Transcoding Versus Multiple Image Versions in Image Servers with QoS Capabilities

Kjartan Rydland[1], Roger Midtstraum[1,2] and Josip Zoric[1]

Telenor Research and Development[1]
Trondheim, Norway

Department of Computer and Information Science[2]
NTNU, Trondheim, Norway

{Kjartan.Rydland, Josip.Zoric}@telenor.com, Roger.Midtstraum@idi.ntnu.no

Abstract

This paper considers image servers that has to adjust image size and image quality to the needs and capabilities of the client applications. We compare two different approaches to solve this problem. (1) Transcoding of images and (2) storing multiple versions of each image in the database. We find that, while transcoding provides great flexibility and easy image management, response times and maximum throughput are very disappointing compared to storing multiple versions. In our experiments, the latter aproch demonstrates a performance that is about 25 times better than the performance of transcoding.

1. Introduction

The Internet is a tremendous success and has changed the way information is distributed, provides new opportunities to offer services and changes the way business is done. Until now, the majority of users have accessed this network from a personal computer with a relative large screen and some sort of wired network connection. This situation is about to change quite radically as wireless technologies like Bluetooth and UMTS are provided on a commercial basis, and all sorts of small computers get a mobile network connection.

To make full use of the new wireless networks, applications will have to run on a much broader range of computers. Outside the working environment and the home office, users will access distributed services from a multitude of small, simple computers, like mobile phones and personal digital assistants (PDAs). This emerging situation is characterized by a much greater diversity in computing resources, for instance factors like screen size, input devices, main memory, secondary memory, processing power and network capacity.

Standardization groups like Parlay [5], ETSI [6], OSA [6] and 3GPP [7] have been working towards the new generation of multimedia enabled telecom services. A new generation of users is expanding the meaning of the service, demanding the shift from the passive role – focused on the classical receive of the services, to more active including user profiles, personalization, demand for their own domains, possibility to store and retrieve their own images. A new generation of mobile telecom devices – PDA like, will have personal cameras included, so the number of images generated and

sent will increase dramatically. This sets demands for more efficient management, retrieval and Quality of Service of images.

In this paper, we discuss how image servers can provide more flexibility and offer many versions of each image. In this way, client applications can choose image versions that, in image quality and file size, are tailored to the capabilities of the client computer and its network connection. To be able to provide different versions of each image, an image server can store a number of different image versions. We call this the "multiple image versions" approach. Alternatively, an image server can store one version of each image. As different image versions are requested, the server uses the stored images to compute appropriate image versions. This latter approach is known as "transcoding".

We analyse the properties and implications of the two approaches. As performance is of particular interest in a server, which is likely to handle many clients, it is also important to investigate the performance of each approach. In this case, it is relevant to consider both the time that clients have to wait for images (response time) and the maximum throughput (image requests per second) that can be handled by the server. This paper is based on work done in [4].

The remainder of this paper is organized as follows. Section 2 provides an overview of the system architecture for an image server, which provides quality of service. Section 3 provides necessary information about digital images and relevant image formats. In Section 4, we analyse the properties of the multiple image versions approach and the transcoding approach, while Section 5 presents the results from our experiments. Finally, Section 6 states our conclusions and provides some ideas for further work.

2. Image Servers and Quality of Service

Figure 1 gives an overview of the system architecture for a typical image server that stores image files and on request sends copies of image files to client applications connected to the same network. An image server consists of a number of secondary storage units (usually hard disks), which stores a number of digital image files, as well as the necessary metadata to manage and retrieve the image files. An Image Manager takes care of image management, handles requests from client applications and delivers the requested image files to the clients.

In order to provide various image qualities to different clients, the server need a QoS Manager, which communicates with the client applications to determine the appropriate image quality in each case. A single image can be delivered in different sizes, with different levels of compression and also in different image formats, e.g. TIFF, JPEG or GIF. Thus, a number of parameters can be adjusted to provide images that fit the needs and characteristics of each client computer and their network connection to the image server.

An adequate image quality can be determined on a per session basis by an initial QoS negotiation between client and server. Alternately, each image request can be tagged with information that determines the image quality requirements. In this paper, we make no assumptions on how this is done; we only assume that there is some means for the clients to state their capabilities.
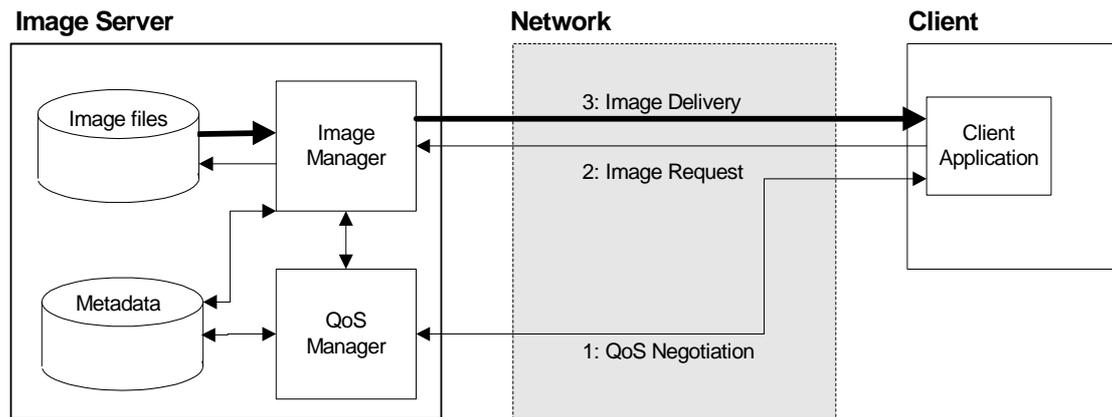
Figure 1, System architecture

In summary, we consider an image server, which store a number of digital images. This server is capable of delivering different versions of each image. This capability is used to deliver the most suitable image quality to each client application that requests images from the server. Before we discuss how this can be done, we give a brief introduction to digital images.

3. Digital Images and Image Formats

A digital image consists of a number of picture elements (pixels), which are arranged in a rectangular array. Each pixel in the array has a value that determines the colour of this picture element. The technical quality of a digital image depends on the number of picture elements and the number of different colours that can be chosen from when each pixel is assigned a colour value. The number of bits used to specify colour values is called the *bit depth* of the image. Today, it is common to have 24-bit pixels, which provides more than 16 million (2^{24}) different colour values.

The size of a digital image is the number of horizontal pixels times the number of vertical pixels times the bit depth of the image. In a high quality image there will be a lot of pixels, often many millions. Digital images can take a lot of storage space and they make high claims for network bandwidth when they are transferred over a network. For this reason, it is common to use compression techniques to store and transfer compressed image files that are much smaller than the original (uncompressed) image file. In order to achieve substantial savings in file size, it is common to use *lossy* compression techniques, which sacrifice some image quality for the sake of a more effective compression. The most common compressed image format for photographic images is JPEG [3], which use the Discrete Cosine Transform (DCT) in combination with entropy coding to achieve a very efficient compression of images.

Associated with a JPEG image is a "quality setting" or a "Q factor", which determines the degree that the image quality is degraded to achieve high compression. Figure 2 shows how the resulting file size varies as the Q factor is changed (1 being the best, giving the largest image files). In this paper, we also consider uncompressed images. For these images, we use the TIFF file format [3] without utilizing any kind of compression.
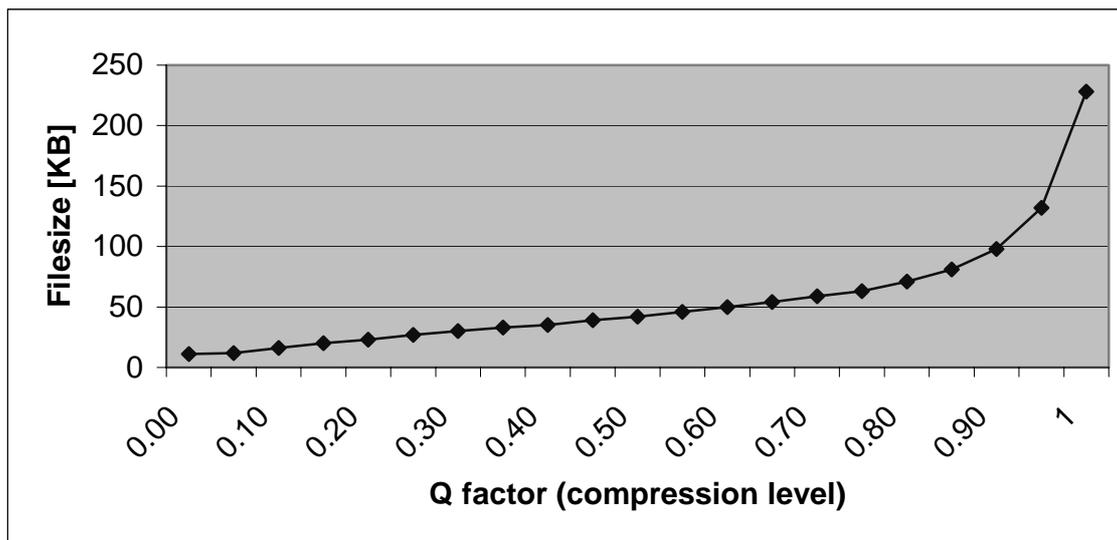
Figure 2, File size as a function of the degree of JPEG compression.

4. Approaches

As stated in the introduction, we will discuss two alternative ways to offer scalable image quality. Either, we have to store multiple versions of the same image, each copy with a different image quality, or we can store one original copy and compute the requested quality on demand. Each approach has its merits, which are discussed in this section.

4.1. Multiple Image Versions

Storing multiple versions of the images, each copy with a different image quality, two important considerations have to be answered:

1. How many versions of each image?
2. Which image qualities?

Two image qualities are given, the best image quality that might be requested and the worst. As the number of intermediate versions is increased, a finer adjustment to the client requirements is possible. But, at the same time, the size of the database grows. In practice, one has to find a suitable compromise between the number of quality levels and the need for extra storage space.

Image versions

In our experiments, we have chosen to consider a situation where we store JPEG images of the following five sizes:

- 400 x 300 pixels
- 640 x 480 pixels
- 800 x 600 pixels
- 1024 x 768 pixels
- 1280 x 960 pixels

Each image size is stored in three JPEG quality levels: 0.5, 0.7 and 0.9. This gives a total of 15 image versions, of different quality, for each image in the database. To make

a test database, we gathered 100 different images and multiplied them until we had 11 GB of image data.

		Image Quality			
		0,5	0,7	0,9	Uncompressed
Image size	400x300	14	19	35	352
	640x480	29	38	71	900
	800x600	48	64	120	1406
	1024x768	69	90	176	2304
	1280x960	95	126	226	3600

Table 1, Average image file size (KB) for JPEG images.

For the 100 original images in our test database, Table 1 shows the average file size for each image version. On average, the 15 versions of a single image take up 1.2 MB of storage. If an image of the best quality is stored without compression, the resulting file will have a size of

$$1280 * 960 \text{ pixels} * 3 \text{ Byte/pixel} = 3.5 \text{ MB}$$

If we store one uncompressed copy of each image, the overhead of storing 15 additional JPEG versions will be 1.2 MB. This is a 34 percent increase in the database size. However, if we abandon the uncompressed image version, the overhead would be the 14 versions in addition to the best JPEG version. In this case, we would use 225 KB for the largest version and 996 KB for the rest. Thus, the relative overhead is much worse at 443 percent, while the total storage consumption per image is reduced from 4.7 MB to 1.2 MB.

Management of image versions

In addition to the increase of the size of the database, storing multiple image versions also complicates the management of the image database in three important ways. First, the server has to store information regarding each of the available copies of each image. Second, the insertion of a new image becomes more complicated, as a number of image versions have to be computed and inserted into the database. Third, changes in image content have to be propagated to every image version. Most likely, these image versions have to be re-computed from the modified original.

Retrieving images

When a client requests an image and the adequate image version is determined, the image server only has to read the requested image version from disk and forward it to the client, without any further processing at the server side. This means that disk I/O is going to be the critical resource in the system.

Therefore, it is an important property that a system with multiple image versions will consume less disk I/O capacity than a system with a single image quality. In an image server where all images are of the best JPEG quality (1280x960, compression level 0.9), each image request will make the disk read and transfer 225 KB. If we anticipate that our 15 image qualities are requested with equal probability, the average image request would make the disk read and transfer 81 KB (the average image file size). Hence,

under reasonable assumptions, an image server with multiple image versions would be able to serve more clients than a server with a single image quality.

While disk transfer time is a linear function of image size, the disk access time is a complex mix of seek time, rotational delay and actual read time [2]. As disks are more effective when they read bigger chunks of data, the differences in total access times, for different file sizes, are not as big as the file size would indicate. In Table 2, we give the average disk read times for the different image qualities when the image files were stored on a 10000 RPM IBM Ultrastar disk with an Ultra 160 SCSI interface. Taking an image of 800x600 as an example, the largest file is 2.5 times larger than the smallest, while the longest read time is only 1.6 times longer than the shortest read time.

In situations where the disk never experiences concurrent requests for image files and the 15 different versions are accessed with equal probability, the average access time is around 47 ms.

		Image Quality		
		0,5	**0,7**	**0,9**
	400x300	29	30	34
	640x480	31	34	44
Image Size	**800x600**	34	40	56
	1024x768	42	51	73
	1280x960	52	60	91

Table 2, Disk access times (ms) for image files retrieved from an otherwise idle disk.

Conclusions

The main properties of the multiple image versions approach can be summarised as follows:

- Relative high storage overhead.
- Minimal disk access time and minimal use of disk transfer capacity.
- Complex management of images, especially handling updates to the image contents.
- Fixed image qualities.

In Section 5 Experiments, we examine the response times and the throughput of a multiple versions server when the load on the server is increased.

4.2. Transcoding

When we use transcoding, the image server reads a source file from the database and re-codes this image file into a target file of a requested image quality. This process involves the following steps:

1. Read source file from disk.

2. Decompress (if necessary).

3. Scale image to requested size (if necessary).

4. Compress (if necessary) and make a (target) file of the requested image format.

5. Forward the target file to the requesting client.

Selecting source files

It is an important decision to determine which type of images should be stored in the database and, hence, being used as source images in the transcoding process. To achieve the least quality degradation an uncompressed image format should be used for the source files. In table 3, we show transcoding times, including the disk reads, when (uncompressed) TIFF images (1280x960) were used as source files to produce the same JPEG image versions that we had in the multiple image versions approach. Routines from Java Advanced Imaging were used to do the necessary image processing.

		Image Quality		
		0,5	**0,7**	**0,9**
Image Size	**400x300**	875	867	868
	640x480	971	969	985
	800x600	907	918	941
	1024x768	1182	1201	1223
	1280x960	1551	1513	1557

Table 3, Time (ms) to read and transcode TIFF image files (1280x960 pixels) into JPEG image files in an otherwise idle system.

As an alternative solution, we tried using files of the the best JPEG quality as source files. This saves disk resources at the expence of a decompression step. For target images of JPEG quality 0.7, this resulted in times that were around 200 ms longer than corresponding times using using TIFF files. Considering disk space, the storage of the JPEG images use less than 1/10 of the disk space necessary to store the TIFF images. Because we focus on retrieval performance, we use only TIFF images as source images in the rest of this paper.

Resource consumption

At the same load, transcoding from TIFF images will use more disk transfer resources than the multiple versions approach. For each image request, a 3,6 MB TIFF file has to be read from disk, taking around 300 ms at low disk traffic. However, independent of target image quality, we have found that reading from disk never taks more than one third of the total response time. Providing more disk transfer capasity would therefore be of little use as long as processing power is the limiting resource in the system.

In the system used in the our experiments, the scaling and compression steps are so demanding on processor cycles that the processor has no spare capasity, even when only one image is requested at the same time. When the traffic increases, more images are requested at the same time, and we experience at least a linear increase in respones times. With our current implementation, the image processing steps are so time consuming that they seriously reduce the practical feasibility of the transcoding approach.

Management

With transcoding, a single copy of each image is stored in the database. This simplifies the management of images and also updates can be done in a single image

file. Transcoding provides greate flexibility as any image quality can be computed, as long as it does not exceed the quality of the source image.

Using transcoding is a simple way to provide QoS for an existing image database because we do not have to change the database contens. It will be sufficient to add software components that handles the QoS management and takes care of the transcoding. Another situation where transcoding would be especially appropriate, are cases where images are published as soon as they are produced.

Conclusions

The main properties of the transcoding approach can be summarised as follows:

- Limited by the size and quality of the source images, any image quality can be provided.

- Very demanding on processor resources.

- No storage overhead due to multiple image versions, but use a lot of disk I/O capasity.

- High storage demands if uncompressed image files are stored in the database and used as input into the transcoding process.

- Easy management of the image database.

- Easy to add to an existing image database.

- Very long response times, even when the server is almost idle.

5. Experiments

To compare the properties of the two approaches when the server has to serve concurrent client requests for images, we measured the response time at different loads. These experiments were performed on a PC with a 700 MHz AMD Athlon processor and 256 MB of memory, running Windows 2000. A total of 11 GB of image files were stored on an 18 GB 10.000 RPM IBM Ultrastar disk with an Ultra 160 SCSI interface. The server and the clients were implemented in Java. Image processing was done with Java Advanced Imaging version 1.1, where the codec is implemented in C.

In these experiments, the clients request the 15 different image qualities that were defined in Section 4.1. All image versions were requested with equal probability. In all experiments, we measured the time from a client requested an image until it was delivered from the server. Clients and server ran on the same computer. In this way, we did not have to consider effects related to the network.

5.1. Multiple image versions

The discussion in Section 4 considers a server, which does not experience concurrent image requests. In order to find out how an increased load (number of requested images per second) affects the response time, we set up an experiment where we varied the load from 13 requests per second (on average 75 ms between requests) to 2 requests per seconds. The times between consecutive image requests (so called inter-arrival times) were drawn from an exponential distribution [1].
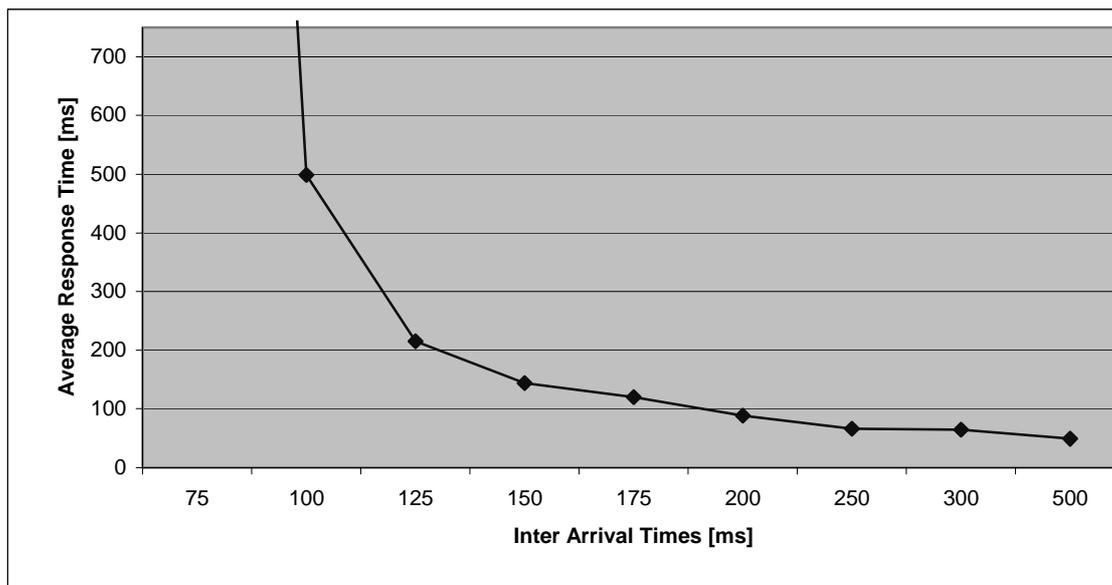
Figure 3, Response time as a function of system load for a server with multiple image versions.

Figure 3 shows the results from these measurements. At low load the response time is 49 ms, which is in accordance with the analytical result of 47 ms for a server that can fulfill each request before it gets a new one. When the load exceeds 10 requests per second (average inter-arrival time between requests is 100 ms), the response time skyrockets (at an inter-arrival time of 75 ms the average response time is 3.6 seconds). With 10 requests per second, the average response time is 0.5 second. Hence, if one wants to keep the response time below 0.5 second, the maximum throughput that the system can sustain is 10 image requests per second.

To increase the maximum throughput of the image server, one has to improve the disk I/O capacity. There is little to be gained by shopping for a faster disk. The I/O capacity should rather be improved by spreading the image files on more disks, for instance using a RAID configuration or running the image server on a cluster of computers with separate disks.

5.2. Transcoding

With transcoding, the processor is the limiting resource in the image server. Performing the same kind of experiment as for multiple versions, we measured response times for loads ranging from two requests in three seconds (1500 ms between requests) to one request every fifth second (5000 ms between requests). Figure 4 shows the results from these measurements. In this case, uncompressed TIFF images (1280x960 pixels) were used as source images. As can be seen in the figure, the response times are very high, ranging from 1.3 second to 3.7 seconds.

In the previous section, we found that a multiple image versions server could handle up to 10 requests per second without exceeding a response time of 0.5 second. A transcoding server is not able to provide an average response time of 0.5 second, no matter how few request are received per second. If we limit the response time to be less than 2.5 seconds, a transcoding image server can sustain a maximum throughput of (only) 0.5 request per second.
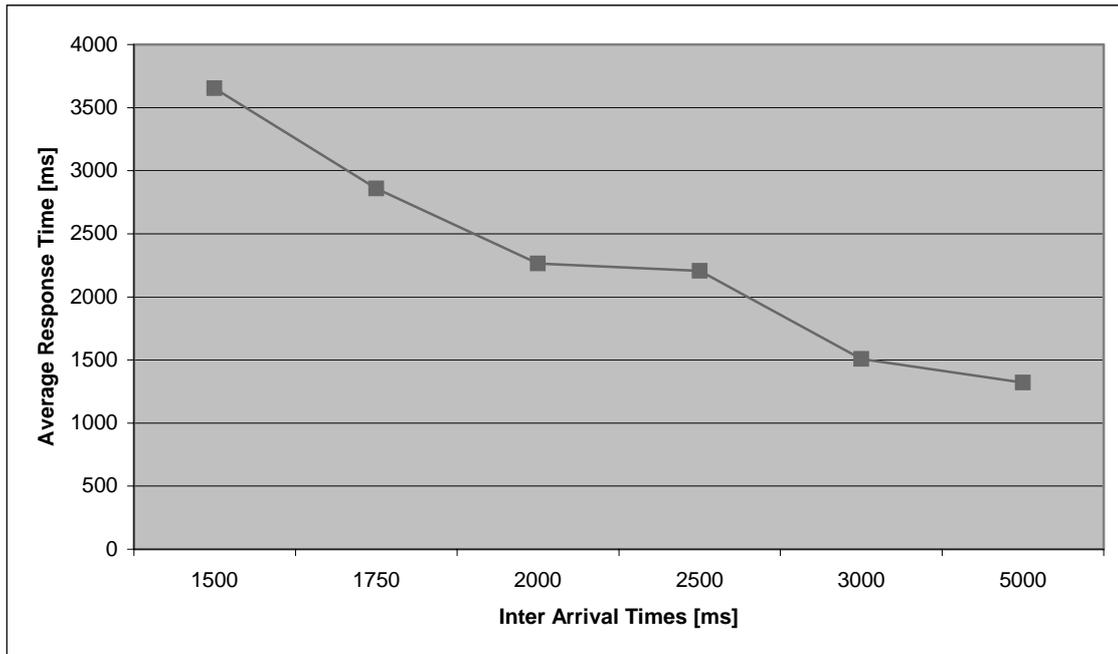
Figure 4, Response times as a function of system load for a server using transcoding.

In order to use transcoding in a high capacity server, one either has to improve the computing resources or make the transcoding more efficient or do both. The computing resources can be improved by using a more powerful processor or, more effectively, by adding more processors. The transcoding can possibly be made more efficient by use of a better codec or by use of special purpose hardware solutions. These possibilities have not yet been studied further.

5.3. Transcoding versus multiple image versions

The previous two sections have shown that the multiple image versions approach is superior when it comes to response time and maximum throughput. I we require that the response time should not exceed 2.5 seconds, a multiple version server can handle 12 requests per second, while a transcoding server can handle only 0.5 request per second. Hence, in situations like the one we have considered, a multiple versions server can service 24 times more clients than a transcoding server.

6. Conclusion

In this paper, we have considered two approaches for building an image server, which can adjust the delivered image quality. Transcoding provides the most flexible solution, incurs no storage overhead and allows easy management of the image database. On the negative side, transcoding maximises disk I/O and is very demanding regarding processor cycles. Storing multiple versions of each image increases the demand for disk space and complicates the management of the image database. This approach provides less flexibility than transcoding, as only the pre-stored image qualities can be delivered. However, the multiple version approach minimises disk I/O and is very lean on processor cycles.

Our experiments show that transcoding gives high response times, regardless of the load on the system, and a very low maximum throughput. Storing multiple image

versions gives much lower response times and a much higher maximum throughput, in our case up to 24 times as many image requests per second. Hence, use of transcoding should only be considered when short response times and high throughput are of little importance.

6.1. Future work

In all our considerations, we have assumed that every image size and quality is equally likely to be requested. In reality, one can expect to experience skewed distributions. If a few image versions dominate, the two approaches can be combined. In this case, one would store only the popular versions and make use of transcoding to provide the less popular versions. Depending on the skewness, this combination can be nearly as efficient as a pure multiple version approach, while possessing the flexibility of the transcoding approach.

Other ideas could be to investigate whether other codecs, use of special purpose hardware or other compression methods could make the transcoding approach more feasible for practical use.

References

[1] J. Banks, J.S. Carson and B.L. Nelson, *Discrete-Event System Simulation*, Prentice-Hall, 1996.

[2] Kjell Bratbergsengen, *Filsystemer – Lagring og behandling av store datamengder*, Department of Computer and Information Science, NTNU, 1997.

[3] James D. Murray and William vanRyper, *Encyclopedia of Graphics File Formats*, O'Reilly & Associates, Inc., 1994.

[4] Kjartan Rydland, *Tjenestekvalitet for distribuerte bildeapplikasjoner*, Hovedoppgave, Department of Computer and Information Science, NTNU, March 2001.

[5] http://www.parlay.org/

[6] http://www.etsi.org/

[7] http://www.3gpp.org/