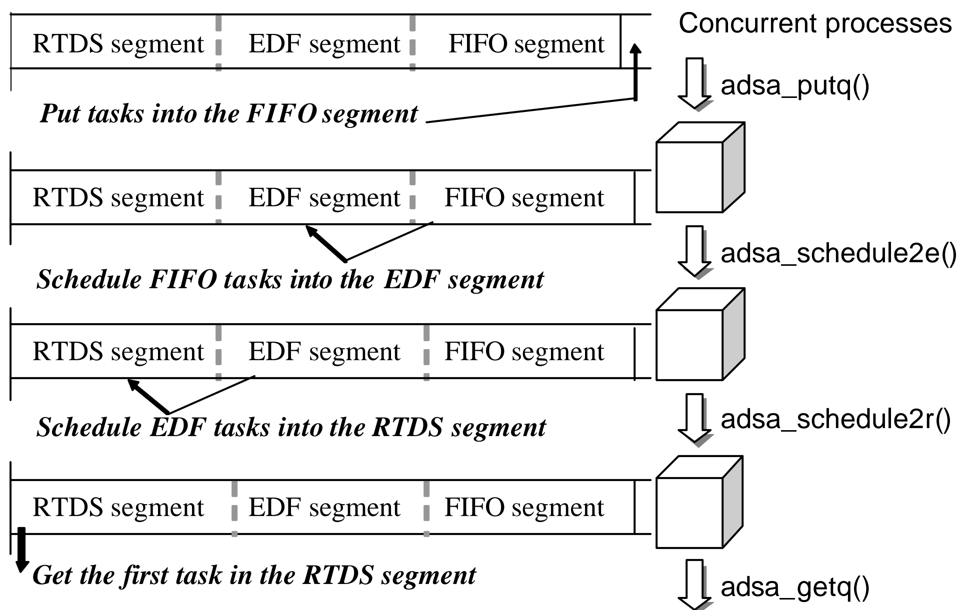

Fig. 6. *The processing flow of a multi-segment cascade scheme.*

adsa_getq() -- Since the original adsa_getq() routine simply gets the first job in the scsi_lu queue for execution, non-real-time tasks may be starved. This routine cann't fairly support non-real-time tasks. To avoid the building of a specialist real-time system (that is isolated from the

standard application environment), we need to design a new adsa_getq() routine for fairly supporting both real-time and non-real-time requests. In our adsa_getq() routine, we make a competition between the first SCB job and the first RTDS job. The first SCB job is selected for service if all the real-time requirements of the RTDS jobs are guaranteed after executing this SCB job. For example, tasks $T_0T_1...T_n$ are in the RTDS segment and T_x is the first job in the SCB section. We want to decide if $T_xT_0T_1...T_n$ is a feasible schedule. When the pre-computed maximum tardiness [19] is applied, the above decision can be easily determined. The same idea can be applied for selecting a RSCB job to signal its rejection. Since the non-real-time tasks can be served as soon as possible, *our scheme can fairly support real-time and non-real-time applications.*

In modern computer architectures, the disk device (for the data access) and the CPU (for the schedule process) can be executed concurrently if they do not read/write the same data. The data throughput can be improved by efficiently concurring the I/O processes and the CPU processes. In our implementation, a multi-segment cascade scheme is proposed to reduce the wasted time in exclusive locks. Based on our scheme, we have exclusive locks between each two cascaded routines (adsa_putq(), adsa_schedule2e(), adsa_schedule2r() and adsa_getq()) that may read/write the same data (the FIFO, EDF, and RTDS segments, respectively). The exclusive lock for the disk I/O of adsa_getq() is only on adsa_schedule2r(). It is much smaller than the entire adsa_schedule() routine. When adsa_putq() and adsa_schedule2e() are executed, disk I/O can be executed without locks.

Note that *the waste time for I/O exclusive lock (called I/O lock time) depends only on the schedule operations used in* adsa_schedule2r(). Since BFI needs to test all rescheduling points in the RTDS segment, we can replace it by a first-fit-insertion (FFI) policy. These two methods have similar performances in disk throughput [22]. However, the average I/O lock time for FFI is only half of that

for BFI. Besides, since adsa_putq() gets only the first task in queue for execution, we can further reduce the I/O lock time by locking only the first task in queue as shown in *Fig.* 7. The same idea can be used in other routines. Thus, the exclusive section contains only one comparison operation. The I/O lock time can be ignored. Based on our processing flow, the I/O process is idled only when the task queue is empty. There is no waste time for the I/O process.

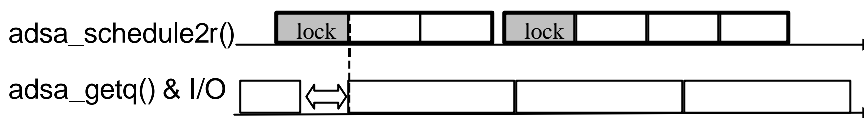

Fig. 7. *An example of our exclusive lock scheme.*

4. Implementation Results

For handling these real-time disk tasks, we use a multi-segment cascade scheme to implement a RTDS algorithm. Our implementation runs on a platform with Pentium-75 PCs under UnixWare 2.01. The applied DASD device is Seagate ST-31200N hard disks [17] with Adaptec SCSI-2742AT control cards [14]. A disk array system with D identical disk devices is considered. For each disk, a 200/D Kbytes/s bandwidth is required for supporting one MPEG-I stream. In our experiments, the disk layout strategy applied is a striping technique. It can also be applied to the round-robin data placement or other disk layout schemes [12-13] with minor modifications. For fair comparisons, we assume that the disk tasks are randomly arrived and uniformly distributed over the disk without data replication and task migration [8]. The same task sets are presented to different solution approaches. We also test the system with different arrival rates and different block sizes. Both the throughput improvement and the bandwidth utilization are compared. Our measurements show that our algorithm takes short disk service time. Furthermore, it also obtains high utilization in the disk bandwidth. Note

that the required service time of a disk task should include not only the data access time but also the task control time (i.e. the I/O lock time). To guarantee the timing requirements, the approximated service time may larger than the required service time. We use the worst-case service time as the service time model.

To evaluate the system performance, we consider different block sizes *B* (KB, the read data) for disk access. Each test case is computed by 100 examples, and each example contains 600/*B* tasks. Therefore, the total requirements of disk bandwidth are the same 600 KB/s for different test cases. *Fig.* 8 shows the average throughput improvements with different block sizes *B*. To simplify the effect of DMA, the value of block size is at most 60 KB (the maximum DMA size). We measure the improvement by comparing the obtained disk throughput with the best-known SCAN-EDF method. Experiments show that we have a high improvement in disk throughput when the number of input task is large. Although the utilization of disk bandwidth is decreasing as shown in *Fig.* 9, the decreasing rate is not large. There is 40% bandwidth utilization when serving 30 disk tasks. Our approach is more suitable than SCAN-EDF in handling large size RTDS problems.

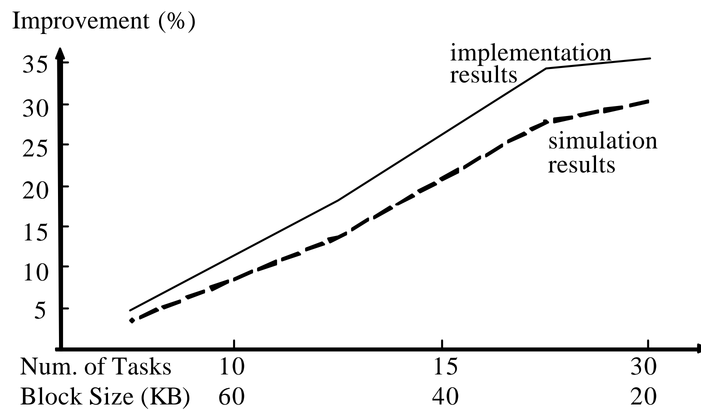

Fig. 8. *Compare the improvement of disk throughput.*

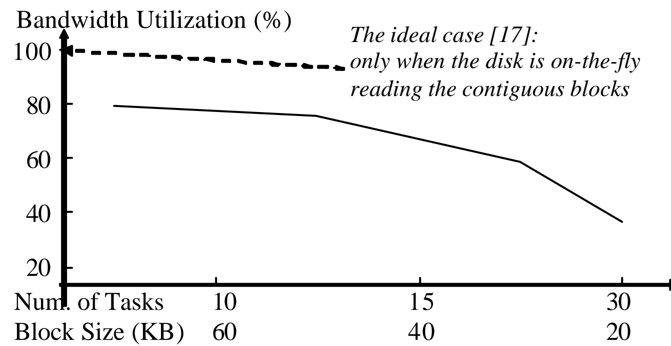

Fig. 9. Compare the utilization of disk bandwidth.

Note that our implementation result is always better than our simulation result. For example, with 30 input tasks, our approach can obtain 34% improvement in implementation. However, our simulation result has only a 27% improvement. The capital reason for this difference is the over-estimation of the approximated service time. Another reason is probably the effect of internal disk cache. Current disk devices usually have an internal cache (usually, to store the data at the same track or at the neighboring tracks under the disk head). On UnixWare, whenever an srb is selected to send to the hardware controller, a new controller command block (the ccb data structure) is allocated to point to this srb. The scatter/gather capability of controller is considered while allocating the ccb's. The adsa HBA driver determines whether multiple srb's can be combined to take further advantage. If multiple srb's can be combined together, the adsa HBA driver would build a scatter/gather list. In this case, a controller command block (the ccb data structure) is allocated to point to the scatter/gather list. Otherwise, a single srb is constructed as a ccb structure. Thus, if the data required for a disk task is just at the internal cache, the required service time can be very short. Note that the internal cache of hard disk has no effect if the block size is large enough. As shown in *Fig.* 8, the simulation result is closing to the implementation result when the block size is increasing.

5. Conclusion

In this paper, we consider only an array of homogeneous disks. However, modern video server is usually a hierarchy system with disks and tertiary mass storages. It would be interesting to consider a heterogeneous and hierarchy storage system. This system needs to consider not only real-time scheduling on different storages but also server scalability and data availability. It also contains the task sequence problem for read/write data consistence. In the paper, we consider only hard real-time tasks. However, in a soft real-time application [4], tasks may be batched [3] with deferred deadlines [29]. Our future work is to extend our system to handle soft real-time requirements. Besides, we also need more studies on the effect of internal disk cache and DMA to yield a more accurate service time model. The same idea may be extended to handle the real-time scheduling problem with both video server and video proxy [30] (as shown in the following figure).

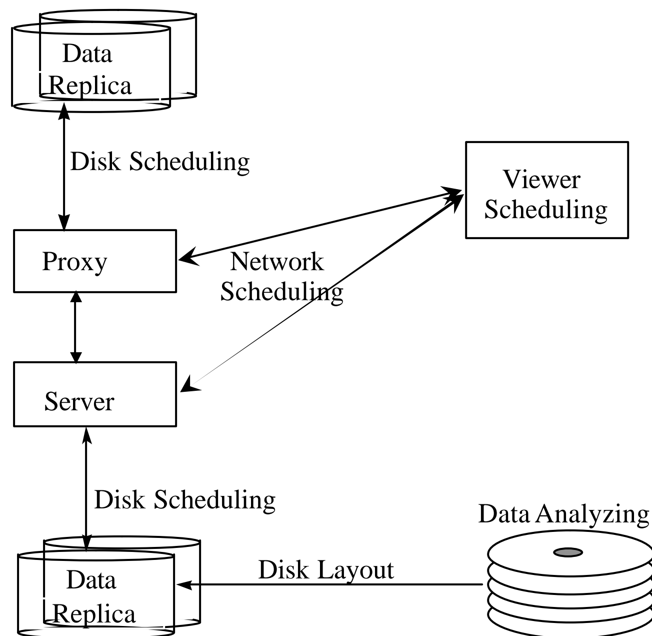

Fig. 10. The workflow of multimedia system design.

References

[1] J.L. Peterson and A. Silberschatz, Operating System Concepts, 2nd Edition, Addison-Wesley, 1985.

[2] D.P. Anderson, Y. Osawa, and R. Govindan, "A file system for continuous media," *ACM Trans. Comp. Systems*, vol. 10, no. 4, pp.311-337, 1992.

[3] A. Dan, D. Sitaram and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," *Proc. ACM Multimedia Conf.*, pp. 15-22, 1994.

[4] D. B. Terry and D. C. Swinehart, "Managing stored voice in the etherphone system," *ACM Trans. Computer Systems*, vol. 6, no. 1, pp. 3-27, 1988.

[5] Ray-I Chang, Meng-Chang Chen, Ming-Tat Ko, Jan-Ming Ho, "Schedulable Region for VBR Media Transmission with Optimal Resource Allocation and Utilization," *Information Sciences*, Vol. 141, Issue 1-2, pp. 61-79, 2002.

[6] D. P. Anderson, "Metascheduling for continuous media," *ACM Trans. Computer Systems*, vol. 11, no. 3, pp. 226-252, 1993.

[7] T.S. Chen, W.P. Yang, and R.C.T. Lee, "Amortized analysis of some disk scheduling algorithms: SSTF, SCAN, and *N*-StepSCAN," *BIT*, vol.32, pp.546-558, 1992.

[8] Y. C. Wang, S. L. Tsao, R. I. Chang, M. C. Chen, J. M. Ho and M. T. Ko, "A fast data placement scheme for video server with zoned-disks," *Proc. SPIE Multimedia Storage and Archiving System*, pp. 92-102, 1997.

[9] D.J. Gemmell, H.M. Vin, D.D. Kandlur, P.V. Rangan, L.A. Rowe, "Multimedia storage servers: a tutorial," *IEEE Computers*, pp. 40-49, 1995.

[10] T. H. Lin and W. Tarng, "Scheduling periodic and aperiodic tasks in hard real-time computing systems," *Proc. SIMMetrics Conf.*, pp. 31-38, 1991.

[11] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," *Proc. Real-Time Systems Symp.*, pp. 201-212, 1990.

[12] S. Chen and M. Thapar, "I/O channel and real-time disk scheduling for video servers," *NOSSDAV*, pp.113-122, 1996.

[13] J. Yee and P. Varaiya, "Modeling and performance of real-time disk access policies", *Computer Communications*, vol. 18, no. 10, November 1995.

[14] AHA-2740a Series Technical Specifications, Adaptec, Inc.

[15] P. V. Rangan and H. M. Vin, "Efficient storage techniques for digital continuous multimedia," *IEEE Trans. Knowledge and Data Engineering*, vol. 5, no. 4, pp. 564-573, 1993.

[16] M. Chen, D. D. Kandlur, and P. S. Yu, "Optimization of the grouped sliding scheduling (GSS) with heterogeneous multimedia streams," *Proc. ACM Multimedia Conf.*, pp. 235-242, 1993.

[17] Specifications for ST-32100N, Seagate Technology, Inc.

[18] R. Steinmetz, "Multimedia file systems survey: approaches for continuous media disk scheduling," Computer Communication, vol.18, no.3, pp.133-144, 1995.

[19] R. I. Chang, W. K. Shih and R. C. Chang, "A new real-time disk scheduling algorithm and its application to multimedia systems," *IEEE IDMS*, 1998.

[20] A.L.N. Reddy and J. Wyllie, "Disk scheduling in a multimedia I/O system," *Proc. ACM Multimedia Conf.*, pp. 225-233, 1993.

[21] A.L.N. Reddy and J. Wyllie, "I/O issues in a multimedia system," *IEEE Computers*, pp. 69-74, March 1994.

[22] C.L. Chen, "A Design and Implementation of Continuous Media Storage Server," Mater Thesis, CIS, NCTU, Taiwan, ROC.

[23] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time

environment," *Journal of ACM*, pp. 46-61, 1973.

[24] R.I. Chang, W.K. Shih and R.C. Chang, "Real-time disk scheduling for multimedia applications with a deadline- modification-scan scheme," *Real-Time Systems*, 1999.

[25] The Design of the PDI Subsystem for SVR4ES/MP, UNIX System Lab., Inc.

[26] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *IEEE Computers*, pp. 16-28, March 1994.

[27] R. P. King, "Disk arm movement in anticipation of future requests," *ACM Trans. Computer Systems*, vol. 8, no. 3, pp. 214-229, 1990.

[28] A. Mok, "Fundamental design problems for the hard real-time environment," MIT Ph.D. Dissertation, Cambridge, MA, 1983.

[29] W. K. Shih, J. W. S. Liu, and C. L. Liu, "Modified rate monotone algorithm for scheduling periodic jobs with deferred deadlines," Tech. Report, Univ. of Illinois, Urbana-Champaign, CS, Sept. 1992.

[30] Shin-Hung Chang, Ray-I Chang, Jan-Ming Ho, and Yen-Jen Oyang, "An Optimal Cache Algorithm for Streaming VBR Video over a Heterogeneous Network," Computer Communications, Vol.28, Issue 16, Pages 1852-1861, 2005.