

# **IBP Deployment Tests and Integration with DiDaS Project**

**Lukáš Hejtmánek, Petr Holub**

2003-11-23

## **1 Abstract**

In this report we describe testing setup of Internet Backplane Protocol (IBP) and its integration with Distributed Data Storage project (DiDaS). We give a short overview of IBP followed by description of developer interfaces for IBP as well as end user tools for manipulating files stored in IBP infrastructure. We have created a library for developers that offers easy interface to IBP resembling traditional UN\*X file operation calls and we have also implanted IBP capabilities into several end user software tools using this library. RAID arrays benchmarks and preliminary IBP benchmarks are presented as well.

## **2 Introduction**

In late 2002 CESNET Development Fund decided to support a project for building distributed data storage called DiDaS as an extension of Grid activities of MetaCenter project [Meta]. The DiDaS project aims to build large distributed storage capacity based on probably the most extensive project in this field: Global Distributed Network Storage developed at University of Tennessee, Knoxville [SIG02], [IBP]. Furthermore DiDaS project addresses support several pilot user communities that will use the storage infrastructure.

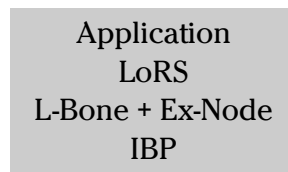
One of these pilot groups is a project developing Distributed Encoding Environment [DEE] for multimedia processing, which will use both DiDaS storage capacity and MetaCenter computing capacity (esp. Linux PC clusters that are available at large now). Contemporary video processing tools suffer from low processing power and low disk capacity available on single computer. Building large multiprocessor systems with shared memory and large disk capacity attached locally is fairly expensive while video transcoding can be run using more cost-efficient PC cluster environment very well. PC clusters can be distributed across various sites (which holds true for MetaCenter PC clusters) and thus distributed storage capacity is very natural solution for storing video that

is to be transcoded, as well as for storing intermediate temporary files resulting from transcoding process and final encoded video. Coordination between distributed storage and scheduling systems on some level is very promising as it allows either to optimize location of files with respect to available computing capacity, or vice versa to optimize location of computational processing with respect to where the data are available in distributed storage.

As a part of the DiDaS project we have developed a library called *libxio* that allows easy incorporation of IBP into existing UN\*X applications. Resulting from collaboration between DiDaS and DEE projects we have enhanced a video processing tool and video rendering tool with capabilities of using IBP storage. In this report we also present an overview of IBP and *libxio* library from both developer's and end-user's point of view to give other groups and projects an opportunity to extend their tools to adopt IBP capabilities. We conclude with some important issues we have encountered while using these prototypes.

### 3 Network storage stack overview

Network storage stack that uses IBP consists of three layers: the IBP layer, L-Bone and Ex-Node layer, and LoRS layer. An overview of this stack is shown in table 1. An application built using this stack is thus interacting with LoRS layer only.



**Table 1:** Network storage stack overview

#### 3.1 IBP

Similar to IP protocol for network connections, the IBP offers unreliable network block storage for data. The IBP uses soft consistency model with time-limited allocation thus operating in “best effort” mode. Basic atomic unit of IBP is a *byte array* providing abstraction independent on physical device the data are stored on. An *IBP depot* (server) is the basic building block of IBP infrastructure offering disc capacity. The IBP defines three classes of requests described below: allocation, read/write and management.

### 3.1.1 Allocation

When an IBP client needs to store data to some depot it allocates some amount of capacity first. Client may specify several attributes for the allocation:

- **permanent vs. time-limited allocation:** The client can specify whether the storage is required to be persistent or whether the server is free to delete it after specified time period.
- **volatile vs. stable:** The client can specify whether the allocation may be revoked by the server at any time or whether it must be kept for the specified time period.
- **byte-array/pipe/circular-queue:** This attribute is used for specification of access mode. Available options are *append-only array*, *FIFO queue*, or *circular queue*.

IBP depot owner may specify capabilities of the depot so it might offer e.g. only volatile and time-limited allocations of size less than 10GB.

### 3.1.2 Read/Write

The IBP protocol has APIs for reading data from the server(s) and writing data to the server(s). Additionally it offers APIs for requesting data transfer directly between two IBP depots without involving the client that initiates the transfer in actual data transfer. This functionality allows to avoid unnecessary data transfers through network end nodes and to optimize transfers by utilizing fast backbone links that are more likely to be available between two IBP depots than between IBP depot and client.

### 3.1.3 Management

Allocation parameters and attributes may be changed and tuned during existence of allocation: the client can modify time-limit or increase size of allocation. However such request may be refused by IBP servers when no depots with requested parameters are available. The client is also able to receive information on current allocation status and both time and space constraints.

## 3.2 L-Bone and Ex-Node

### 3.2.1 L-Bone

L-Bone layer creates a compounding layer over particular IBP depots that is able to offer locations to clients meeting their requests to maximum extent. L-Bone

servers form a directory layer based on LDAP mechanism. An example might be a depot that both meets client's requirements and is the geographically closest one (or the closest one from network point of view). Each created IBP depot must be registered to L-Bone server to be available for L-Bone server brokering.

### **3.2.2 Ex-Node**

The Ex-Node is a construct remotely resembling UN\*X I-node concept. It comprises particular IBP allocations and forms representation of a whole file. Individual allocations inside one Ex-Node neither need to be of same size nor need to form continuous block. The allocations inside one Ex-node may even be overlapping. The Ex-Node uses XML representation when saved (serialized) to local file. The Ex-Node keeps track of end-to-end services: each block may contain MD5 checksum and may be encrypted using DES, AES, or XOR algorithms. The blocks may also be compressed using UN\*X compress algorithm available in *libz*.

### **3.3 LoRS**

The layer of logistical tools called LoRS is built on the top of L-Bone and Ex-Node layers. It offers interfaces for locating set of IBP depots, APIs for reading, writing and overwriting of the data in the depots with given number of copies and number of concurrent TCP streams to be used. This layer also provides functions for reading and writing Ex-Node from/to the local file using XML representation as mentioned above.

## **4 Prototype implementation of IBP for DiDaS project**

There is an initial experimental implementation of IBP infrastructure for Distributed Data Storage project (DiDaS) designed for preliminary tests and benchmarks of IBP infrastructure as well as for experiments with incorporation of IBP functionality into some pilot applications. Production setup will be optimized based on experiences gained with this prototype.

For preliminary tests and benchmarks we have two servers with internal disc arrays. One server has SCSI disc array and the other one has parallel ATA disc array. We are planing to expand IBP depots to seven servers distributed across academic high speed network CESNET2. We want to use both servers with external RAID arrays and servers with internal RAID arrays to compare both kinds of storage. We will offer about 6TB to 10TB of network storage interconnected with 1Gbps - 2Gbps network links.

In our experimental setup the server with SCSI disc array hosts LDAP server, L-Bone server and IBP depot and also provides web interface to the L-Bone server while other the one hosts IBP depot only.

## 5 End-user how-to for IBP in DiDaS

Up to now we have modified two applications based on requests of end user community. *transcode* program [transcode] has been patched for the DEE project [DEE] to work with IBP in computational cluster infrastructure so that the *transcode* program can load and store files from/to IBP depots. The second application is a media player *Mplayer* [Mplayer] that is able to play the content directly from IBP depot. Besides these two applications there is a number of command line utilities available for the manipulation with files in the IBP depots.

If user wants to access file in the IBP depot then URI in form *lors://host:port/local\_path/file?bs=number&duration=number&copies=number&threads=number&timeout=number* is used. User can also use the short form URI *lors:///local\_path/file*. When user wants to read an IBP file, the *local\_path/file* specifies local file where is stored Ex-Node with XML specification. When writing an IBP file the *local\_path/file* specifies local file where to store Ex-Node with XML specification of IBP depots used. The parameters in the full form of URI mean as follows:

- *host* specification of L-Bone server location
- *port* specification of L-Bone server port (default 6767)
- *bs* specification of block-size for transfer in megabytes (default 10)
- *duration* specification of allocation duration in seconds (default 3600s)
- *copies* specification of number of copies (default 1)
- *threads* specification of number of threads (concurrent TCP streams) (default 1)
- *timeout* specification of timeout in seconds (default 100)

There are environment variables that can be used instead of URI parameters for the LoRS layer.

- *LBONE\_SERVER* specifies L-Bone server location
- *LBONE\_PORT* specifies L-Bone server port (default 6767)

- LORS\_BLOCKSIZE specifies block-size for transfer in megabytes (default 10)
- LORS\_DURATION specifies duration of allocation in seconds (default 3600s)
- LORS\_COPIES specifies number of copies (default 1)
- LORS\_THREADS specifies number of threads (concurrent TCP streams) (default 1)
- LORS\_TIMEOUT specifies timeout in seconds (default 100)

At least LBONE\_SERVER or host must be set.

## 5.1 *transcode*

*transcode* program can be used in almost traditional way. If user wants to store the file to the IBP depot then URI in form *lors:///local\_path/file* is used instead of local file name. The environment variables should be set unless user specifies URI in its full form. The same form of URI can be used for loading the file from the IBP depot. Transcode can read and write ordinary local files as well. The prefix *lors://* is required for accessing IBP, otherwise local file is used.

For example:

```
LBONE_SERVER=udomiel.ics.muni.cz; transcode -i lors:///video.dv.xnd -P1 -N 0x1
-y raw -o lors:///temp1-remux.avi.xnd -E 44100,16,2 -J resample
```

Transcode will use source file identified by serialized description of IBP Ex-Node stored in *video.dv.xnd* file on local disc in the current directory. The L-Bone server location is taken from environment variable (referring to *udomiel.ics.muni.cz*) and resulting serialized Ex-Node XML description will be stored in the current directory as *temp1-remux.avi.xnd*

## 5.2 *Mplayer*

*Mplayer* has been modified to use the same URI semantics as *transcode*. However we have encountered serious problem with reading latency from IBP when reading data and playing video in single thread (what is the way *Mplayer* internally works). More detailed description together with benchmarks can be found in second appendix.

## 5.3 **Command line utilities**

There is number of general purpose command line utilities available, which can be used for manipulation with files stored in the IBP infrastructure. Some of the

parameters can be specified in `/.xndrc` file to avoid repeating specification of the same information when using command-line tools.

### 5.3.1 *lors\_upload*

The *lors\_upload* utility is used for storing files into the IBP depots. Basic usage is as follows:

```
lors_upload -f -H host -n my_file
```

The file *my\_file* will be stored into the IBP depot returned by L-Bone server specified using `host` parameter. The `-f` option says that output Ex-Node file will be called *my\_file.xnd*. The `-n` switch turns off any *end-to-end* services that can otherwise be DES (`-e`), AES (`-a`) or XOR (`-x`) encryption, compression (`-z`), or MD5 check sum (`-l`).

There are other options available:

- `-o` name for the output Ex-Node file
- `-P` L-Bone server port
- `-l` location hint (e.g. `city=Brno`)
- `-d` specification of expiration time
- `-s` volatile storage
- `-h` stable storage
- `-m` maximum number of depots returned by location hint
- `-b` specifies logical block size of input file and chunk size of resulting Ex-Node
- `-c` number of copies
- `-F` number of blocks, which input file should be divided into; an alternative to `-b` switch
- `-t` maximum number of threads used for transfer (meaning number of concurrent TCP streams)
- `-T` timeout

### 5.3.2 *lors\_download*

The *lors\_download* utility is used for downloading files from the IBP depots. Basic usage is as follows:

```
lors_download -o my_file exnode.xnd
```

The data described by *exnode.xnd* will be stored to local file called *my\_file*. If *-o* is omitted, the data is sent to *stdout*.

Other options are available:

- *-r* maximum number of threads used for downloading (meaning again number of concurrent TCP streams)
- *-R* download will resume if working on a partially downloaded file (auto-detection is employed)
- *-q* specification of the number of blocks to pre-buffer before storing into file
- *-C* specifying number of blocks to cache

*-b* and *-t* options have the same meaning as with *lors\_upload*.

### 5.3.3 *lors\_trim*

The *lors\_trim* utility is used to decrease reference count for specified Ex-Node. If reference count is decreased to zero, the allocation is removed from the IBP depot. Basic usage is as follows:

```
lors_trim -o new_exnode.xnd -d exnode.xnd
```

This command means that the original *exnode.xnd* can be deleted while *new\_exnode.xnd* is created. If *-f* switch is used then the original Ex-Node file *exnode.xnd* will be overwritten by updated Ex-Node.

### 5.3.4 *lors\_augment*

The *lors\_augment* utility can be used for increasing reference count for any particular Ex-Node. *-c* switch can be employed to determine how many copies are requested. We can use the same options as with *lors\_upload*. If *-f* switch is used then the original Ex-Node file *exnode.xnd* will be overwritten by updated Ex-Node.

### 5.3.5 *lors\_modify*

The *lors\_modify* utility may be used to remove the specified capabilities from an Ex-Node. Following capabilities can be removed: read (-r), write (-w), or manage (-m). Note: there is no utility that adds capabilities back to Ex-Node.

### 5.3.6 *lors\_ls*

The *lors\_ls* utility lists allocation blocks for given Ex-Node(s). The listing includes offsets, lengths, and locations. Basic usage is

```
lors_ls exnode.xnd.
```

## 6 Developer how-to for IBP in DiDaS

We have developed a abstraction library called *libxio* that is now available for the developers. It provides standard UN\*X I/O interface that allows to access local files as well as with files represented by URI `lors:///local_path/exnode`. The functions *xio\_open*, *xio\_close*, *xio\_read*, *xio\_write*, *xio\_ftruncate*, *xio\_lseek*, *xio\_stat*, *xio\_fstat*, and *xio\_lstat* are available. These functions have the same semantics as standard UN\*X I/O functions except for IBP files can not be opened in O\_RDWR mode at the moment.

```
#include <xio.h>

int xio_open(const char *pathname, int flags);
int xio_close(int fd);
ssize_t xio_read(int fd, void *buf, size_t count);
ssize_t xio_write(int fd, const void *buf, size_t count);
int xio_ftruncate(int fd, off_t length);
off_t xio_lseek(int fildes, off_t offset, int whence);
int xio_stat(const char *file_name, struct stat *buf);
int xio_fstat(int fildes, struct stat *buf);
int xio_lstat(const char *file_name, struct stat *buf);
```

For accessing IBP infrastructure the *libxio* library uses *lors* library that provides interface described in subsequent subsections.

## 6.1 Creating a new file

```
lorsGetDepotPool(LorsDepotPool *dp, char *host, int port,  
                IBP_depot *dpt, int maxdepot, char *location_hint,  
                unsigned long storage_size, int storage_type,  
                time_t duration, int max_threads, int timeout, int opts)
```

This function returns a list of appropriate IBP depots (dp) meeting given criteria: *location\_hint* [city=Brno], *storage\_size* (in megabytes), *storage\_type* [IBP\_SOFT|IBP\_HARD] (volatility), *duration* (expiry time in seconds).

```
lorsExnodeCreate(LorsExnode **ex)
```

This function returns a newly created Ex-Node.

## 6.2 Opening a file for reading

```
lorsFileDeserialize(LorsExnode **ex, char *filename, char *schema)
```

This function opens an Ex-Node XML file called *filename* and creates a corresponding Ex-Node structure in memory. The *schema* parameter is currently unused.

```
lorsUpdateDepotPool(LorsExnode *ex, LorsDepotPool **dp,  
                   char *lbone_server, int lbone_server_port,  
                   char *location_hint, int nthreads, int timeout, int opts)
```

This function returns a list of appropriate IBP depots for given Ex-Node. L-Bone server may have NULL value in which case no location hinting will be used and IBP depots are selected randomly.

## 6.3 Writing to a file in IBP

```
lorsQuery(LorsExnode *ex, LorsSet **set, longlong offset,  
          longlong size, int opt)
```

This function returns a set (*set*) of allocations containing data block beginning at *offset* and having *size* length. LORS\_QUERY\_REMOVE for *opt* parameter is used.

```
jrb_empty(set->mapping_map)
```

This function returns *true* if and only if the *set* does not contain any data block. This means that data will be written to empty (unallocated) block.

### 6.3.1 Writing to an empty block

```
lorsSetInit(LorsSet **set, ulong_t data_blocksize, int copies, int opts)
```

This function returns a newly created empty set of blocks.

```
lorsSetStore(LorsSet *set, LorsDepotPool *dp, char *buffer,  
            longlong offset, longlong length, LorsConditionStruct *lc,  
            int nthreads, int timeout, int opts)
```

This function stores *set* into the IBP depots. The parameter *LorsConditionStruct* is always NULL at this time. It is recommended to use value `LORS_RETRY_UNTIL_TIMEOUT` for the *opts* parameter.

### 6.3.2 Overwriting an existing block

First we need to assign the number of copies and block size to the particular set:

```
set->copies = number  
set->data_blocksize = size
```

```
lorsSetUpdate(LorsSet *set, LorsDepotPool *dp, char *buffer,  
            longlong offset, longlong length, int nthreads,  
            int timeout, int opts)
```

This function overwrites existing blocks with new data. All parameters have already been described above. More precisely this function splits the existing allocation into pieces and decreases reference count of overwritten blocks.

```
lorsAppendSet(LorsExnode *ex, LorsSet *set)
```

This function adds *set* to Ex-Node that describes file we work with.

```
lorsSetFree(LorsSet *set, int opt)
```

This function frees *set* that is no longer necessary (e.g. when it has already been added to Ex-Node).

```
lorsFileSerialize(LorsExnode *ex, char *filename,  
                int read_only, int opts)
```

This function saves Ex-Node into the local file. Setting parameter *read\_only* to 1 means that we want to save read-only capability while write and manage capabilities will not be saved. The parameter *opts* must be always set to 0.

## 6.4 Reading from a file in IBP

We select a set from Ex-Node using `lorsQuery()` function. This set covers requested data blocks.

```
lorsSetLoad(LorsSet *set, char *buffer, longlong offset,  
           long length, ulong_t block_size, LorsConditionStruct *lc,  
           int nthreads, int timeout, int opts)
```

This function reads requested data up to *length*. On success, the number of read bytes is returned. Zero indicates end of file while negative value means error.

## 6.5 Closing a file

Ex-Node data should be saved using `lorsFileSerialize()` function before we close a file. We can free structures used by calling `lorsExnodeFree(LorsExnode *ex)` and `lorsFreeDepotPool(LorsDepotPool *dp)` functions.

## 6.6 Truncating a file

Again we start off by creating a set using `lorsQuery()` function. This set contains truncated blocks.

```
lorsSetTrim(LorsSet *set, longlong offset, longlong length,  
           int nthreads, int timeout, int opts)
```

This function decreases reference count for blocks beginning at *offset* having length of *length*. The parameter *opts* has to have the value `LORS_TRIM_ALL`.

## 6.7 Getting size of a file

The actual size of data that described by Ex-Node is kept in `LorsExnode->logical_length` variable.

# 7 Conclusions

Our first version of *libxio* stores data to the IBP depot with each single *xio\_write* call. This results in the important latency issue that does not harm a lot while video transcoding but it also results in very large Ex-Node XML description. Each call adds approximately 400B to Ex-Node. We are using a write buffer of

10MB size at the moment. The buffer reduces latency and rapidly reduces the Ex-Node XML file at the cost of unnecessary *memcpy* call. However we believe that extra hit to extremely fast memory is a lot better than extra hit to relatively slower network.

Our preliminary tests have shown IBP storage as very suitable for DEE project. We don't not use any location hinting so far but as we are planning to spread IBP depots over Czech academic network, the location hinting and some form of integration with scheduling system will be necessary. As integration with scheduling system currently in use - PBS [PBS] - seems to be rather tough task and new generation of scheduling systems with required capabilities seems to come rather shortly (e.g. from DataGrid Project [EDG]), we plan to do the integration of PBS and IBP on application layer so that application will give hints to PBS where the jobs should be scheduled to.

The experiments with *Mplayer* have been somewhat worse since the read latency has very negative impact on *Mplayer*. We can use read buffer as well but this only helps on fast network with low latency. The only chance is to start another thread that will try to read as big blocks as possible from IBP depots. The authors of IBP technology promise a new generation of IBP that will meet real-time requirements in better way. We will go on with modifying our library to use read thread until the new version of IBP is developed.

## **8 Appendix A Software and Hardware RAID arrays benchmarks**

Internal discs array was equipped with single Intel Pentium 4 Xeon 2.4GHz processor and 1GB RAM. On this computer was installed operating system Linux with vanilla kernel 2.4.22 with patch for XFS filesystem. Disc arrays were benchmarked with [IOZone] program that does sequential reading and sequential writing test to XFS partition.

We used parallel ATA Western Digital 250GB 7200RPM discs with 8MB cache, serial ATA Western Digital 250GB 7200RPM discs with 8MB cache and SCSI Seagate Cheetah 73GB 10000RPM discs.

We did performance test on single disc, four discs in software RAID 0 array, four discs in software RAID 5 array, four discs in hardware RAID 0 array and four discs in hardware RAID 5 array. In the case of hardware array we increased shared memory size to 1GB, *vm.min-readahead* to 128, and *vm.max-readahead* to 256 in kernel.

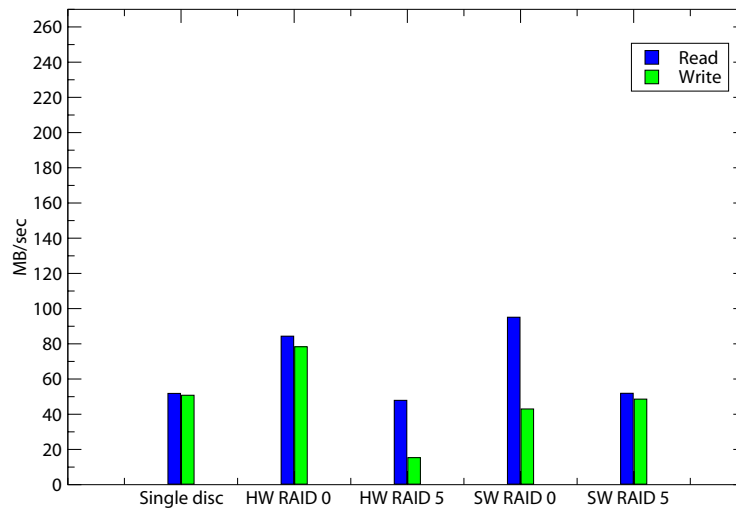
## 8.1 IDE RAID arrays

In this case, each disc was switched to master mode. The Adaptec AAR 2400A card has four parallel ATA interfaces. The 3ware Escalade 8506-8 card has eight serial ATA interfaces. While benchmarking these arrays we have found that Linux driver for Adaptec AAR 2400A is unstable as of kernel version 2.4.22 and should not be used for any production system. Following setup was used for the IDE RAID arrays benchmarks:

- single disc attached to internal Intel ICH3 controller
- four discs attached to PCI32 Adaptec AAR 2400A card
- four discs attached to PCI64 3ware Escalade 8506-8 card with SATA interface

PATA	Single disc	HW RAID 0	HW RAID 5	SW RAID 0	SW RAID 5
read	51.823MB/sec	84.359MB/sec	47.902MB/sec	95.111MB/sec	51.902MB/sec
write	50.785MB/sec	78.330MB/sec	15.349MB/sec	42.989MB/sec	48.616MB/sec

**Table 2:** Parallel ATA disc array benchmark



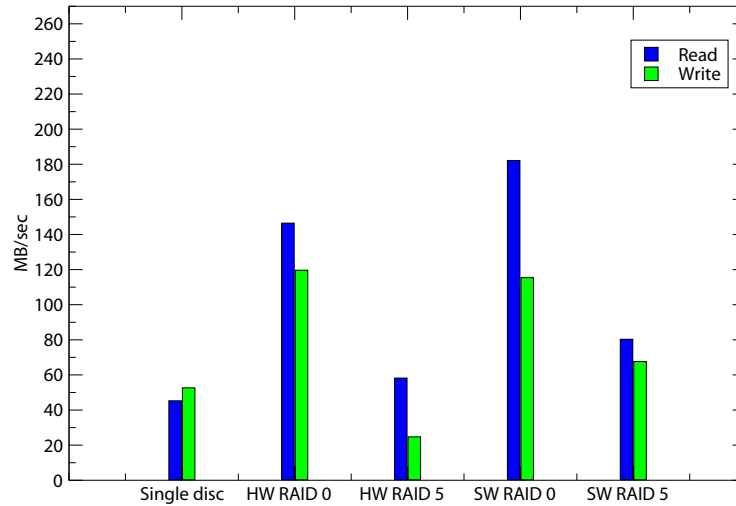
**Figure 1:** Parallel ATA disc array benchmark.

## 8.2 SCSI RAID arrays

In this case, four discs were attached to one bus. Adaptec AIC card used poses only one internal SCSI interface. Following setup was used for the SCSI RAID arrays benchmarks:

SATA	Single disc	HW RAID 0	HW RAID 5	SW RAID 0	SW RAID 5
read	45.273MB/sec	146.437MB/sec	58.186MB/sec	182.129MB/sec	80.294MB/sec
write	52.663MB/sec	119.643MB/sec	24.719MB/sec	115.402MB/sec	67.601MB/sec

**Table 3:** Serial ATA disc array benchmark



**Figure 2:** Serial ATA disc array benchmark.

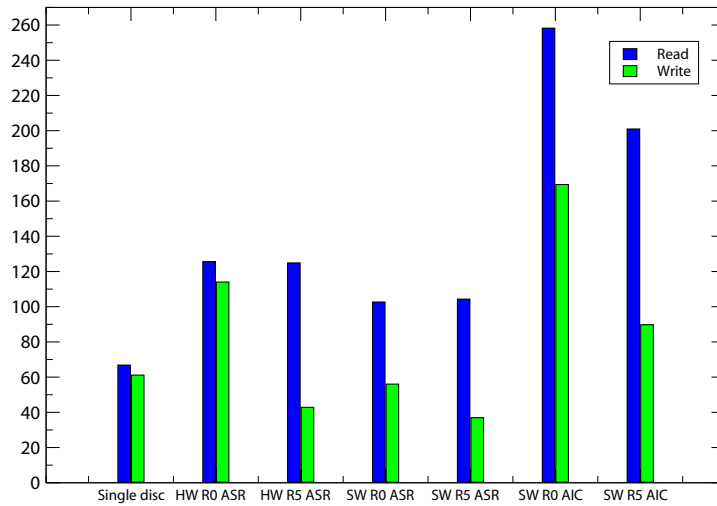
- single disc attached to PCI64 Adaptec AIC 7901A Ultra 320 Single channel card
- four discs attached to PCI64 Adaptec AIC 7901A Ultra 320 Single channel card
- four discs attached to PCI64 Adaptec ASR 2200S Dual Channel card

SCSI	Single Disc	HW RAID 0 ASR	HW RAID 5 ASR	SW RAID 0 ASR	SW RAID 5 ASR
read	66.846MB/sec	125.564MB/sec	124.893MB/sec	102.670MB/sec	104.319MB/sec
write	61.191MB/sec	114.030MB/sec	42.864MB/sec	56.046MB/sec	36.956MB/sec

**Table 4:** SCSI disc array benchmark

## 9 Appendix B IBP read latency benchmarks

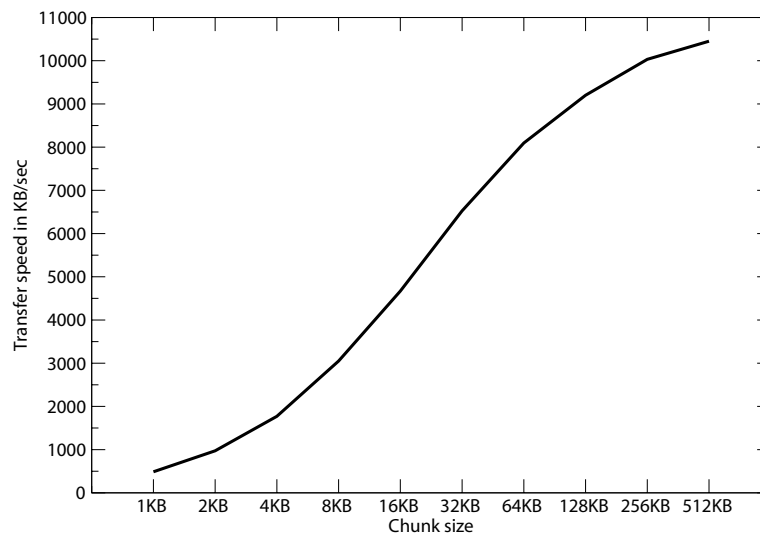
We have done several experiments with playing content directly from IBP depots. LoCI laboratory [LoCI] uses command line utilities together with Mplayer for playing content from IBP depots. For example:



**Figure 3:** SCSI disc array benchmark.

`lors_download my_file.xnd | mplayer -`

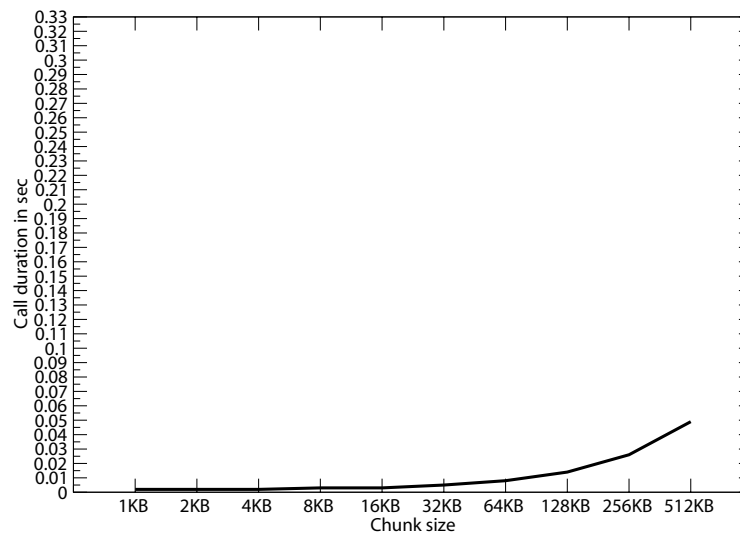
We patched *Mplayer* so that it can handle Ex-Nodes on its own. While *Mplayer* runs in single thread mode (and author of *Mplayer* insists on the fact that this is the way to go) we must look after read latency from IBP depot. We have set up an experiment to show how the downloading speed and latency depends on the chunk size. We have done this experiment both on local network and the network whose connection is a little bit slower. We have used one-stream transfers only as multiple streams require multiple threads.



**Figure 4:** LoRS speed on local 100Mbps (FE) network.

Chunk size	Downloading speed	Single read call duration
1B	0.16KB/sec	0.006sec
2B	1.09KB/sec	0.002sec
4B	2.22KB/sec	0.002sec
8B	4.44KB/sec	0.002sec
16B	8.85KB/sec	0.002sec
32B	17.73KB/sec	0.002sec
64B	35.11KB/sec	0.002sec
128B	70.30KB/sec	0.002sec
256B	139.35KB/sec	0.002sec
512B	270.71KB/sec	0.002sec
1KB	486.38KB/sec	0.002sec
2KB	973.24KB/sec	0.002sec
4KB	1770.69KB/sec	0.002sec
8KB	3049.94KB/sec	0.003sec
16KB	4668.81KB/sec	0.003sec
32KB	6526.62KB/sec	0.005sec
64KB	8096.14KB/sec	0.008sec
128KB	9201.35KB/sec	0.014sec
256KB	10032.53KB/sec	0.026sec
521KB	10451.11KB/sec	0.049sec

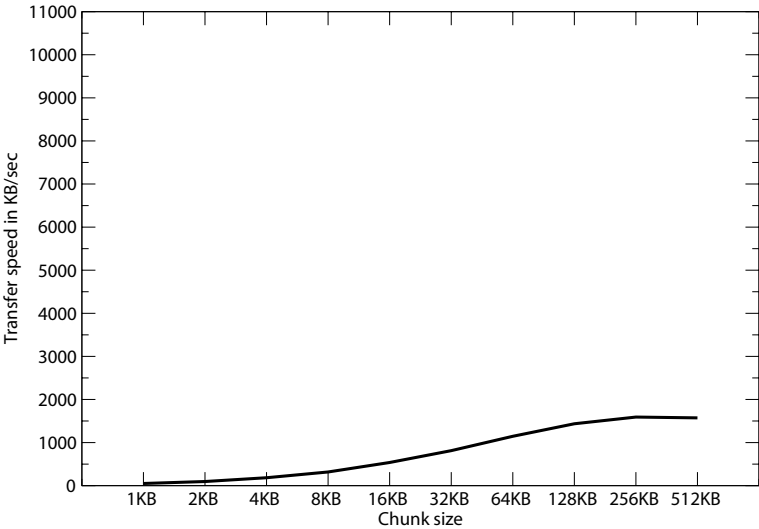
**Table 5:** LoRS speeds on local 100Mbps (FE) network



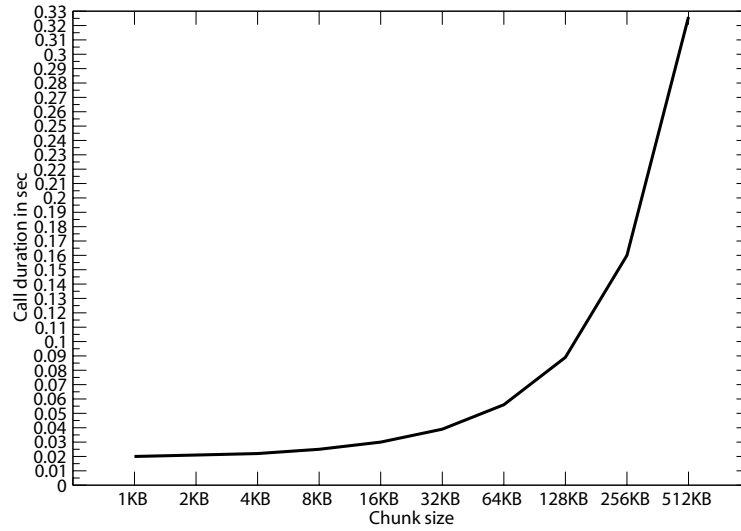
**Figure 5:** LoRS latency on local 100Mbps (FE) network.

Chunk size	Downloading speed	Single read call duration
1B	0.05KB/sec	0.018sec
2B	0.11KB/sec	0.019sec
4B	0.21KB/sec	0.018sec
8B	0.41KB/sec	0.019sec
16B	0.80KB/sec	0.019sec
32B	1.70KB/sec	0.018sec
64B	3.35KB/sec	0.019sec
128B	6.65KB/sec	0.019sec
256B	12.78KB/sec	0.020sec
512B	26.50KB/sec	0.019sec
1KB	49.75KB/sec	0.020sec
2KB	95.08KB/sec	0.021sec
4KB	183.11KB/sec	0.022sec
8KB	317.41KB/sec	0.025sec
16KB	538.05KB/sec	0.030sec
32KB	812.89KB/sec	0.039sec
64KB	1145.00KB/sec	0.056sec
128KB	1436.87KB/sec	0.089sec
256KB	1591.58KB/sec	0.160sec
521KB	1572.75KB/sec	0.326sec

**Table 6:** LoRS speeds on foreigner 100Mbps (FE) network

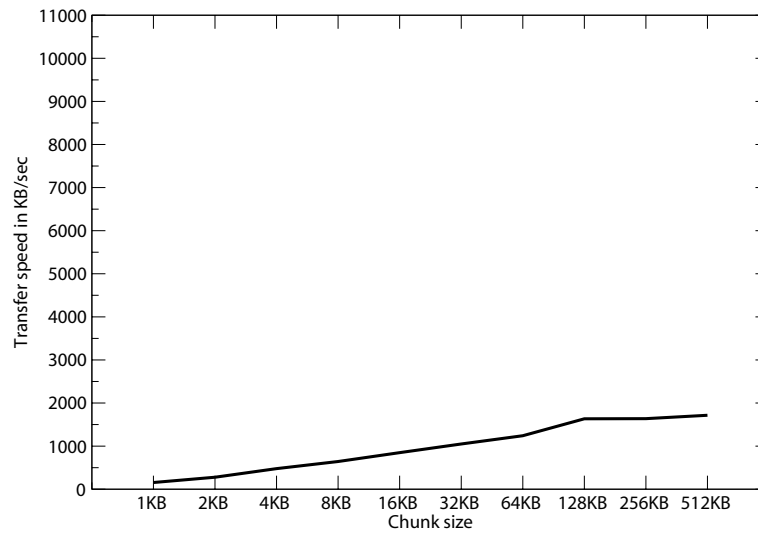


**Figure 6:** LoRS speeds on foreigner 100Mbps (FE) network.



**Figure 7:** LoRS latency on foreigner 100Mbps (FE) network.

First we thought that LoRS layer creates that latency thus we did the same experiment with IBP layer directly.

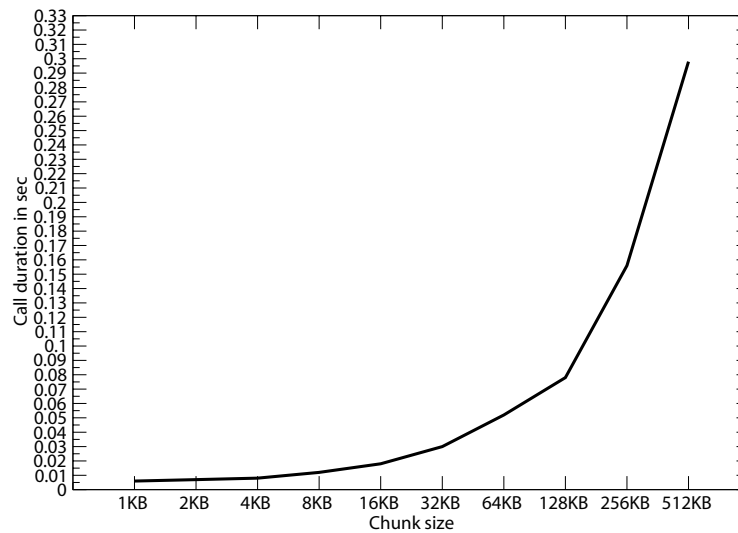


**Figure 8:** IBP speeds on foreigner 100Mbps (FE) network.

We compared IBP transfer speed with single stream ftp transfer speed using bbftp [bbftp] on the slower network.

Chunk size	Downloading speed	Single read call duration
1B	0.16KB/sec	0.006sec
2B	0.29KB/sec	0.006sec
4B	0.62KB/sec	0.006sec
8B	1.22KB/sec	0.006sec
16B	2.43KB/sec	0.006sec
32B	5.04KB/sec	0.006sec
64B	9.82KB/sec	0.006sec
128B	19.32KB/sec	0.006sec
256B	38.15KB/sec	0.006sec
512B	69.12KB/sec	0.007sec
1KB	155.51KB/sec	0.006sec
2KB	278.38KB/sec	0.007sec
4KB	478.45KB/sec	0.008sec
8KB	642.54KB/sec	0.012sec
16KB	848.10KB/sec	0.018sec
32KB	1049.23B/sec	0.030sec
64KB	1241.19KB/sec	0.052sec
128KB	1633.99KB/sec	0.078sec
256KB	1637.22KB/sec	0.156sec
521KB	1715.69KB/sec	0.298sec

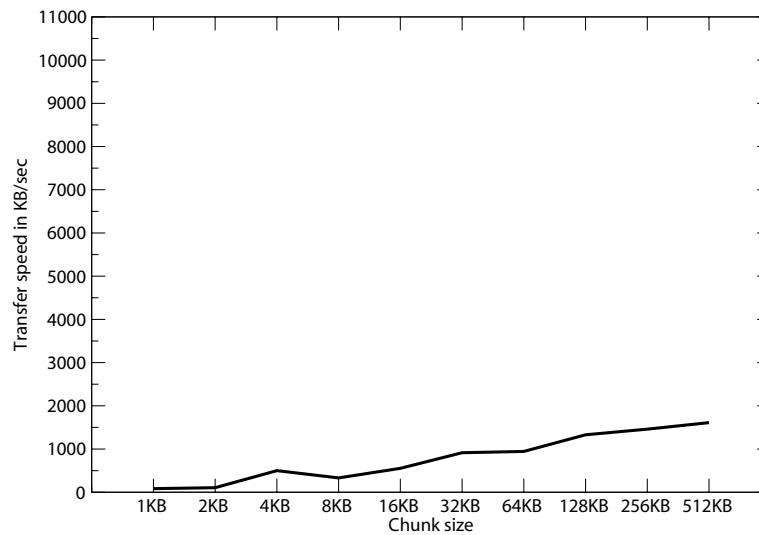
**Table 7:** IBP speeds on foreigner 100Mbps (FE) network



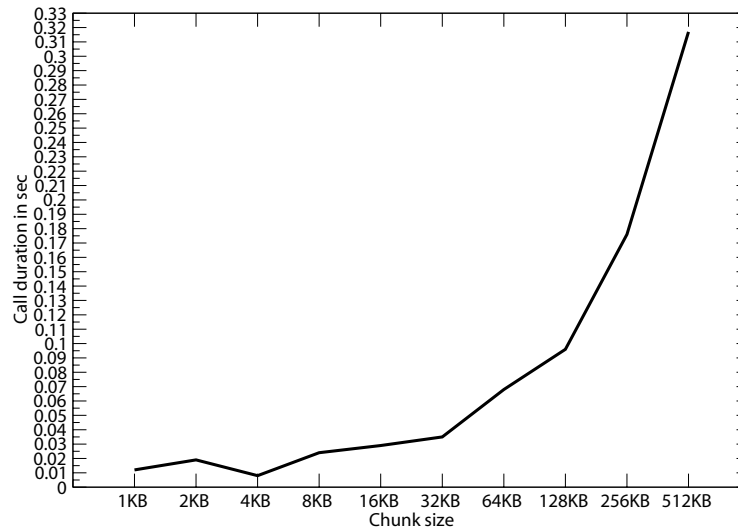
**Figure 9:** IBP latency on foreigner 100Mbps (FE) network.

Chunk size	Downloading speed	Single read call duration
1B	0KB/sec	0.005sec
2B	0KB/sec	0.008sec
4B	0KB/sec	0.008sec
8B	1KB/sec	0.011sec
16B	0KB/sec	0.047sec
32B	3KB/sec	0.008sec
64B	1KB/sec	0.048sec
128B	3KB/sec	0.037sec
256B	28KB/sec	0.009sec
512B	40KB/sec	0.012sec
1KB	84KB/sec	0.012sec
2KB	105KB/sec	0.019sec
4KB	500KB/sec	0.008sec
8KB	331KB/sec	0.024sec
16KB	553KB/sec	0.029sec
32KB	915B/sec	0.035sec
64KB	945KB/sec	0.068sec
128KB	1330KB/sec	0.096sec
256KB	1460KB/sec	0.176sec
521KB	1610KB/sec	0.317sec

**Table 8:** bbftp speeds on foreigner 100Mbps (FE) network



**Figure 10:** bbftp speeds on foreigner 100Mbps (FE) network.



**Figure 11:** bbftp latency on foreigner 100Mbps (FE) network.

## References

- [Meta] MetaCenter Project, <http://meta.cesnet.cz/>
- [SIG02] Beck M., Moore T., Planck J. S. “An End-to-End Approach to Globally Scalable Network Storage”. SIGCOMM 2002.  
<http://loci.cs.utk.edu/ibp/files/pdf/SIGCOMM02p1783-beck.pdf>
- [IBP] Internet Backplane Protocol, <http://loci.cs.utk.edu/ibp/>
- [LoCI] Logistical Computing and Internetworking Lab, <http://loci.cs.utk.edu>
- [transcode] Linux Video Stream Processing Tool.  
<http://www.theorie.physik.uni-goettingen.de/%7Eostreich/transcode/>  
and  
<http://zebra.fh-weingarten.de/%7Etranscode/>
- [DEE] Distributed Encoding Environment project,  
<http://sitola.fi.muni.cz/projekty/strizna/strizna.html>.
- [Mplayer] Linux Media Player Tool. <http://www.mplayerhq.hu>
- [PBS] Portable Batch System: OpenPBS (<http://www.openpbs.org/>) and  
PBSPro (<http://www.pbspro.com/>)
- [EDG] EU DataGrid Project (<http://www.eu-datagrid.org/>)
- [IOZone] Disc I/O Benchmark Tool. <http://www.iozone.org>
- [bbftp] bbFTP – Large files transfer protocol. <http://doc.in2p3.fr/bbftp/>