A MJPEG Encoder for the NOW to visualize and navigate on data-intensive scenarios

José María González
chema@cs.berkeley.edu

May 17, 1999

Abstract

This paper describes the design and development of a virtual-reality system that permits a remote user to navigate on very large image assets. The main idea is to allow the user to access interactively to a huge amount of data (the images where he is navigating) adapting the resulting stream bandwidth (a MJPEG video one) to the capacity of his link. The main piece of the system is a parallel MJPEG encoder that reads huge amounts of data and distillers from there the resulting video stream. We use a data-flow programming environment and I/O substrate for networks of computers (River) as the basis of our implementation: River provides a framework to glue parallel applications so that the adaptable distillation of the video stream can be done easily.

1 Introduction

One of the main problems that presents the parallel handling of huge multimedia (video) sources in a network of workstations (NOW) is the inability of a user to extract the information dealt with and monitor the task being carried out, so that he can add new stimuli and therefore interact with it. While the cluster can be connected by huge inner communication links, the bandwidth of the networks that connects it to a remote controlling computer is much smaller and also more variable. Unless the user computer is a part of the cluster of workstations, it is impossible to access to the huge bandwidth being processed in an amount of time small enough to permit an interactive management of the system. [1]

The solution to this problem is to degrade the images to a bandwidth small enough to permit their transmission through the user's box link. Unfortunately, both the reading of the images from disk and their efficient encoding are very time-consuming tasks, so the only possible implementation is a parallel one. This brings some new issues up, like the need to control the processing power used and therefore the maximization of the scalability of the solution provided.

[1] Even if the remote controller display is part of the cluster, there are a different set of problems associated with the local management of huge amounts of multimedia data that won't be issued in this paper: both the presentation display and the internal remote controller bus have a maximum bandwidth.

This paper provides a solution to this problem in the form of a MJPEG parallel encoder. The encoder implementation is based on the use of a data-flow programming environment and I/O substrate for networks of computers, River ([AAT+99]). River provides a framework to glue parallel applications so that the adaptable distillation that the encoder carries out in the video stream can be done easily. It also permits the fast parallel disk access to the images. The result is an efficient, scalable, parallel encoder that supports more than 22 MB/s of data reading and encoding on close to 50 machines. The implementation demonstrates the usefulness of the framework and should be also regarded as a contribution to the growing number of data-intensive applications created on River.

We have also designed and developed a virtual-reality system that permits a remote user to navigate on very large, real-image assets. The main idea is to allow the user to access interactively to the images where he is navigating adapting the resulting stream bandwidth (a MJPEG video one) to the smaller capacity of his link. The main piece of the system is the parallel MJPEG encoder that distills from the images the resulting video stream.

The rest of the paper is structured as follows: section 2 explains the parallelization process and the architecture of the system. Section 3 gives some details on the implementation of the encoder, and section 4 discusses its performance results, focusing especially on the system scalability. We wrap up with some future work (section 5) and then conclude.

2 Architecture

This section presents the parallelization process and the resulting encoder architecture. As the main objective of our encoder is to be interactive, we want as many of the processors involved in the parallelization process as possible working in every frame. To be able to divide the task of encoding every frame between all the processors, we split every frame in horizontal strips, and subdivide every strip in vertical-chopped blocks. Every block is encoded separately.

The rationale under this parallelization is fourfold:

1. interactivity: we want to minimize the answering delay of the encoder. The alternative for non-interactive encoders will probably be maximizing the encoding bandwidth.

2. avoiding inter-process communication: we will explain this in depth below.

3. flexibility: we can easily play with the number of stripes and columns to permit adding more processors to the system.

4. simplicity: this solution can be implemented pretty straightforward.

For simplicity, every node can encode any of the blocks of any of the frames. This is possible because the video codification system we are using

(MJPEG) permits to avoid using spatial information from outside a block to encode it. Moreover, MJPEG doesn't present temporal dependences between blocks, making it easier the parallelization process. These are the main reasons for choosing MJPEG as the video coding system. Moreover, MJPEG provides high bandwidth reduction with high quality (1:20 at least with a good quality parameter). [2] Of course this degradation can be insufficient for the video stream to fit the extern cluster to remote control computer link. In this case, the lossy MJPEG compression should be accompanied by frame decimation.

An obvious lack this parallelization is that we don't permit dependences between blocks encoded in different encoders. This can cause a lower quality in the resulting stream.

The encoder architecture, with the navigator add-ons, is depicted in figure 1.

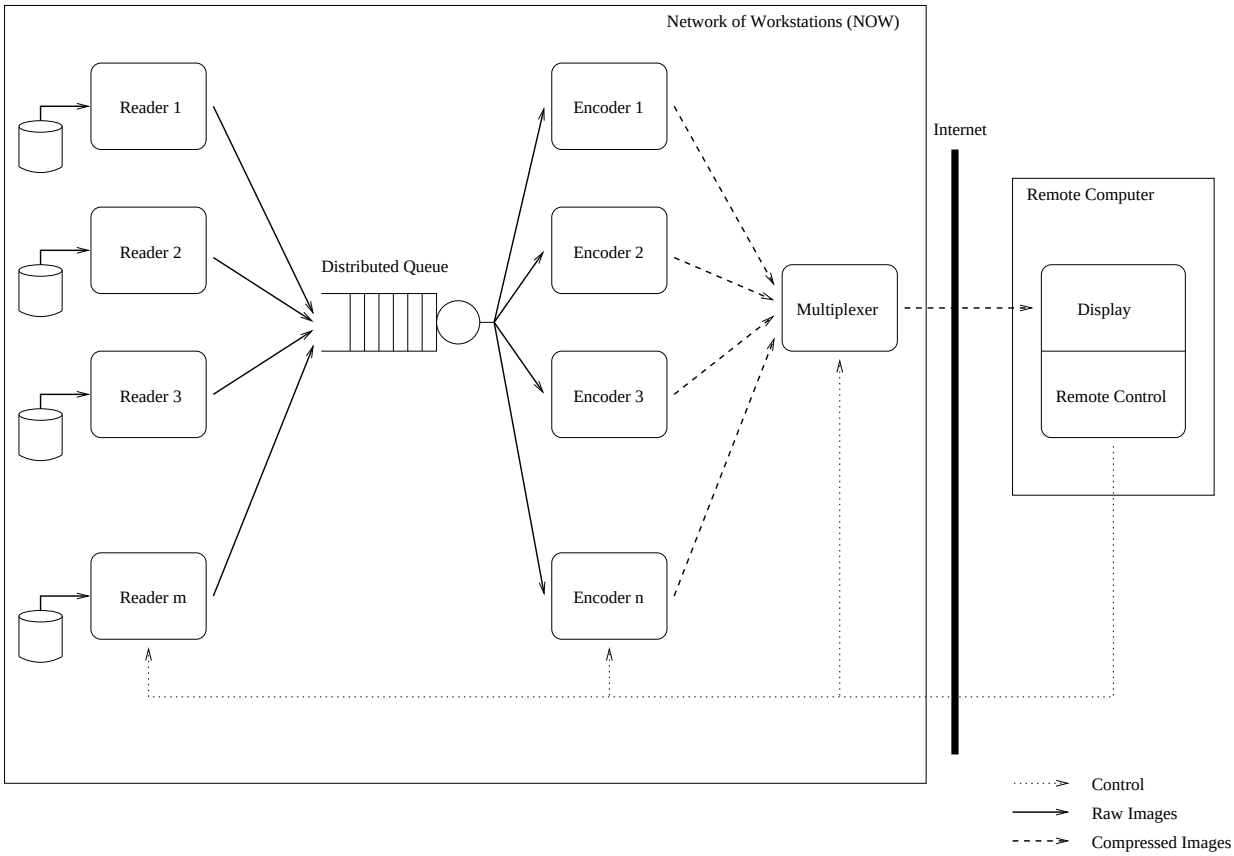

Figure 1: Encoder System Architecture

2.1 Readers

The readers wait for a remote control command asking for a new frame. Once received, they use the 3D coordinates attached to the command and

[2] Our MJPEG encoder is based on an adaptation of the freely available Independent JPEG Group (IJG) JPEG encoder.

their own name to calculate the image blocks they must provide. To ease this, we map a strip to every name, so that every reader is in charge of fetching a different strip. They complete it by fetching the strip blocks (from left to right) and sending them to the distributed queue one by one. Figure 2 depicts the task of a reader in a small example.

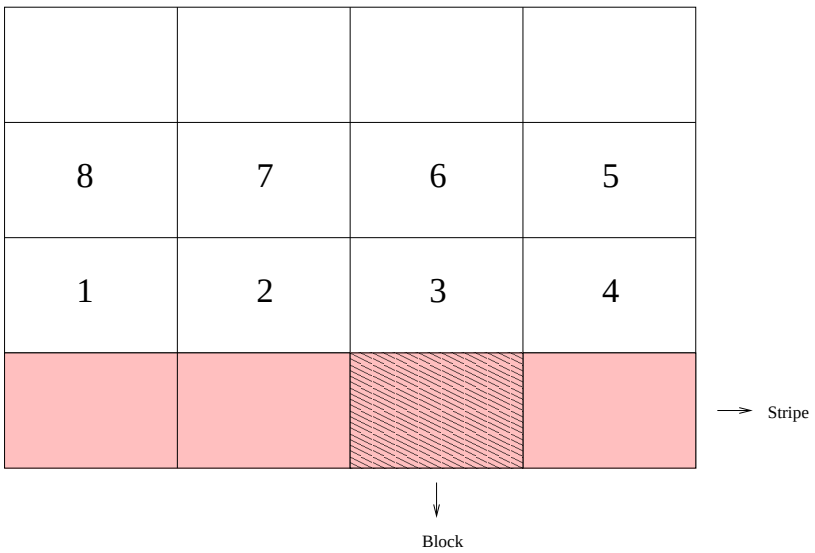

Figure 2: Example of a reader block fetch order

When a reader finishes its strip, it starts reading and sending the upper strip blocks in the opposite order (from right to left) so that slower encoders are helped by fast ones. When they receive a new request from the remote control, they know that all the blocks were received at the multiplexer, and therefore they stop sending the old strip, read the new coordinates and start the process again.

One interesting aspect of this approach is that it doesn't require any communication between the modules except the data itself. We wanted to avoid a centralized coordination point to make the system as scalable as possible and to avoid overwhelming the multiplexer, which could easily end up being the system bottleneck. A simpler alternative implementation would be for the remote control to send its request to the multiplexer so that this one can act as a global coordination by accounting which blocks composing the frame have been read and/or encoded and which not, and coordinating the readers so that they will fetch and send them to the distributed queue.

2.2 Encoders

The encoders wait in the distributed queue output. When a new block is available, the first encoder who is idle takes it and compresses it. When it finishes, it sends the results of the encoding to the multiplexer.

2.3 Multiplexer

The multiplexer receives all the compressed blocks from the encoders and stores them. When all the blocks of an image are present, it glues them to form a full image that is send to the remote display.

2.4 Remote Computer

The remote computer is composed of two processes: one that reads the images from the output of the encoder and displays them (*display*) and another one that controls the full framework, interfacing with the user and requesting the needed frames to the encoder *remote control*. The remote control asks for a new position when the user wants to move his position. The number of frames the remote control will ask is limited to 30, which is the maximum number of frames the display will present.

3 Implementation Details

The main implementation basis of the parallel encoder is River ([AAT+99]), a data-flow programming environment and I/O substrate for cluster of computers. River tries to provide maximum performance through two simple design features, a distributed queue, and a storage redundancy mechanism called graduate declustering. The idea of the first one is to balance work across consumers by permitting fast ones do the work of slower ones. The second one tries to balance production across producers in the same manner.

The River implementation consists of three River modules, one for each of the encoding process types (reader, encoder, and multiplexer), plus some scripts to fire them. The system comprises also two tcl applications in the remote computer (one for the displayer and the other for the remote control terminal) Figure 3 depicts the River implementation of the cluster side.

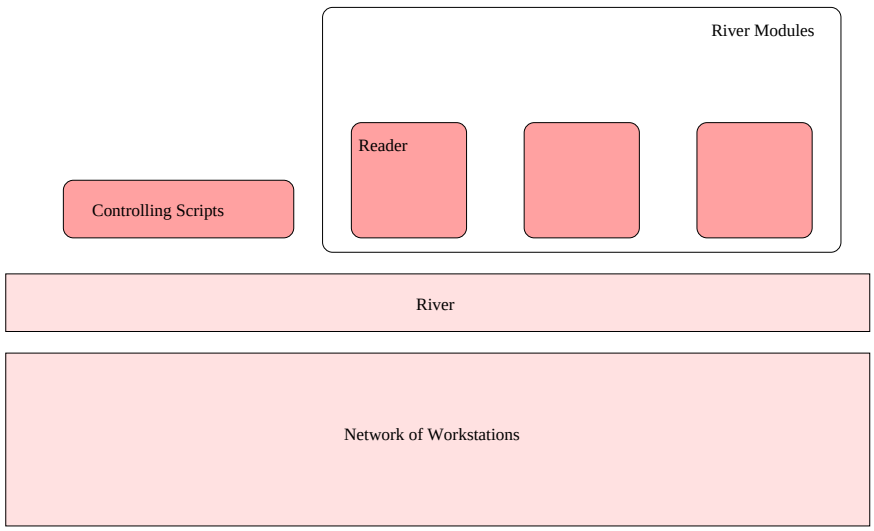

Figure 3: The River implementation of the system

4 Performance Results

The first two experiments we did tried to know the readers and encoders performance isolated. In the first case, we got disk performance values ranging from 2 to 3 MB/s (with some peaks in low loaded nodes close to the nominal 5 MB/s). We checked that when a file was present in the node cache the performance grows to 120 MB/s. On the other hand, the performance of the IJG JPEG encoder is more constant: close to 2.6 MB/s [3]

The main experiment tries to know how many data is our system able to process as a function of the configuration used (i.e., the number of nodes in the cluster that are working for our system). We have tried different configurations for the number of readers and encoders (the number of multiplexers is always one because we check that it is idle almost all the time). However, as the reader and encoder throughput is similar, we opted for always using the same number of modules of each type.

Due to project temporal constrains, this experiment doesn't use the distributed queue, but maps the readers to the encoders one to one with AM-Reservoirs. This affects to the performance because now a mismatch between the encoder and its mapped reader speed will cause the performance of the pair to be limited to that of the slower one.

The experiment consists on, for every configuration, try to encode a fixed number of images as fast as possible. The remote control sends a request for an image and, as soon as it receives it, it asks for another one. We did the experiment with medium (227×149, full-color (3 bytes/pixel), 100 KB bitmaps) and huge images (3632×2235, full-color, 23 MB ones) as source. In both cases we got similar throughput numbers, so we used them instead of the number of frames. Figure 4 presents the experiment results.

We have marked the points where we could be able to manage some popular video codification system (supposing their encoding processor needs are similar to the MJPEG ones). They are explained in table 1.

Name	Explanation	Bandwidth
ATV (Advanced Television)	1920×1080 pixels, 12 bits color, decimated to 10 frames per second	30 MB/s
CIF (Common Image Format)	30 fps	4.6 MB/s
QCIF (Quarter Common Image Format)	30 fps	1.14 MB/s
H.261	parameter p from 1-30	$p\times$ 8KB/s

Table 1: Video Coding Systems marked in figure 4

From the experiment results, we can see that we are able to produce (extract and encode) more than 1 MB/s of unordered raw data (including

[3] We did these experiments in the Berkeley Network of Workstations (NOW), composed of 100 SUN Ultrasparcs and 40 SUN Sparcstations running Solaris, 35 Intel PCs running Windows NT or a PC UNIX variant, and between 500-1000 disks, all connected by a Myrinet.

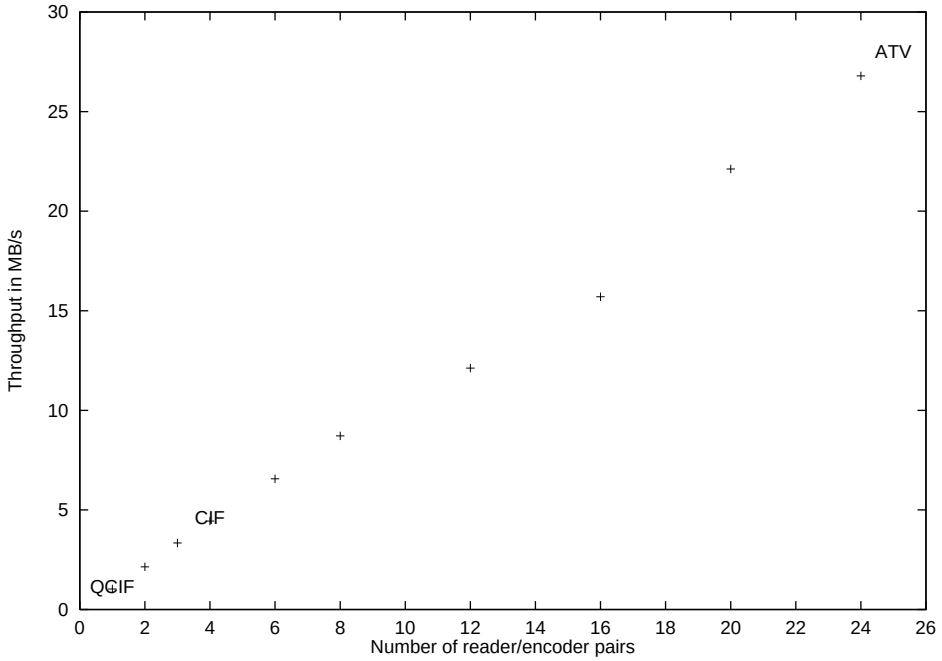

Figure 4: Encoding system performance results

disk seeks to get the different blocks) per reader/encoder pair in a medium-loaded cluster. We can also see that the system scalability is quasi-perfect. Even we haven't been already able to do it, we are looking forward testing the system with 50 encoder/reader pairs. From the study of the occupancy values of all the modules, we feel confident that the system will scale because the only single point (the multiplexer) is idle 99% of the time.

An interesting experiment will be to play with the vertical chopping parameter for different configurations. Creating more blocks gives more opportunities to overlap the reading and encoding process, but on the other side it introduces extra overhead in the multiplexer (now it has to glue more blocks) and diminishes the reading performance (readers access to the same amount of data but in a lot of smaller streams, therefore increasing the impact of seek and rotational time), while diminishing the quality obtained at the encoders (they have less space to look for matching blocks).

An important detail of the performance experiments is that as we wanted them to be as fair as possible. This means that we had to ensure that everyone found the same file cache state and the same workload. In the first case, we wanted none of the files to be fetched was present in the cache: As we have said, the fact of finding the file we want to fetch in the cache improves the read performance between 40 and 60 times, so the experiments would be flawed. We have implemented a file system cache cleaner module on River. This module, called between every experiment, will read enough files to flush its cache. [4]

[4]Of course these cache effects will have a good effect in the navigator performance (when the user returns to a virtual location he has been before).

In the second, we did all the experiments in a small interval of time, where the node workload was approximately constant (a medium value). Some smaller experiments done in low and high workload instants confirmed us the insight on the importance of taking care of the system load: we got from close to doubled performance to a forth depending on the instant the experiments were done.

5 Future Work

As we have seen, with very huge read and encoding bandwidths, the system scales perfectly, so the architecture limiting factor is only the link bandwidth. This means that we can adapt n MB/s (1 MB/s per reader/encoder pair) of a raw video stream to any output link bandwidth just by encoding on-the-fly and decimating the resulting stream. An interesting ideas would be to make this output bandwidth of the navigator adaptable to the instantaneous link one (currently is just fixed to the link).

A possible enhancement would be to implement the system at a lower level, possibly using the MPI interface. This should permit carrying out fine tuning while not undermining the portability of the system.

Another problem that must be addressed is the display module. The tools we are using up to the moment are not powerful enough to support the rendering of images at the stream speed (close to 20 frames per second). An interesting alternative is to add some more processing at the multiplexer so that instead of just creating MJPEG frames it also adds an RTP header, therefore permitting the use as a display of common multimedia tools like `mash`. It could be also interesting to change the simple MJPEG codification to a more popular and efficient encoding, probably MPEG. This would require probably some communication between the modules plus adding a framework to support temporal dependences between frames compressed in the same or different encoders (keep some old blocks information in the first case and adding inter-encoders communication in the second). Moreover, it would oblige us to rethink the use of the same number of readers and encoders. With more complicated encoding systems, more encoders should probably be added to maximize the system utilization.

Another enhancement would be to add some kind of prefetching policy to the navigator. Actually it is limited to just fetching and encoding the frame it is requested, but it should be pretty straightforward to try to guess the user trajectory by prefetching frames. Moreover, our display process could cache some of the images it has received in order to limit the requirements to the encoder. We can guess that the probability that the user returns to the same places he has been is relatively important, and therefore this will increase significantly the system performance.

6 Conclusions

This paper describes the design and development of a virtual-reality system based on a parallel MJPEG encoder. We have shown that the encoder is

completely scalable, providing 1 MB/s of unordered raw data access and encoding per pair of nodes (reader and encoder). This system can be used as a tool for multimedia projects in parallel architecture, permitting a remote user to visualize the data-intensive processes he is dealing with in an interactive manner.

References

[AAT+99] R.H. Arpaci-Dusseau *et al.* "Cluster I/O with River: Making the Fast Case Common". IOPADS 99.

[CP98] D.E. Culler, J.P. Singh "Parallel Computer Architecture: a Hardware/Software Approach". Morgan Kaufmann Publishers ISBN 1-55860-343-3

[SH98] M. Stonebraker and J.M. Hellerstein "Readings in Database Systems", 3rd edition. The Morgan Kaufmann Series in Data Management Systems ISBN 1-55860-523-1

[PM92] W.B. Pennebaker and J.L. Mitchell "JPEG still image data compression standard". Van Nostrand Reinhold, 1992 ISBN 0-442-01272-1

[Ham92] E. Hamilton. "JPEG File Interchange Format", version 1.02. C-Cube Microsystems, Sep. 1992

[KR88] B.W. Kernighan and D.M Ritchie "The C programming language", 2nd edition. Prentice Hall Software Series, 1988 ISBN 0-13-110362-8

[Ous94] John K. Ousterhout. "Tcl and Tk Toolkit". Addison-Wesley, 1994 ISBN 0-201-63337-X

[Wel97] Brent B. Welch. "Practical Programming in Tcl & Tk", 2nd edition. Prentice Hall Press, 1997 ISBN 0-13-616830-2