Real-time Capable File System for Scalable Media

Algorithms & Caching Strategy for Exploiting Scalability on File System Level

Heiko Sparenberg, Matthias Martin, Siegfried Foessel
Moving Picture Technologies
Fraunhofer IIS
Erlangen, Germany
{heiko.sparenberg, martinms, siegfried.foessel}@iis.fraunhofer.de

Abstract—**Professional movie distribution and mastering formats use high quality, intra-frame compression formats like JPEG 2000, a codec allowing for scalability in resolution and quality. Latest processor types, either CPUs or high-end GPUs, can decode digital cinema-compliant JPEG 2000 image sequences in real-time. This brings out a new bottleneck during a movie playback since current consumer hard-drives are not fast enough for data rates being used in digital cinema or digital archiving. This paper shows a method for using file-inherent scalability on file system level as well as a caching strategy especially adapted for scalable media files.**

Keywords-component; real-time; virtual file system; caching for scalable media; JPEG 2000; image/video data compression

I. INTRODUCTION

Digital movie distribution and archiving requires image compression in order to reduce the memory requirements. The Digital Cinema Initiative [1] defined the use of JPEG 2000 [2] for the distribution of digital movies. Image sequences can be compressed with a data rate up to 250 MBit/s. The same trend can be observed in the field of digital archiving. Again, JPEG 2000 seems to be the most promising codec since it offers both, lossy and lossless compression. Data rates here are generally higher compared to digital cinema. The International Organization for Standardization (ISO) adopted new JPEG 2000 profiles [3] with a data rate up to 500 MBit/s or even lossless.

For years it was impossible to decode such a JPEG 2000-compressed image sequence on a standard PC without special hardware accelerator cards, since available computing resources were not sufficient. But latest processor types as well as algorithms [4] for the acceleration using consumer graphic cards [5] allow for real-time playback of those movies. Unfortunately, these results brought out a new bottleneck in the processing chain: current consumer hard-drives are not able to deliver the required performance in order to read a movie in real-time (Tab. 1). Especially external hard-drives – connected via USB – lack the required resources. This may cause playback software to skip frames, even if the decoding performance of the destination machine is fast enough for real-time applications. Fortunately, JPEG 2000 – besides other media-formats like H.264 SVC [6] or MPEG4-SLC [7] – is designed in a way allowing for scalability in various dimensions, e.g. quality or resolution.

TABLE I. DISC SPEED MEASUREMENT (READING) USING FOUR COMMON STORAGE DEVICES, FILE SIZE: 2MB, BLOCK SIZE: 8 KB, ACCESS: WIN 32 API – UNCACHED, CPU: 3 GHZ, SOFTWARE: PERFORMANCETEST 7.0

	Type	*Min [MB/s]*	*Max [MB/s]*
1	Conventional Hard Disk[a]	48	66
2	Low cost USB Stick	6	16
3	USB Drive[b]	8	19
4	Solid State Drive	117	125

a. Seagate Barracuda 7200.10 - ST380815AS [8],
b. Seagate Momentus 5400.6 connected via USB 2.0 - ST9320325AS

In order to compensate a lack of (decoding-) resources, a reduction of e.g. resolution is already extensively applied on application-level. Unfortunately, file systems do not offer functionality to take advantage of the file-inherent scalability in order to increase their performance. Hence, playback software will not receive the required data within the appropriate time. Images have to be skipped and a smooth playback of the movie is not possible.

These aspects led to the idea that scalability can be exploited on file system level, thereby eliminating another bottleneck in the decoding chain. This paper presents a collection of ideas and methods that show how scalability can be used on file system level in order to improve the performance of a storage device. Developed and optimized for JPEG 2000, the algorithms can be applied to other scalable media formats so that real-time applications are possible, even if the storage device lacks the necessary performance.

II. SUBSTITUTION STRATEGY

JPEG 2000 codestreams consist of a header portion and subsequent data packets (Fig. 1a, Orig.). Each packet contains information for only one of the scalable dimensions: quality, resolution, component, or precincts. In order to speed-up decoding, packets can be discarded by the decoding application. Especially resolution and quality packets are perfect candidates for this method. An impact on both, a reduction of image quality and a faster decoding speed, can be observed. Unfortunately, a simple discard of packets at the file system level is not a satisfactory solution, since the file size as well as the internal structure of a file would differ from the original file; decoding applications will not be able to react on

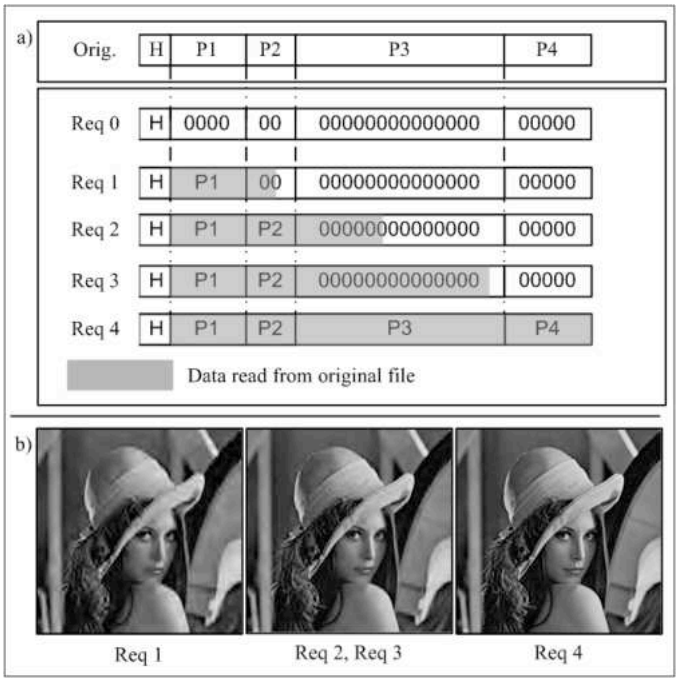

Figure 1. a) Caching method including four requests. Each request will be stored in memory, complete packages will be returned to calling application. b) Results of substitution strategy in combination with caching strategy.

such spontaneous changes. Therefore, it is important that a strategy depicts the metadata and structure of the original file. Furthermore, the delivered data packets must also present a compliant codestream in order to make sure that the decoding application receives a valid image.

Since we assume that the storage device is too slow to deliver the complete file, we expect it to be able to deliver a considerably, smaller portion or at least the header information in time. The header reveals how many packets are available in the codestream as well as the size of these packets. This knowledge can be used to create a virtual file with the same structure acting as proxy for the original image. Initially, all reconstructed packets are undefined since no real data has been read from the storage device. Therefore, all data will be substituted with zero-packets (Fig. 1a, Req 0); decoders would display a grey image with the resolution of the original file.

Depending on the current storage device a certain amount of data can be read in the given time. All packets being read completely will be returned to the calling application; all non-completed packets will be substituted by zero-packets (Fig. 1a, Req 1-4). The decoding application accepts the zero-packets since it has no information whether the data really belongs to the image or if it was generated by the substitution strategy. It is important, that a decoding of these packets will not add any additional information to already processed data from the previous packets [9]. The substitution method thus enables for a creation of images with identical meta-information of the original data; a decoding application must therefore not react on sudden changes such as resolution. By replacing the missing data with zero-packets the image content is changed in such a way that the quality decreases. Fig. 1b shows three different images: (1) the left image shows a reduced quality since most of the data from the original image has been replaced with zero-packets (Fig. 1a, Req 1), (2) the center image shows more details and fewer block-artefacts compared to the left image since less data has been replaced with zero-packets (Fig. 1a, Req 2-3), (3) the right image shows the full quality since no data from the original file has been replaced with zero-packets (Fig. 1a, Req 4).

Depending on the original files, compressed images may have a varying file size. That's why the substitution strategy will not take effect if the file size is small and the storage device is able to deliver the complete file in the given time. But other images in a sequence might be too large to be read fast enough.

III. Caching Strategy

A cashing strategy on top of the substitution method allows for an incremental increase in quality for images that have been accessed more than once, e.g. during the post-production where a predetermined scene gets played in a loop in order to add effects or to adjust the color correction. The file system itself will determine if it can serve a request with data in the cache, or if it has to read from the storage device. Furthermore, a combination of both – data from the cache as well as data from the storage device being read during the current request – will decrease the number of substituted data packets and thus increase the image quality. Fig. 1 shows how the caching strategy works: As mentioned in the substitution strategy, a reconstructed file will be created in memory by using the metadata of the original file (Fig. 1a, Req 0) in case it is not already present in the cache. Still assuming that the storage device is too slow to deliver the whole image in time, the file system only reads a smaller portion of the original file. All portions being read from the storage device will be stored in the cache. Packets that could not be read in its entirety will be replaced by zero-packets. For example, the first request (Fig. 1a, Req 1) offers enough time to read the first packet of the original image completely, therefore the file system can send a valid codestream to the calling application. All data read from the storage-device will be stored in the cache without paying attention to packet boundaries. Since information of the original file was replaced largely, the image lacks quality (Fig. 1b, left image). If the same image is requested again, the file system continues to read the original file at the position where the last call stopped. Therefore, the next two calls (Fig. 1a, Req 2-3) enable the file system to read more packets of the image file. Since packets can only be returned if they have been read completely from the storage device, these requests lead to the same result (Fig. 1b, center image). Finally, the next request (Fig. 1a, Req 4) allows the file system for returning the complete image.

IV. Proposed Architecture

The architecture proposed in [10] offers a suitable platform for the integration of the methods described in this paper. The

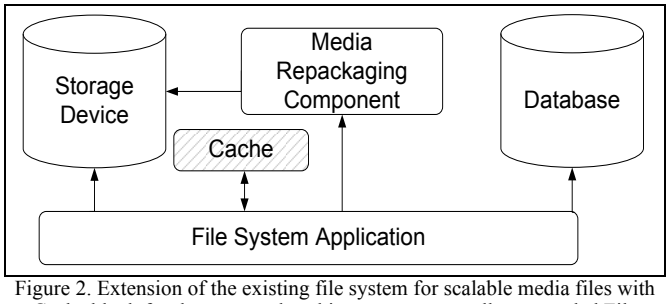

Figure 2. Extension of the existing file system for scalable media files with Cache block for the proposed caching strategy as well as extended File System Application including the substitution method.

approach uses a virtual file system to add user rights to specific parts of the images. A Media Repackaging Component takes advantage of the user rights and generates new versions of the existing images by re-ordering the packets within a scalable codestream. This mechanism allows for different versions in terms of resolution and quality derived from the same data without time-consuming re-coding. Fig. 2 shows an extension of the existing architecture including a new Cache block for the strategy described above. The File System Application has to be modified in order to implement the substitution strategy. The internal structure of the architecture is transparent to a calling application and requests to the File System Application will be processed as expected from a conventional file system. The virtual file system can be mounted on to an operating system's file system tree.

V. RESULTS

In order to determine the improvement of the substitution method over the conventional access to a storage device, two different types of measurement series were performed: (1) disc speed measurements using a conventional consumer hard disk and (2) disc speed measurements varying in block-size, being forwarded to the conventional hard disk by the virtual file system. Example: if a decoding application requests a picture from the virtual file system, a request to the conventional hard disk with block-sizes of 250 kB, 500 kB, etc. gets forwarded within the request, in order to read a part of the original picture. In accordance to the substitution method described above, incomplete or missing portions will be replaced by zero-packets.

JPEG 2000 images with an average file size of 3.5 MB and resolution of 2k were used. By encoding the images with four resolution- and five quality-stages the scalable codestream includes exactly 75 packets. Under the assumption that a file system must deliver 24 frames per second (fps) to support real-time playback, a response time lower than or equal to 42 ms for each frame – as well as a data throughput of 84 MB/s – has to be reached.

The response time of the conventional hard disk is listed by the most right block shown in Fig. 3. The direct access to the drive gained a data throughput of max. 66 MB/s which results in a response time of 52 ms. Considering the real-time aspect, play out with 24 fps is not possible with the used configuration. Furthermore, the measurements show that when using the substitution method, the real-time-limit can be undercut (Fig. 3 - 1st and 3rd block). With a response time of 36 ms and 41ms,

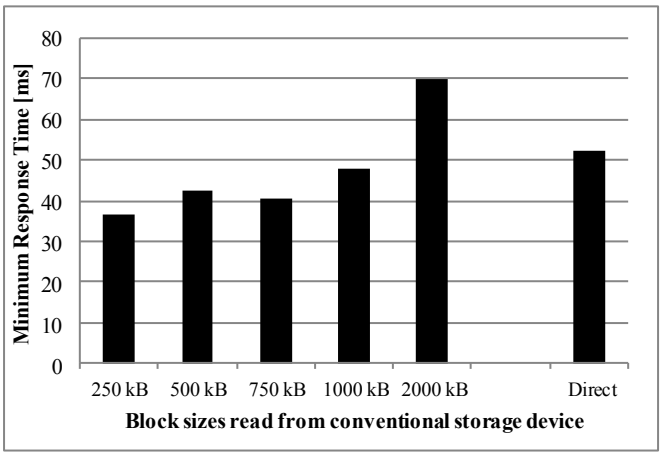

Figure 3. Minimum response time for an image sequence depending on the data amount read from the hard disk during each request using the substitution strategy (left five columns) in comparison with direct response time of the conventional file system (right column).

the block sizes of 250 kB (respectively 750 kB), are small enough for supplying real-time applications. Starting from a block size of 1000 kB even the substitution method can not deliver real-time performance. Essentially, the reason to look for is that the virtual file system runs in the operating system's user mode. Requests to the system are therefore accompanied with a number of changeovers between user and kernel mode of the operating system. This reduces the performance of the overall system. Therefore, it is expected that a port of the proceedings into a file system running in the operating system's kernel mode, will achieve significantly better performance.

The described effect of the substitution method increases if the file sizes of the original images get bigger: a conventional hard disk could just continue to supply the fixed data throughput and the number of frames delivered per second would deteriorate with larger source files. The performance of the substitution method would not deteriorate, since it would still read a defined block size from the original files. Since the missing data (zero-packets) must only be copied from memory, the performance remains constant. By reducing the block size (<250 kB), the response time of a storage can be decreased further. This also enables the usage of slower devices, e.g. USB sticks or optical discs, to be used in real-time applications.

As a supplement to the substitution method the caching strategy for scalable media files was also investigated: different test series were recorded with the same image-set described above. Fig. 4 shows the average number of accesses to an original file necessary until it completely resides in the cache of the virtual file system. From that point on further inquiries can be served entirely from memory without requests to the original file. It can be observed, that, when using a block size of 750 kB, more than 50% of the 75 packages within the scalable codestream are already in the cache after the first access to the file. Such a substituted variant can be more than sufficient for previewing purposes; especially when small displays are used. Fig. 5 shows that the data rate between the

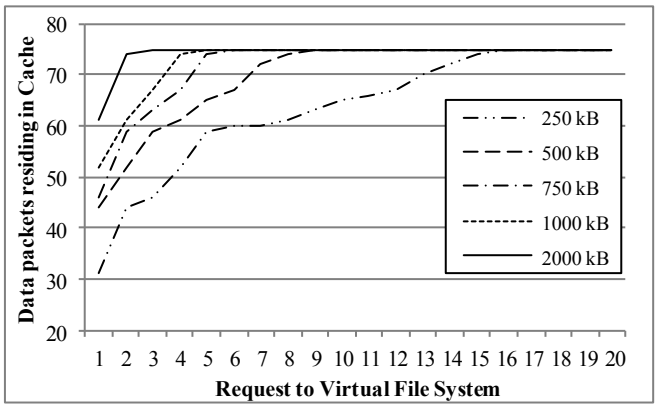

Figure 4. Average number of queries needed to copy complete images to the cache as a function of block size being read from the original file per request.

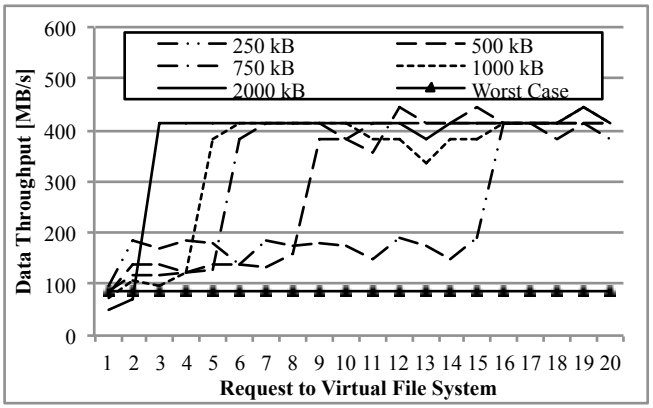

Figure 5. Data throughput of the caching strategy as a function of block size being read from the original file.

calling application and the virtual file system increases erratically, as soon as all data of a certain file can be read completely from the cache. The performance increases up to 410 MB/s corresponding to memory transfers between the virtual file system and a calling application. Again, a significantly higher performance is expected if the algorithms will be ported to a non-virtual file system.

VI. Conclusion

The strategy to substitute packets within a JPEG 2000 codestream with zero-packets offers scalability on file system level. By adding the proposed substitution strategy to the existing architecture of a virtual file system for scalable media, the data throughput – and therefore the response time – of the overall system can be improved. This behavior results in a file system application being able to respond in real-time, even if the storage device, containing the original files, is not fast enough. Depending on the amount of data, which can be read from the real storage device during one request, the information content of a virtual file differs.

Of course, the substitution strategy will change the representation of the image on the one hand. But on the other hand the strategy is able to deliver a representation of the image in a lower quality in the given time. Without applying the strategy, certain images cannot be read completely in the given time and the decoding application would be forced to skip at least the current frame if real-time is demanded.

The advantage of this strategy is that the file-inherent scalability is exploited not only in the decoding application, since it would not get the data in the required time. Rather, the file system determines that it is too slow for the request and takes advantage of the scalability. It is important that this procedure is completely transparent to a decoding application since the structure of the original file, as well as the virtual file structure generated by the substitution strategy, remains the same.

The caching strategy is optimized for the architecture of scalable media files, allowing for an increase in quality of pictures that are requested more than once. For the presented method it is elementary that the performance of the storage device can be predicted. The measured velocity can change continuously; read- and write-requests of the operating system – as well as competing requests from multiple decoding applications – have a direct impact on the performance of the used storage device. A dynamic block size – determined by the system itself – could further improve the performance of the overall system. This will be part of further research on this topic. The caching strategy is currently implemented and optimized for the image compression codec JPEG 2000, but should also be compatible to other scalable media codecs like H.264 SVC or MPEG4 SLC.

The measurements confirm that the developed methods perform as expected. Furthermore, they allow the use of known applications for scalable files on file system level, in order to avoid bottlenecks. The transfer of the algorithms in a non-virtual file system is appropriate and will be focussed due to further research. Particularly by using media files, that require very high data rates, the methods can help to ensure the real-time capability of playback systems.

References

[1] Digital Cinema Initiatives (DCI), ”Digital Cinema System Specification V1.2”, 7th March 2008

[2] ISO/IEC 15444-1, “Information technology – JPEG 2000 image coding system – Part 1: Core coding system”, 2000

[3] ISO/IEC 15444-1:2004/Amd 2:2009 , “Extended profiles for cinema and video production and archival applications”, 2009

[4] V. Bruns, H. Sparenberg, S. Fossel, “Video Decoder and Methods for Decoding a Sequence of Images”, Patent, unpublished

[5] V. Bruns, H. Sparenberg, A. Schmitt, Accelerating a JPEG2000 Coder with CUDA, 45th JPEG committee meeting, Poitiers, France, July 2008

[6] H. Schwarz, D. Marpe, T. Wiegand, “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard“ , IEEE Transactions on Circuits and Systems fpr Video Technology, Vol. 17, No.9, 9 Sep. 2007

[7] R. Yu, R. Geiger, S. Rahardja, J. Herre, X. Lin, H. Huang, “MPEG-4 Scalable to Lossless Audio Coding”, Audio Engineering Society, San Francisco, 28 October 2004

[8] Seagate Products & Services, 27th October 2011 <http://www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_7200_10.pdf>

[9] D. S. Taubman and M. W. Marcellin. JPEG2000: Image Compression Fundamentals, Standards and Practice. Kluwer Academic Publishers, 2002. , pp. 410–414.

[10] H. Sparenberg, A. Schmitt, R. Scheler, S. Foessel, K. Brandenburg, “Virtual File System for Scalable Media Formats”, 14th ITG Conference - Dortmunder Fernsehseminar, 2011