DISTRIBUTED ENCODING ENVIRONMENT BASED ON GRIDS AND IBP INFRASTRUCTURE

Petr Holub[*†] **and Lukáš Hejtmánek**[*]

[*]Faculty of Informatics and [†]Institute of Computer Science,
Masaryk University Brno, Botanická 68a, 602 00 Brno, Czech Republic

e-mail: hopet@ics.muni.cz, xhejtman@mail.muni.cz

Abstract

This paper introduces an environment for distributed video transcoding based on Grid computing infrastructure and Internet Backplane Protocol storage infrastructure. A model for scheduling jobs with respect to location of data in distributed storage is introduced for optimal processing performance. We describe our abstraction library `libxio` *providing a uniform access interface to both local files and data stored on IBP. Modifications of video tools done based on this library in order to work with IBP are being presented as well as a system that controls an overall distributed encoding process and helps scheduling system to schedule jobs in agreement with the distribution of data in IBP. Two pilot groups that use this system for preparing video content for streaming are mentioned.*

1 Introduction

In recent years there has been a growing demand for creating video archives available on the Internet ranging from archives of university lectures [1, 2] through archives of medical videos and scientific experiments recordings. Pioneering this effort in the academic sphere in the Czech Republic we are facing an ever increasing demand for both computational processing power and large storage capacity.

Instead of buying expensive dedicated and closed solutions we have decided to opt for the development of our own open and flexible system for distributed video encoding based on the powerful Grid infrastructure made available in Czech Republic by the MetaCenter project [3]. The MetaCenter project was enhanced during year 2003 by a new project called Distributed Data Storage (DiDaS) incorporating new distributed storage based on an Internet Backplane Protocol (IBP) [4]. Such storage infrastructure can be efficiently used for distributed computing that processes large volumes of data which is exactly what is needed for distributed video processing. However as the scheduling system in use is not capable of scheduling jobs with respect to location of the data and there is also neither data location optimisation nor prefetch functionality and our data-intensive application requires at least some of them for optimal performance, we had to enhance the underlying infrastructure.

The rest of the paper is organised as follows: Section 2 gives an overview of the infrastructure and tools used for building the encoding environment, Section 3 shows a theoretical model we use for optimisation of computing with respect to location of the data, Section 4 details the implementation of the encoding environment, Section 5 briefs pilot applications, and Section 6 gives some direction for the planned future development of the whole system.

2 Technical Background

As a part of the Grid infrastructure the MetaCenter provides great computational power in the form of IA32 Linux PC clusters that are being rapidly expanded each year because of the cost-efficiency of this solution. The filesystem shared across these clusters is based either on a rather slow AFS filesystem (several terabytes of storage are available) or on a somewhat faster NFS which has its own problems such as broken support for sharing files larger than 2 GB and a capacity on order of only a few tens of gigabytes available. Therefore we need a different means of storage for processing large volumes of data.

The DiDaS project has been launched specifically to create a distributed data storage system using the IBP. The IBP uses soft consistency model with a time-limited allocation thus operating in *best effort* mode. A basic atomic unit of the IBP is a byte array providing an abstraction independent of the physical device the data is stored on. An IBP depot (server), which provides set of byte arrays for storage, is the basic building block of the IBP infrastructure offering disk capacity. By mid 2004 the IBP data depots should be present in all cluster locations as well as distributed across other locations in the Czech academic network.

The PBS Pro [5] scheduling system is used for job scheduling across the whole MetaCenter cluster infrastructure. The PBS supports queue based scheduling as well as properties that can be used for limiting where a job may be run based on user requirements. These properties are static and defined on per-node basis. Under ideal circumstances the PBS is capable of in-advance reservations and estimate of time when a specified node is available for scheduling new jobs unless a priority job is submitted. The latter feature requires cooperation with users that submit their jobs as the PBS needs an estimate of processing time provided by the job owner for each job—otherwise the maximum time for the specified queue is used and this results in non-realistic estimate of when a specified processing node will be available.

For video processing we use the `transcode` tool [6], which features transcoding between many common formats. Unfortunately this tool is unable to directly produce RealMedia format, which is one of few streaming formats with strong multi-platform support and which is also the format of choice for the CESNET video archive and the Masaryk University lecture archive. Therefore we also need to use Helix Producer [7] to create the required target format. The Helix Producer needs raw video with PCM sound as input file and as this format is rarely the format of the input video, we use the `transcode` for pre-processing data for the Helix Producer.

3 Scheduling Model

For efficient processing of data stored in a distributed storage infrastructure we consider the following functionality as crucial in order to achieve efficient processing:

- *Selection of best hosts to perform the processing.* The "best" host does not need to be the fastest one. Actually, it is the one on which the computation finishes in the shortest time. Thus we need to make estimate of completion time for the tasks on each host and then select hosts accordingly.
- *Data location optimisation and prefetch support.* Data location optimisation means we move or copy the data inside the distributed storage infrastructure (using data replicas) to improve performance of the processing. The prefetch functionality means downloading the data to be processed into the processing node in advance. Some evaluation criteria are needed to decide whether prefetch is needed/helpful or not. The minimum condition is that the data location optimisation or data prefetch must accelerate the processing.

In this section we introduce the algorithm we have designed for efficient job scheduling meeting requirements mentioned above while taking real-world problems into account.

Let's assume we have a set of processing nodes P and for each processor $p \in P$ we have an estimate of processing power $s_{p,u}$ for task u which is split into chunks with length of l_u. The scheduling system can provide information on when each processor p will be available ($t_p^{\text{sched\_free}}$) and we also need an estimate of available bandwidth between a distributed storage depot set D and the processor p in time t which will be denoted as $b_{D,p}(t)$. Data movement in the opposite direction (meaning uploading data from processor p to depot set D) will be denoted $b_{p,D}(t)$.

In general, Completion Time Estimate (CTE) can be obtained by solving the following equations for the given location of the data in depots D, using processor p and uploading resulting data of length l_u^{out} into depot set D'

$$\int\limits_{t_p^{\text{sched\_free}}}^{\text{CTE}_d\,(p,D,u)} \min\{s_{p,u}, b_{D,p}(t)\}\;\mathrm{d}t = l_u \tag{1}$$

$$\int\limits_{t_p^{\text{sched\_free}}}^{\text{CTE}_u\,(p,D,u)} b_{p,D'}(t)\;\mathrm{d}t = l_u^{\text{out}} \tag{2}$$

$$\text{CTE}\,(p,D,u) = \max\{\text{CTE}_d\,(p,D,u), \text{CTE}_u\,(p,D,u)\} \tag{3}$$

This model also presumes that the uploading into the storage infrastructure takes place in parallel with downloading otherwise the lower bound in (2) needs to be modified accordingly.

If we assume that $b_{p,D}(t)$ is constant in the interval $\left\langle t_p^{\text{sched\_free}},\ \text{CTE}\,(p,D,u)\right\rangle$ (which can be justified since job duration is less than time resolution of network traffic prediction), we get the following relation

$$\text{CTE}\,(p,D,u) = t_p^{\text{sched\_free}} + \max\left\{\frac{l_u}{\min\{s_{p,u}, b_{D,p}(t_p^{\text{sched\_free}})\}}, \frac{l_u^{\text{out}}}{b_{p,D'}}\right\} \tag{4}$$

To simplify the model even further, we can assume that the uploading into the infrastructure is not the bottleneck since $l_u^{\text{out}} << l_u$ while $b_{D,p} \approx b_{p,D'}$ or that the uploading phase takes negligible time only compared to downloading and processing if the uploading occurs after the processing. Thus we obtain the formula that will be used further on in this paper for the sake of simplicity

$$\text{CTE}\,(p, D, u) = t_p^{\text{sched\_free}} + \frac{l_u}{\min\{s_{p,u}, b_{D,p}(t_p^{\text{sched\_free}})\}} \tag{5}$$

In case that the presumption of negligible uploading time is not valid, the model and the resulting algorithm can be easily extended to support it.

Now the problematic part remains the $b_{p,D}(t)$ estimate. In certain situations this value can be obtained from network traffic prediction service like Network Weather Service (NWS) [8, 9]. However, there are important limitations as there are no available services known to the authors that can provide advanced enough functionality[1]. When no prediction service is available, we can either measure average available bandwidth for the TCP protocol (or another protocol used for data transfer) instead and take it as a constant value or we have to assume that the network and the storage are not the bottleneck.

3.1 Job Scheduling Algorithm

A general job scheduling algorithm, which assumes processors of uniform processing power and jobs of different size, belongs to *NPO* class. However our model assumes processors of different speeds and tasks of uniform size as the multimedia transcoding can be divided into number of uniform chunks (with exception of the last chunk) as shown in Section 4.

Taking advantage of the uniform size of the tasks, we suggest using a greedy algorithm shown in Figure 1. It can be shown that it belongs to *PO* class and it provides an optimum solution for scheduling jobs on processors. It can be also shown that it belongs to *PO* class if storage depots and processors are connected via a complete graph since data replicas can utilise each depot to the maximum extent. In case of a common graph it belongs to *NPO* class again as the greedy algorithm can prevent maximum utilisation of some depots. For example when the second fastest processor is connected only to the fastest depot while the fastest processor is connected to all depots and the processor is fast enough to utilise the fastest depot to its maximum then the second fastest processor has no access to the free depot.

The algorithm also provides requests for the data location optimisation by creating a new data replica for task u in the distributed storage infrastructure if processing capacity of the processor is larger than available bandwidth as shown on lines 14–17. If the bandwidth between a storage depot set and processor is still lower than available

[1] There are three most important issues: prediction in N-to-1 instead of 1-to-1 fashion (as data can be downloaded from or uploaded to multiple storage depots), in advance traffic reservation (for already scheduled jobs), and possibility of isolating regular traffic (as the processing can be regular and thus it can be already included in traffic prediction). An additional problem is that we need the prediction in an end-to-end manner, meaning that if the bottleneck is on the depot or processing node itself, the available bandwidth prediction gets decreased correspondingly. This allows avoiding unnecessary replicas when the bottleneck is on the processing node and thus prefetch should be employed instead.

```
1  foreach $u \in U$ do
2     $D_u := \emptyset$;
3  od
4  foreach $d \in D$ do
5     $P_d := \emptyset$;
6  od
7  foreach $p \in P$ do
8     $t_p^{\mathrm{sched\_free}} := \mathrm{PBS\_free}(p)$;
9     $\mathrm{ProcTime}(p) := \frac{l_u}{\min\{s_{p,u}, b_{D,p}(t_p^{\mathrm{sched\_free}})\}}$;
10     /* assuming uniform size tasks $u \in U$ */
11 od
12 foreach $u \in U$ do
13    $p : \forall p_1 \in P.\ t_{p_1}^{\mathrm{sched\_free}} + \mathrm{ProcTime}(p_1) \geq t_p^{\mathrm{sched\_free}} + \mathrm{ProcTime}(p)$;
14    while $(s_{p,u} > b_{D_u,p}\ \wedge \exists$ free depot) do
15          $d : \forall d_1 \in D.\ b_{d_1,p}(t_p^{\mathrm{sched\_free}}) \leq b_{d,p}(t_p^{\mathrm{sched\_free}})$;
16          $\mathrm{sched\_depot}(u, d)$;        /* $P_d := P_d \cup \{u\};\ D_u := D_u \cup \{d\};$ */
17    od
18    if $s_{p,u} > b_{D_u,p} \wedge \nexists$ free depot
19       then $\mathrm{prefetch}(p, u)$; fi
20    $\mathrm{PBS\_sched}(p, u)$;
21    $t_p^{\mathrm{sched\_free}} := \mathrm{PBS\_free}(p)$;
22 od
```

Figure 1: Simplified job scheduling algorithm with multiple storage depots per processor used for downloading (i. e. N-to-1 data transfer) and neglecting the uploading overhead.

processing power and no improvement can be achieved by making additional replicas, the algorithm can initiate prefetch of data into storage capacity local to the processing node (lines 18–19).

4 Implementation Overview

The implementation is composed of two parts: enabling applications to work with data in the IBP storage and a system that handles parallel encoding of multimedia content.

4.1 Access to IBP Infrastructure

We have developed a general purpose abstraction library called `libxio` [10] that provides an interface closely resembling standard UN*X I/O interface allowing developers to easily add IBP capabilities into their applications providing access to both local files and files stored in IBP infrastructure represented by IBP URI. The library provides equivalents for the following standard I/O functions: `open`, `close`, `read`, `write`, `fttruncate`, `lseek`, `stat`, `fstat`, and `lstat`.

The IBP URI has the following format:

```
lors://host:port/local_path/file?bs=number
  &duration=number&copies=number&threads=number
  &timeout=number&servers=number&size=number
```

where the `host` parameter is a specification of an L-Bone server (IBP directory server) to be used, the `port` is a specification of an L-Bone server port (default is 6767), the `bs` is a specification of block-size for transfer in megabytes (default value is 10 MB), the `duration` specifies allocation duration in seconds (default is 3600 s), requested number of replicas is specified by the `copies` (defaulting to 1), the `threads` specifies number of threads (concurrent TCP streams) to be used (default is 1), the `timeout` parameter is specification of timeout in seconds (defaulting to 100 s), the `servers` parameter specifies number of different IBP depots to be used (default is 1), and the `size` specifies projected size of file to ensure that the IBP depot has enough free storage. It is possible to override default values using environment variables, too. If the given filename doesn't start with the `lors://` prefix, the `local_path/file` is accessed as local file instead.

When writing a file into the IBP infrastructure the `local_path/file` specifies the local file where a serialised XML representation of the file in IBP will be stored. We advise users to use the `.xnd` suffix for files representing data in IBP. This file is the only representation of the data in IBP infrastructure and thus keeping it safe is important as long as the data is needed. In our case we store the XML representation files in an AFS directory which is available on all clusters and which is also regularly backed up. At least an L-Bone server must be specified when writing a file into IBP. In our experience the file with serialised representation (meta-data) occupies approximately 1/10th of the actual data size in IBP on average.

When a file stored in IBP is read, the `local_path/file` specifies the local file containing a serialised XML representation of the IBP file. The user can also use a short form URI `lors:///local_path/file` as the servers are already specified in local XML representation.

Based on `libxio` library we have enabled IBP capabilities in two applications [10]. The `transcode` program has been patched so that it can load and store files to/from IBP depots. As the `transcode` has very modular internals and some file operations are implemented inside libraries for certain file formats, it is necessary to patch such libraries as well. Currently we have patched the `libquicktime` library as QuickTime MOV files are common products of editing software (e. g., AVID Xpress provides fast codecs for producing DV video wrapped in a QuickTime envelope which can be processed by `transcode` when `libquicktime` is slightly patched).

The second application is the Mplayer media player that is able to play content directly from the IBP. However, there is a serious problem because of the single-threaded Mplayer architecture that results in a latency problem when files in IBP are accessed. Because of its architecture the Mplayer is also doomed to downloading data from the IBP using only a single thread.

4.2 Distributed Encoding Environment

We have created an umbrella system that allows users to easily encode video in a distributed manner called Distributed Encoding Environment (DEE). An input video (typically in DV format with an AVI or QuickTime envelope produced by AVID or Adobe video editing software) gets uploaded into the IBP infrastructure from the editor's workstation first. The video is then taken from IBP by IBP-enabled `transcode`, remultiplexed to ensure proper interleaving of audio and video streams, split into smaller chunks which are uploaded directly to IBP and encoded on many cluster worker nodes in parallel (see Fig. 2). At this stage the DEE system uses the PBS system to submit the parallel jobs on worker nodes.

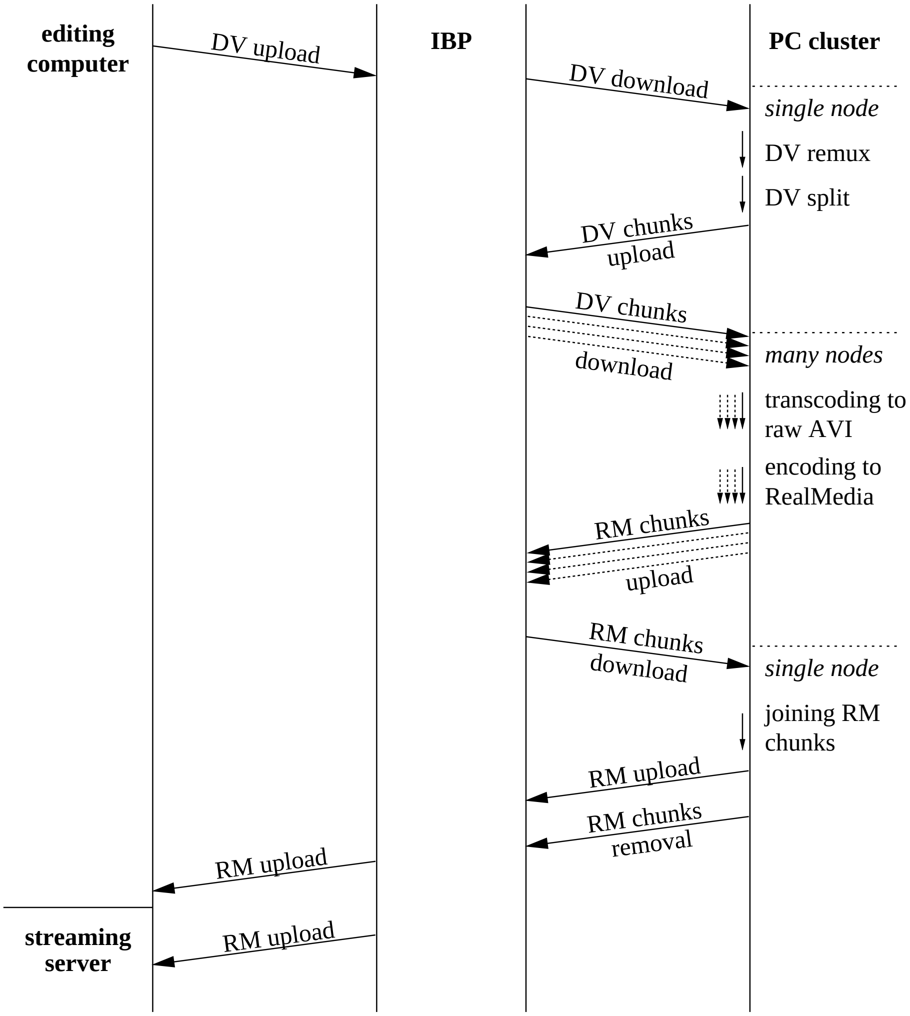

Figure 2: Distributed Encoding Environment Architecture.

During the parallel processing phase, each video chunk is first transcoded to raw video with PCM sound since this format is required by the Linux version of Helix Producer.

All required transformations are performed at this step, typically including high quality deinterlacing and resizing, audio resampling, and possibly noise reductions (if the original file is very noisy due to a low light level or due to low-quality camera used). This parallel phase uses storage capacity local to each worker node to create and process the raw video file. The raw file is then fed into the Helix Producer and the resulting chunk of RealMedia video is stored back into the IBP. As a final step the RealMedia chunks are joined, the complete file is stored in the IBP, and the RealMedia chunks are removed.

When the Helix Producer is replaced by various transcode output modules or when the raw video is piped into another encoding program the DEE system can be used also for producing other video formats: DivX version 4 and 5, MPEG-1, MPEG-2, etc. Also when no intermediate (raw) file is needed in the encoding process, the system can directly transcode the data from the IBP while simultaneously uploading results back into the IBP.

Since there is no direct support for dynamic properties like location of files in the IBP infrastructure available in the PBS scheduling system and there is currently also no network traffic prediction service available in the MetaCenter infrastructure, we have defined some static properties for the computing nodes that allow us to assess the proximity of the computing nodes to the IBP storage depots. Computing nodes which have the same processing characteristics and share the same network connection are given some static attributes. We measure static average estimate of bandwidth available between each such set of nodes and each storage depot. The DEE system then uses these properties and locations of files to be transcoded and gives hints to PBS where the jobs should be run according to the algorithm shown in Figure 1. It can also initiate replication of the data in the IBP infrastructure when processing power of the node is higher than network capacity from the IBP depots that keep the data.

5 Pilot User Groups

A pilot test group participated from the Faculty of Informatics, Masaryk University, which has provided recordings of many lectures at the faculty for the last two years [11]. This group needs to process more than 20 hours of video per week to produce files for both streaming and downloading. The RealMedia format is used for streaming in two quality levels. One is for students with low bandwidth connectivity and low processing power resulting in files with quarter PAL resolution (384×288) at 15 frames per second with supported bitrates ranging from 56 to 768 kbps. The other one is for students with high bandwidth and processing power available resulting in video with full PAL resolution and frame rate (768×576 @ 25 fps) at a bitrate slightly below 3 Mbps. For downloading, DivX format files are produced in such way that one lecture fits one CD. At the beginning of year 2004, three new lecture halls have been opened that can be used to continuously capture and record all lectures taking place there in a routine way. The analogue sound and video are converted into DV format using Canopus ADVC-100 boxes, the DV video is captured via IEEE-1394 interface and it is uploaded into the IBP infrastructure for processing by one computer per lecture hall. Thus even more processing power as well as storage capacity is needed starting this year.

The other pilot group is a team from neurosurgery department at the St. Anna University Hospital in Brno. The neurosurgeons have large archives of neurosurgical operations they want to make available for students of medicine. High quality video with low compression is required to preserve value of these recordings as even slight nuances may be significant for diagnosing patient.

After the project advances from pilot operation to production status we expect its adoption by the wider Czech academic community.

6 Future Work

We would like to focus our attention on integrating the new generation of Grid scheduling systems that allows jobs to be scheduled with respect to location of processed data directly (e. g., the resource brokering system from the EU DataGrid project). After more thorough analysis we want to initiate development of a new network traffic prediction service that will be suitable for using with distributed data storage, regularly occurring traffic, and that will support in-advance bandwidth allocations. We would also like to create a graphical user interface to our distributed encoding system that makes the system available even for less technically skilled users.

7 Acknowledgements

This work is supported by CESNET Development Foundation projects 017/2002 and 018/2002. The authors would like to thank to Luděk Matyska, Miloš Liška, and Eva Hladká for supporting our work and for stimulating discussions.

References

[1] *Standford Online and Online Classroom*. `http://scpd.stanford.edu/scpd/about/delivery/stanfordOnline.htm`.

[2] *Berkley Lecture Browser*, 2002. `http://bmrc.berkeley.edu`.

[3] *MetaCenter Project, CESNET*. `http://meta.cesnet.cz/`.

[4] M. Beck, T. Moore, and J. S. Planck: “An end-to-end approach to globally scalable network storage.” In *SIGCOMM’02*, 2002. `http://loci.cs.utk.edu/ibp/files/pdf/SIGCOMM02p1783-beck.pdf`.

[5] *Portable Batch System*. Altair Engineering, `http://www.pbspro.com/`.

[6] T. Oestreich and T. Bitterberg: *`transcode` – Linux Video Stream Processing Tool*. `http://www.theorie.physik.uni-goettingen.de/~ostreich/transcode/`.

[7] *Helix Community products: Helix DNA Client, Helix DNA Producer, and Helix DNA Server*. `http://www.helixcommunity.org/`.

[8] R. Wolski: “Dynamically forecasting network performance using the network weather service,” Cluster Computing, 1(1):119–132, 1998.

`http://www.cs.ucsb.edu/˜rich/publications/nws-tr.ps.gz`.

[9] R. Wolski, N. T. Spring, and J. Hayes: “The network weather service: a distributed resource performance forecasting service for metacomputing,” Future Generation Computer Systems, 15(5–6):757–768, 1999. `http://www.cs.ucsb.edu/˜rich/publications/nws-arch.ps.gz`.

[10] L. Hejtmánek and P. Holub: *IBP deployment tests and integration with DiDaS project*. Technical Report 20/2003, CESNET, 2003.

[11] E. Hladká and M. Liška: “Přednášky ze záznamu (Lecture Recording).” In *Širokopásmové sítě a jejich aplikace (Broadband Networks and Their Applications)*, pages 130–133, CVP UP Olomouc, 2003. ISBN 80-244-0642-X.