A Fully Automated Fault-tolerant System for Distributed Video Processing and Off-site Replication

George Kola, Tevfik Kosar and Miron Livny
Computer Sciences Department, University of Wisconsin-Madison
1210 West Dayton Street, Madison WI 53706
{kola,kosart,miron}@cs.wisc.edu

ABSTRACT

Different fields including biomedical-engineering, educational research and geology have an increasing need to process large amounts of video and make them electronically available at different locations. So far, this has been a failure-prone tedious operation with an operator needed to babysit the processing and off-site replication of processed video. In this work, we developed a fault-tolerant system that handles large scale processing and replication of digital video in a fully automated manner. The system is highly resilient and handles a variety of hardware, software and network failures making it possible to process videos using commodity clusters or grid resources. Finally, we discuss how the system is being used in educational research to process several hundred terabytes of video.

Categories and Subject Descriptors: D.1.3 [Software]: Programming Techniques - Concurrent Programming

General Terms: Performance, Design, Reliability, Experimentation

Key Words: Video processing, data pipelines, distributed systems, clusters, off-site replication, fault-tolerance, grid, educational research.

1. INTRODUCTION

In different fields including bio-medical engineering, educational research [9] and geology, there is a need to process large amounts of video and make them available at different locations for collaborative analysis. While the computation intensive nature of the tasks and the inherent parallelism make them good candidates for distributed processing, the data intensive nature creates a set of new challenges. These applications do not perform well under existing distributed scheduling systems and may end up not making forward progress depending on the failure characteristics. In this work, we have designed and implemented a fully automated system that processes these videos and handles off-site replication in a fault-tolerant manner.

Our system takes care of transferring the original videos to the compute nodes, processing them and transferring the encoded videos to a set of destinations and handle all the failures that can occur during the process. A unique feature of our system is the separation of the data movement from the computation. We treat the data movement as a full-fledged job and schedule it. This separates the data-movement failures from processing failures and allows us to optimize the data movement by intelligent scheduling.

The system allows all non-interactive video processing to be performed in this distributed manner. The system is flexible enough that the destination can be determined by performing arbitrary computation on the video and permits complex data placement operations.

2. METHODOLOGY

We wanted to build a fault-tolerant system for fully automated distributed video processing. This system must take care of staging the source video to the compute cluster, execute the video processing in the dependency order and finally stage the processed video to a set of destinations.

Certain independent set of processing can be done concurrently. An example is encoding the video to different formats at different resolutions. Each video can also be processed in parallel by splitting it into chunks on frame boundaries, processing the chunks concurrently and finally merging the processed chunks. Certain types of processing consists of refining searches, where a first-pass coarse-grain search is done on all the videos to mark interesting ones and later several of the different searches are done on the interesting videos. This could be more complex with the results of detailed searches used to refine future coarse-grain searches. Further, the set of destinations to transfer the processed video and the original may be determined by performing certain computation. The system should be flexible enough to handle such processing and data movement.

Existing distributed scheduling systems do not work well for such data intensive video processing applications. The amount of data coupled with access over wide-area network make existing distributed filesystems unsuitable. Distributed scheduling systems in the absence of a distributed filesystem allocate the processor, stage-in the input data, perform the computation and stage-out the result of the computation. If the data transfer in the stage-out fails, the computation is re-done because the stage-out is part of the computational job and the whole job is re-tried. This hurts performance in case of intermittent network outages and motivates the separation of computation and data movement.

When several compute nodes simultaneously try to read/write data to/from the storage server, they have the potential to cause denial of service and/or crash the server. Even if that does not happen, the performance will be suboptimal. To handle this problem, data movement should be scheduled appropriately taking into account the network, server and client characteristics.

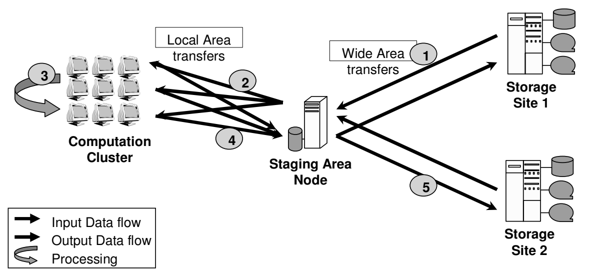

Figure 1: Designs 1 & 2 use Stage Area nodes of the cluster

Due to the above reasons, in our model we separate computation and data placement into separate full-fledged jobs and schedule them appropriately. To handle the dependencies we use a Directed Acyclic Graph (DAG) model where we represent jobs as nodes and dependencies between jobs as directed arcs. For instance, if a job A should be executed before job B, we draw an arc from A to B. A node in the DAG is executed only after all its parent nodes are executed.

In this model provided enough processors are available, computation is performed immediately after data becomes available. In this sense, the model is similar to data flow computing.

We use a computation scheduler to schedule the jobs on the compute cluster. It is not necessary to replace the existing computation scheduler on a cluster as our computational scheduler has the ability to use existing scheduling systems while giving users an uniform interface. This is very useful if we want to use heterogeneous grid resources.

We schedule the data transfer using a specialized data placement scheduler. We have added features to make the scheduler take into account network and host characteristics to optimize the transfer. We also tune the concurrency level to maximize the throughput of the system.

We have come up with 3 designs for our system and present them in the next section.

3. SYSTEM DESIGNS

The most suitable method for staging-in and staging-out the video depends on the cluster configuration and the characteristics of the computation such as the number of independent computation to be performed on the source video. In this section we present three designs and show the cases where each would be the most suitable.

Our system is not limited to a single cluster configuration and can make use of multiple heterogeneous clusters and any of the design can be used in each. In each of the design, we schedule the data movement taking into account network and end-host characteristics. By using network bandwidth and latency measurements, we tune the TCP buffer size to be equal to the bandwidth-delay product. Empirically, we found that as the number of concurrent transfers to/from a storage server increased, the throughput increased to a point and then started decreasing. Further, the number of concurrent transfers needed depended on the data-rate of the individual transfers. Using these empirical values observed during the course of the transfers, we tune the concurrency level to maximize throughput.

3.1 Design 1: Using Cluster Stage in/out Area

We allocate space on the stage area of the compute cluster, schedule the data transfer and after the completion of data transfer, we schedule the computation. This stage-in area is on the same local area network as the compute nodes. This configuration is shown in figure 1

Certain clusters may have a network filesystem and depending on the data, users may want to use the network filesystem. If there is no network filesystem or user prefers to access data from local disk (this case is preferred if the application does multiple passes through the video), then the data is moved from the stage-in area to the compute node using the mechanism provided by the computing system.

There may be multiple independent processing to be performed on the video, so the source video is deleted from the stage area and the space de-allocated only after all processing on this video have completed successfully. This also ensures that the data is staged only once from the remote storage thereby increasing performance by reducing wide-area data traversals.

Moving the data to the local area ensures that it takes deterministic time to move the data to the compute node. This bounds the amount of idle time the processor has to wait before performing the computation. Further, if the compute cluster has a network filesystem, we would be able to use it.

The stage-out process takes place in a similar fashion. If a network filesystem is not being used, space is allocated on the stage-area and the processed video is moved there and then it is scheduled to be transferred to the set of destinations. Once the processed video has been successfully transferred to all the destinations, it is deleted from the stage-area and the space is de-allocated.

3.2 Design 2: Optimizing the Stage in/out Process Using Hierarchical Buffering

Staging-in the data creates an unnecessary data-copy. We try to address this in the second design by using the hierarchical buffering. The hierarchical buffer server executes at the stage area nodes and tries to use memory and then disk to buffer incoming data. It creates logical blocks out of the data stream and performs management at that level. When sufficient blocks have been buffered to sustain transfer at local area network rates to the cluster node, a compute node is acquired and the hierarchical buffer server starts streaming incoming blocks and buffered blocks to the compute node.

If multiple independent computations are to be performed on the source video, the hierarchical buffer server sends a copy of the video to each of the compute node requiring that video thereby performing a multicast.

The hierarchical buffer client running on the compute node takes care of re-assembling the data into the file that the application wants.

This design is suitable for the case where we need to explicitly move the data to the compute node before execution. If the objective is to minimize the compute node idle time, the hierarchical buffer server can be made to call-back when enough data has been accumulated so that from now on it can transfer continuously at local area speeds to the compute node. If there are intermittent failures, the amount of data to be buffered before acquiring the compute node can be increased. Further, this amount can be dynamically tuned depending on the failure characteristics.

3.3 Design 3: Direct Staging to the Compute Node

In the third design as shown in figure 2, we directly stage-in the data to the compute node. This requires that the compute nodes have wide-area access.

While acquiring the processor and starting the stage-in may waste processing cycles, this is more suitable if the scheduling system has compute on demand support. Here, the data transfer can take place

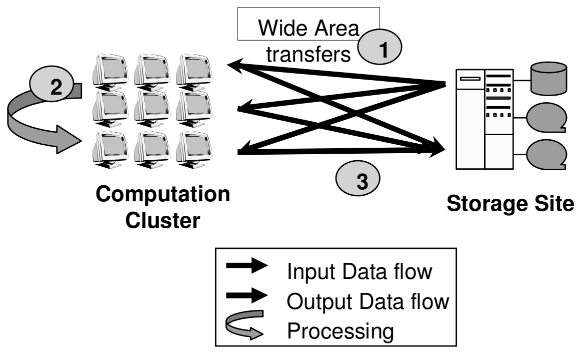

Figure 2: Design3: Direct transfer to compute nodes

at certain rate while the executing computation continues and when the data transfer is completed, the executing computation is suspended and the processor is acquired for the duration of our computation.

To acquire an processor, we need to have higher priority than the currently executing job. We need to do this carefully so that we pick idle nodes first and then randomly choose nodes that are executing lower priority jobs. The objective is to try to reduce starvation of certain job classes.

If there are multiple computation to be performed on the source video, we need to send the video to the other nodes as well. For this, we run the hierarchical buffer server on one computation node and make it write out a copy of the data to disk and stream the video to other compute nodes needing that video. This reduces the wide-area traversal of the source video to the minimum, but introduces more complexity.

3.4 Comparison of the Designs

The different designs are suitable for different conditions.

The first design is simple and universal. The only requirement is the presence of a stage area. Most cluster systems provide that. It is suitable if a network filesystem is being used by the compute nodes. It also works if the compute nodes are in a private network and the stage-in area is the head-node with outside accessibility. It is the most robust and handles intermittent network failures well.

The second design gives a performance advantage over the first one if the data has to be explicitly staged to the compute node. We can use this design only if we have the ability to execute our hierarchical buffer server on the stage area nodes. Minimizing the disk traversal improves performance significantly.

If there a high failure rate or intermittent network disconnections, it may be better to use the first design instead of the second. Note, in this data is being streamed to the compute node after a certain threshold amount of data has been received. This does not do well, if failure occurs after start of streaming data to the compute node. This goes to the problem of finding a suitable threshold. Increasing the threshold improves the failure-case performance but decreases the normal case performance because it increases the amount of data that traverses the disk. At this point we dynamically set the threshold to be equal to the amount of data that has to be buffered to sustain transfer to the compute node at local-area transfer rates. This works well because more data is buffered for slow wide-area connection than for faster wide-area connections. This takes into account the failure characteristics as well because failures and retries reduce the transfer rate requiring more data to be buffered before starting streaming.

The third design gives the best performance if the compute nodes have wide-area connectivity, computational scheduling system has a feature like compute-on-demand and the processor while executing has some extra cycles to spare for data transfer. The last condition happens when the processor has multi-threading support. Here, we need to be careful not to suspend other data-intensive jobs and we need to careful in acquiring processors so as not to starve certain job classes. Further, if other computation nodes need the data, we need to be run hierarchical buffer server to stream data to them. Failure handling is more complex in this case.

4. IMPLEMENTATION

In this section we discuss the tools we used to build our system.

Stork

We carefully schedule the data transfers using Stork [6], a specialized data placement scheduler. Data placement encompasses all data movement related activities such as transfer, staging, replication, data positioning, space allocation and deallocation. Stork provides a uniform job interface for data placement jobs irrespective of the data transfer protocol used. Further, it has the ability to tune data placement jobs at run-time. We use this feature to optimize the system taking into account network, server and client characteristics.

Condor

We use Condor [7] as the computation scheduler. Condor has a functionality called Condor-G which allows us to schedule and run jobs on heterogeneous grid resources. Using this ensures that users have the same computation job interface irrespective of the scheduling system used. Essentially, users submit condor jobs but the cluster system may employ any scheduling system. Further, if the computing system uses Condor, we can use the compute-on-demand feature for design 3.

DAGMan

We have a DAG Model to represent jobs. There may be dependencies between computation jobs and between data placement and computation. We use DAGMan [3] to handle the dependencies. We have enhanced it to submit the data placement jobs to Stork and computation jobs to Condor. This DAG model is flexible and any arbitrary computation can be performed to determine the set of destinations.

DiskRouter

We use the DiskRouter [5] to provide the hierarchical buffer management feature used in design 2. It is capable of performing application level multicast used for streaming the video to multiple computation nodes. We enhanced DiskRouter to support call-backs and to allow dynamic tuning of the buffering before call-back.

Tuning Infrastructure

We have designed a tuning infrastructure to tune the wide-area data transfers. We estimate the bottleneck bandwidth of the wide-area link using *pathrate* and use it to tune TCP buffer size of data transfer protocols.

We also try to determine the characteristics of the storage server to determine the optimal concurrency level. This information can be generated by running our profilers on the storage server and we do it automatically if we are allowed to execute on the storage server. The profilers look at CPU load and I/O characteristics of the server at different concurrency levels and determines reasonably close to optimal values for our configuration. In the absence of the ability to run profilers, we determine the storage server characteristics by observing the throughput of our transfers. Since this profiling occurs by performing actual transfers, it takes the end-host characteristics into account.

We have an initial profile phase where we determine the characteristics of the network and the storage server. This phase involves running *pathrate* followed by actual transfers. After this phase, we periodically estimate the network bandwidth by running *pathrate* and tune buffers appropriately. In case of failures, we use an exponential back off strategy.

Visualization

Our interaction with users showed the need to visually represent what is happening in the system. While this information can be found from log files, users preferred a visualization. We have a primitive visualization setup where we generate visualizations of the data transfers and job completion using DEVise [8] and publish it on a web-page.

Fault-tolerance

We have a hierarchical fault-tolerance mechanism. Both Condor and Stork have persistent job-queues. So, crash and restart of the Stork server or condor server is not a problem. Stork has a sophisticated retry mechanism to add fault-tolerance to data transfers. In case a data-transfer fails, stork tries to see if it can use an alternate protocol to try the transfer. It is possible to make the alternate protocol such as to resume the previous failed transfer. In case of transient failures, Stork uses retries to ensure that data transfers complete.

Finally, another level of fault-tolerance is provided by DAGMan. DAGMan logs the progress of jobs to persistent storage and can continue from last state. In case the machine running Stork server breaks down and is replaced by another, DAGMan can still resume from the previous state. In DAGMan, the number of retries is specified. After that, it writes out a rescue DAG which specifies the jobs which have not executed successfully and this can be passed back to DAGMan to retry again. We also distinguish permanent failures which cannot be fixed by retries, for instance certain processing may fail because the source video is corrupted. In such cases, the error is reported to the user through log files.

5. A CASE STUDY: EDUCATIONAL VIDEO PROCESSING PIPELINE

Wisconsin Center for Educational Research(WCER) has nearly 500 terabytes of video on miniDV tapes and they want to make the original and MPEG1, MPEG2 and MPEG4 formats of the original electronically available at their storage server and SRB mass storage at San Diego supercomputing center(SDSC).

We helped them use our system to encode the videos to different formats and transfer them to SDSC. Since they provide researchers access to these videos via their *transana* [9] software, its meta-data showing the video locations has to be updated after a successful video transfer.

After a straight-forward install of our system on their side, we created job templates for the different operations and a DAG template to specify the dependencies. We wrote a DAG generator which takes as input a list of videos and generates a DAG for the whole operation using the templates.

When they have a set of videos to be encoded and replicated, they generate a DAG for the process using our DAG generator and submit it to our system.

At present, our system uses design 1 and transfers the video to

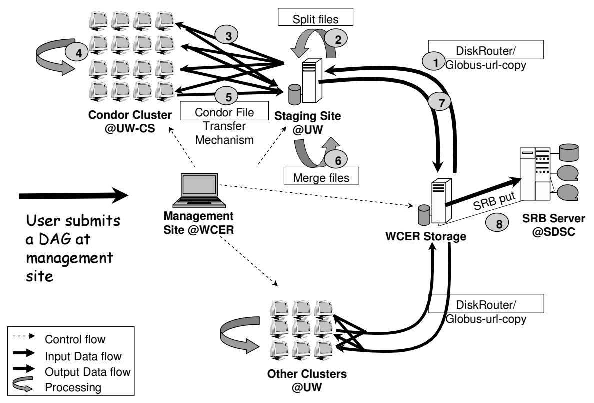

Figure 3: WCER Configuration

the stage area of UW Madison computer science(UW-CS) cluster system using DiskRouter tools and then schedules the computation on that cluster. The whole process is shown in figure 3. We have also shown other UW clusters. We are waiting on getting access to them and would soon start using them.

Because the condor file transfer mechanism used to move data from the stage-area to the individual nodes does not support files larger than 2 GB and most of the videos are around 13 GB, we split the video in the stage area and modify the processing job to merge the input files before processing.

DiskRouter tools was used for direct transfer to the stage-area because of its ability to handle these large files. The system by use of multi-protocol support in Stork supports GridFTP. Since the installed version of GridFTP does not support these large files, we are presently unable to use it. The latest version of GridFTP supports large files and we would be trying it soon.

We also tried using Design 2 and Design 3. Design 2 improved performance by about 20%, but design 3 slightly worsened the performance because we need to stream the data to two other nodes and in this cluster configuration, the stage-area node has gigabit connectivity while the compute nodes have only 100 Mbit connectivity.

During the course of processing, we had several failures. There were several intermittent network failures, most lasting around 5 minutes and some longer. In addition, there were planned maintenance when patches were applied to the routers. Also, SRB system at SDSC has a weekly planned maintenance window when it is not accessible. Our system was robust to all these failures and continued the processing without any human intervention.

6. RESULTS

Figure 4 shows the DEVise visualization of the data flow taking place. The y-axis is in MBPS. We are performing tuning and get close to the maximum data transfer rate on the links. The link from UW-CS to WCER is limited by the 100 Mbit interface on the WCER machine while other traffic limits the data transfer rate from WCER to UW-CS.

The break in the middle is an artificial network outage we created. As it can be seen, the system recovers from the outage automatically without any human intervention.

We also determined that we need two concurrent transfers for GridFTP and one transfer for DiskRouter to maximize the throughput of the storage system.

7. RELATED WORK

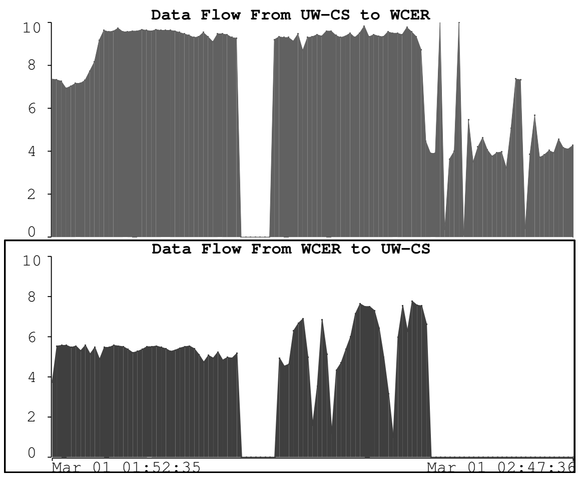

Figure 4: DEVise visualization

Considerable work has been done on parallelizing MPEG encoding [1] [4]. Our work would be able use these existing work and add full automation and fault tolerance to the process. We take into account the fact that the source and destinations may not be in the same local area network as the compute nodes and handle wide-area data movement.

Another differentiating factor of our work is that normal(non-parallelized) video processing tools can be employed. Here, we get speedup from concurrently processing multiple videos and concurrently performing various processing. We believe that this is very useful in a real-world setting where parallelizing certain processing requires significant work. Further, a user may be concerned about the turn-around time of an entire set of processing and may not care about the individual processing time.

BAD-FS [2] builds a batch aware distributed filesystem for data intensive workloads. This is general purpose and serves workloads more data intensive than conventional ones. But, we are not sure if it would be able to handle video processing workloads which are highly data-intensive and for performance reasons prefer to access source data from local disk rather than over a network filesystem. Further, BAD-FS at present does not schedule wide-area data movement which we feel is necessary given the size of the videos. Our system is specialized towards video processing and would perform better for this class of applications compared to the more general purpose BAD-FS.

8. FUTURE WORK

We would like to improve the visualization showing the entire processing. We are trying to design an automated way of determining the most suitable of the 3 designs and using it.

9. CONCLUSIONS

We have successfully designed and built a fault-tolerant system capable of fully automated distributed video processing. The system is resilient to a wide-variety of network, hardware and software failures and recovers automatically without any human intervention. It schedules wide-area data transfer taking into account network and end-host characteristics and maximizes the throughput of the system.

10. ACKNOWLEDGMENTS

We would like to thank Chris Thorn, Chris Fassnacht and other people at WCER for letting us to use their resources, for using our system and giving us feedback. We would also like to thank Steven Schultz for helping us resolve problem with video encoding tools.

11. REFERENCES

[1] S. Akramullah, I.Ahmad, and M. Liou. Parallelization of mpeg-2 video encoder for parallel and distributed computing systems. In *Proceedings of 38th Midwest Symposium on Circuits adn Systems*, August 1996.

[2] J. Bent, D. Thain, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Explicit control in a batch-aware distributed file system. In *Proceedings of the First USENIX/ACM Conference on Networked Systems Design and Implementation*, March 2004.

[3] Condor. The Directed Acyclic Graph Manager. http://www.cs.wisc.edu/condor/dagman, 2003.

[4] Y. He, I. Ahmad, and M.L.Liou. Real-time distributed and parallel processing for mpeg-4. In *Proceedings of 1998 IEEE International Symposium on Circuits and Systems (ISCAS '98)*, June 1998.

[5] G. Kola and M. Livny. Diskrouter: A Flexible Infrastructure for High Performance Large Scale Data Transfers. Technical Report CS-TR-2003-1484, University of Wisconsin, 2003.

[6] T. Kosar and M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid. In *Proceedings of the 24th Int. Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.

[7] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[8] M. Livny, R. Ramakrishnanand, K. Beyerand, G. Chenand, D. Donjerkovicand, S. Lawandeand, J. Myllymaki, and K. Wenger. Devise: Integrated querying and visual exploration of large datasets. In *Proceedings of ACM SIGMOD*, May 1997.

[9] C. Thorn. Creating new histories of learning for Math and science instruction: Using NVivo and Transana to manage and study large multimedia datasets. In *Conference on Strategies in Qualitative Research: Methodological issues and practices in using QSR NVivo and NUD*, London, February 2004.