Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

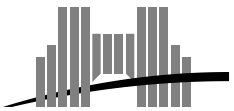
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Improving Cluster IO Performance with Remote Efficient Access to Distant Device

Olivier Cozette
Cyril Randriamaro
Gil Utard

March 2002

Research Report N° 2002-11



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

INRIA

Improving Cluster IO Performance with Remote Efficient Access to Distant Device

Olivier Cozette
Cyril Randriamaro
Gil Utard

March 2002

Abstract

Grand challenge applications often need to process large amounts of data so high performance IO systems are needed. Cluster computing is a good approach to build cost effective IO intensive platform: sort benchmark records (MinuteSort, Datamation) are held by cluster architecture! New advances in IO technologies (disk, controller and network) let expected significant performance improvements for data intensive applications on cluster architectures. The counterpart of this evolution is that more stress is put on the different shared buses of each node. In this paper we investigate a solution to reduce the bus pressures to improve IO performance we called READ^2 (Remote Efficient Access to Distant Device). With READ^2, a cluster node is able to directly access to a remote disk without interfering with the remote processor and remote memory. With READ^2, a cluster can be considered as a *shared disk* architecture instead of a *shared nothing* one, and inherits works from the SAN community. This papers presents what are the architectural benefits of READ^2, i.e. a better use of IO and memory buses which may improve performance of streaming application.

Keywords: High Performance IO, Cluster, Streaming Applications.

Résumé

L'utilisation des grappes pour des applications traitant de grandes masses de données est une approche séduisante. Par exemple, les meilleurs performances pour les tests de tri (Datamation, MinuteSort) sont aujourd'hui obtenues avec ce type d'architecture. Les dernières évolutions des périphériques d'entrées/sorties (disques, contrôleur, réseau) laisse espérer des améliorations sensibles des performances. La contrepartie de cette évolution est qu'une pression de plus en plus forte est mise sur le maillon faible des noeuds : les bus. Dans cet article nous proposons une technique pour réduire cette pression et repousser les limites de performances que nous appelons READ^2 : nous employons les nouvelles capacités d'accès des cartes réseaux à l'espace d'adressage IO des bus d'entrées/sorties pour piloter directement les disques distants sans interférer avec le nœud hôte. Une grappe peut alors être considérée comme une architecture où les disques sont entièrement partagés par l'ensemble des nœuds. Ce papier présente une étude préliminaire sur les principaux bénéfices que permet une telle approche : une meilleur utilisation des ressources d'entrées/sorties et une amélioration des performances globales des applications traitant de grand flot de données.

Mots-clés: Entrées/Sorties Hautes Performances, Grappes, Applications Flot de Données.

Improving Cluster IO Performance with Remote Efficient Access to Distant Device*

Olivier Cozette, Cyril Randriamaro
PaLADIN – LaRIA
Université de Picardie Jules Verne
5, rue du Moulin Neuf
80000 Amiens, France
cozette,crandria@laria.u-picardie.fr

Gil Utard
ReMaP – LIP
École Normale Supérieure de Lyon
69364 Lyon Cedex 07, France
Gil.Utard@ens-lyon.fr

March 2002

1 Introduction

Grand challenge applications often need to process large amounts of data. Data mining, signal processing and video on demand (VOD) are applications limited by IO operations, a very large data set needs to be handled. For example, San Francisco museum picture collection was digitalized and generated 3.2 Teraoctets. So is the European particle accelerator (LHC of the CERN [5]): the amount of data to process reaches several Petabytes per year. To deal with such applications, cluster architecture based on commodity components can be designed: disk drives are aggregated to provide a large parallel file system. Hence, San Francisco museum [12] uses 20 PCs with 368 disks. Some sort benchmark records demonstrated that cluster architectures are able to compete with specialized one [1].

Commodity clusters are considered as *shared nothing architecture*. A consequence is that to share data, each node must acts as a server to other nodes. So, for distributed parallel file system like PVFS [8] or PPFS [6], each node is burden to serve local data requested by another node. Good performance may be obtained for homogeneous collective parallel IO by an adequate placement and redistribution schemes [7], but for general accesses, the overhead is non negligible. The overhead is two folds: the first is on the operating system running on each node, the second on the hardware architecture. In this paper, we focus our study on the hardware architecture point of view.

Thanks to the new IO technology advances, today some disk drives are able to achieve up to 90MByte/s of sustained bandwidth, disk controllers can achieves several hundred MByte/s of bandwidth, and network cards can achieve several Gigabit/s bandwidth! By adding several IO components in each node, it is possible to get plenty of IO bandwidth for data intensive applications. However, this increasing IO bandwidth put pressure on the IO and memory buses of each node, which cannot be scaled.

The Use of Network Attached Devices is an alternative to provide large and scalable parallel file systems where cluster node accesses directly shared data without interfering with other node. Several works, like GFS [10] or NASD [4] proven the effectiveness of such a technology. Unfortunately, this approach implies the deployment of an expensive network infrastructure like Fibre Channel.

We investigate how to reduce pressures on the different buses by the use of a new method to access remote data called READ^2 (Remote Efficient Access to Distant Device). In READ^2, we exploit the capability

*This work is supported by a grant of the "Pôle de Modélisation de la Région Picardie" and by the INRIA ReMaP project.

of modern network interface cards to directly access remote IO device. For instance, in [13] authors combine two cooperative Myrinet cards on the same IO bus for efficient IP forwarding: the data throw directly from one Myrinet card to the second one and the processor is not involved in the data-path, increasing the peak bandwidth of the forwarding scheme. In READ^2, we propose to use the IO access capability of modern network interface cards which allows nodes to directly access to remote disk drive. With the READ^2 techniques, cluster is considered as a *shared disks architecture*, where all disks may be accessed by any node like Network Attached Storage Device.

In this paper we study what are the bus usage benefit of READ^2 for streaming applications. In a first part we present the architectural cluster model we consider and introduce READ^2. In a second part we propose a model for parallel streaming applications, where a continuous stream of data are processed. This model is illustrated for two example applications. Then, we predict what are the benefit of a READ^2 approach in general and for the application examples. Finally we present an experimental validation of our model.

2 Architecture Model and READ^2

In this section we present a cluster architectural model and introduce the READ^2 approach.

2.1 Architectural model

A cluster may considered as an interconnection of different buses. A node is composed of:

A Input/ouput bus: (IO bus) to connect network card, disk and IO bridge; usually a PCI bus.

A Memory bus: to connect memory to IO bridge.

A Processor bus: to connect processor to IO bridge.

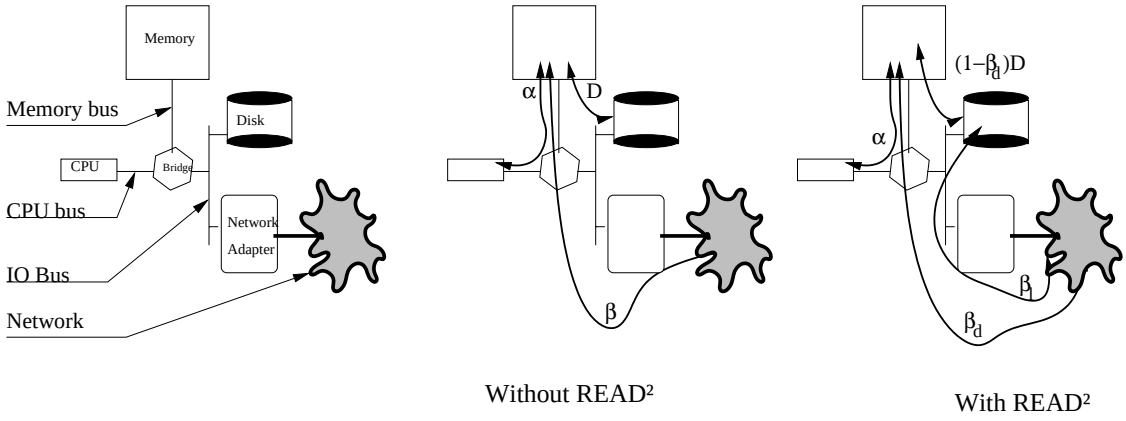


Figure 1: Architecture and variable description

Interconnection are sum up on Figure 1. These buses are characterized by the following constant.

M_d	maximum disk throughput
M_{io}	maximum IO bus throughput
M_m	maximum memory bus throughput
M_c	maximum instruction processing

In a cluster nodes are interconnected by network interface card plugged on the IO bus: the network glues IO busses to build a parallel machine. From a logical point of view, the network may be considered as an another bus level. The network is characterized by the constant M_n which is the maximum network throughput.

2.2 READ^2: Remote Efficient Access to Distant Device

Cluster are usually considered as a *shared nothing* architecture: each node is independent and collaborates with other nodes by message exchanges, so the processor, memory and disk of each node are only accessed by the local system. In fact it is a high network level (session level) point of view. If we consider a lower level network point of view (transport level) some components of each node (memory, disk) may be shared by all nodes. For instance the SCI network technology [11] or some Virtual Shared Memory implementations based on remote DMA, allows nodes to share memory.

Usually, to share disk data like in parallel file system (eg PVFS), each node must act as a data server for other node: it is a *peer-to-peer server* approach (P2PS). A consequence is that when a node access data on a remote node, its memory bus and IO bus are involved two times in the data path. Patterson and *al* in [2] demonstrated that for streaming applications, cluster architectures are penalized by this overuse of the different buses, specially the IO bus, which is the bottleneck of current architectures. Moreover, more stress is put on the memory bus by the local file system witch may involves buffer copying policy. Some works like DAFS [3] try to remove this copying penalty by remote memory access, but always involves two traversals of buses in the data path.

In this paper we propose a new technique to allows nodes to share disks: each nodes is able to directly access and control any remote disk. A consequence is that the memory is not involved in the data path, and the IO bus is involved only one time. We investigate what is the benefit of such an approach for streaming applications.

3 Streaming application model

We are considering parallel applications processing a contiguous stream of disk data and where the different part of the process (read, compute, communicate and write) are pipelined to achieve maximum overlapping. The local disk stream throughput is represented by the parameter $D = D_r + D_w$, where D_r is the throughput of the read data and D_w is the throughput of the write data.

The application is characterized by the following parameter (see Figure 1):

α: This parameter corresponds to the average cpu memory accesses for each datum of the local disk stream.

β: This parameter corresponds to the average data communicated to/from other nodes for each datum of the local disk stream. We assume that this communication is equidistributed between nodes. This measure is divided in two parts:

- **β_d:** This parameter represents the average communicated data of the local disk stream which is not involved or produced by the local computation.
- **β_l:** This parameter represents the average communicated data which are involved or produced by the local computation.

We have $\beta = \beta_d + \beta_l$.

γ: This parameter corresponds to the average cpu instruction for each datum of the local stream.

This parameters are illustrated for two applications: scan and transpose.

3.1 Scan

In this application, each node independently read its local disk and select some record which are wrote back to disk in another file. We assume that each record is composed of a key plus some data and the ratio between the key size and the record size is denoted by r. The fraction of records selected is denoted by f. So, $D_w = f \times D_r$ and $D = (1 + f) \times D_r$.

Because, there is no communication in this application, we have:

$$\beta = 0$$

For each record, the key is accessed by cpu to verify if the record is selected or not. So the average cpu memory accesses is denoted by:

$$\alpha = \frac{r}{1+f}$$

We assume that the scan select record where key is in a fixed range. So there are two comparison instructions for each record, the average cpu usage is:

$$\gamma = \frac{2r}{1+f}$$

3.2 Transposition

Let n be the number of nodes in a cluster. $M \times M$ matrices are distributed all over the nodes and stored by block of size $B \times B$, such as $nB = M$ (the memory of one node holds one block). Each block line is assigned to one node and stored on its local disks (cf fig. 2). When a matrix is transposed, the i^{th} block of the j^{th} processor becomes the j^{th} block of the i^{th} processor. Each node must read $n-1$ blocks from the

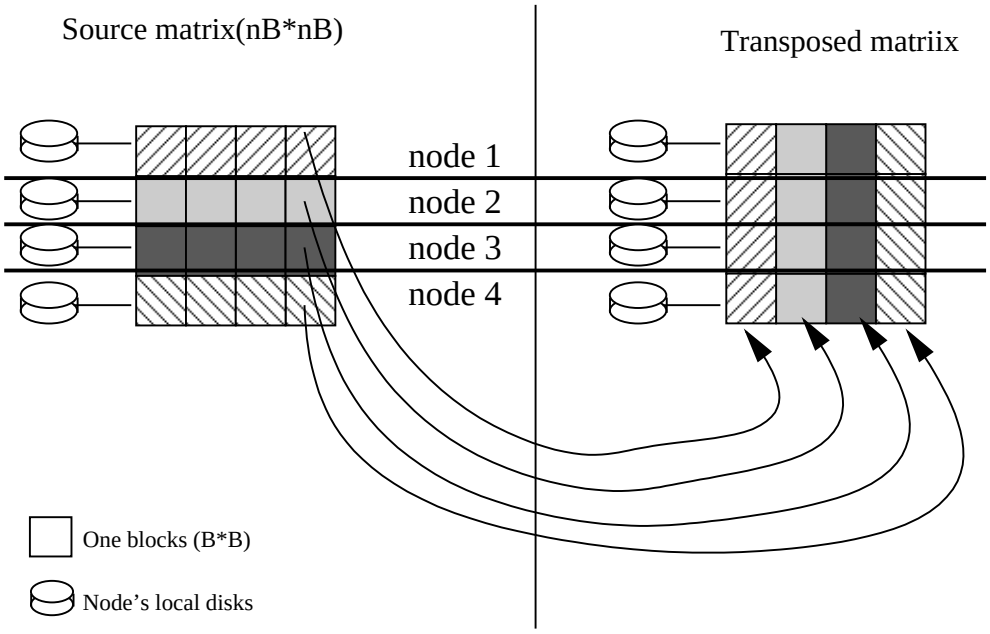


Figure 2: Parallel block transposition

other nodes and send them $n-1$ of its own blocks. One block per node remains local.

Let estimate the different parameter. We have $D_r = D_w$, the same amount of data is read and wrote, then $D = 2D_r = 2D_w$. For communication, each node send $\frac{n-1}{n}$ of the local input disk stream ($D_r = D/2$) and receive the same amount from other nodes for transposition. The transposed data are wrote on the local disk ($D_w = D/2$).

$$\beta_l = \frac{n-1}{n}/2 \qquad \beta_d = \frac{n-1}{n}/2 \qquad \beta = \frac{n-1}{n}$$

Each row must be transposed. So there is two cpu memory access for each datum (one load for the read stream and one store for the write stream), so we have

$$\alpha = \gamma = 1$$

4 Prediction of the bus usage

From the previous model, we now derive the different bus throughput usage of a streaming parallel application. Let B_{io} be the IO bus usage, B_m the memory bus usage, B_c the cpu usage and B_n the network usage. Let n the number of node. For a given streaming application characterized by the triplet (α, β, γ), we determine what is the used throughput of each bus for a given streaming throughput $D = D_r + D_w$. We determine the different bus usage when the application is using a classical *peer-to-peer* server (P2PS) approach for remote data access and when READ^2 is used. The table 1 summarize the different bus usage

bus	P2PS	READ^2
B_{io}	$(1 + \beta_l + \beta_d)D$	$(1 + \beta_l)D$
B_m	$(1 + \alpha + \beta_l + \beta_d)D$	$(1 + \alpha + \beta_l - \beta_d)D$
B_c	γD	γD
B_n	$n \times \beta D$	$n \times \beta D$

Table 1: The different bus usages

This table clearly shows where is the benefit of Direct Remote Disk Access.

- The IO bus usage is reduced: the data of local disk stream not involved or produced by the local computation cross the IO bus one time only, the benefit is equal to $\beta_d D$.
- There are less contention on the memory bus: the data of local disk stream not involved or produced by the local computation don't cross the memory bus, the benefit is equal to $2\beta_d D$.

So, this two main gains allows us to increase the maximum bandwidth of disk stream when the IO bus is the bottleneck. At the same time we get more memory traffic for local computation. Note that for applications where $\beta_d = 0$ (eg scan) there is no gain.

Bus usage prediction for transposition

We estimate the benefit of READ^2 for the transposition. There are two benefits: The first one is an increase of the peak bandwidth achievable (D), the second one is a freeing of the memory bandwidth.

Peak bandwidth of the transposition

Peak bandwidth is reached when one resource (cpu, bus, network) reaches its limit, that is $M_m = B_m$ or $M_{io} = B_{io}$ or $M_c = B_c$:

- Without READ^2 ,
$$D = min\left(\frac{M_m}{1 + \alpha + \beta_l + \beta_d}, \frac{M_{io}}{1 + \beta_l + \beta_d}, \frac{M_c}{\gamma}\right)$$
- With READ^2 ,
$$D = min\left(\frac{M_m}{1 + \alpha + \beta_l - \beta_d}, \frac{M_{io}}{1 + \beta_l}, \frac{M_c}{\gamma}\right)$$

Hence, transpose maximum disk throughput is:

- Without READ^2,
$$D = min\left(\frac{M_m}{2 + \frac{n-1}{n}}, \frac{M_{io}}{1 + \frac{n-1}{n}}, M_c\right)$$
- With READ^2,
$$D = min\left(\frac{M_m}{2}, \frac{M_{io}}{1 + \frac{n-1}{2n}}, M_c\right)$$

The majority of architectures assume $M_{io} < 2M_m$ and $M_c >> M_m$:

- Without READ^2 ,
$$D = \frac{M_{io}}{1 + \frac{n-1}{n}}$$
- With READ^2 ,
$$D = \frac{M_{io}}{1 + \frac{n-1}{2n}}$$

Consider common bus bandwidth, ie $M_{io} = 133Mo/s$, $M_m = 800Mo/s$ and $M_c = 2000$ Mips, peak bandwidths achievable are:

- Without READ^2 , $D = 66MB/s$ (achievable with two 40 MB/s disks), $B_m = 190MB/s$ and network rate is $66MB/s$ per link (current Myrinet bandwidth is 200MB/s per link).
- With READ^2 $D = 86MB/s$ (achievable with three 40MB/s disk), $B_m = 172MB/s$ and network rate is $86Mb/s$ per link.

So direct remote IO improves by 25% the transposition rate.

Memory bus contention

The memory bus freeing by READ^2 is determined by β_d. As in the previous section, we assume the IO bus is saturated. With READ^2, although the peak bandwidth of the transposition is increase, the memory bandwidth is less used than without READ^2:

- Without READ^2, $D = 66Mo/s$ and $B_m = 190MB/s$.
- With READ^2, $D = 82Mo/s$ and $B_m = 172MB/s$.

The gain of memory bandwidth is about 10%. This gain may be use by another application. This will be illustrated in the next section.

5 Experimental validation

To validate the previous model, the transposition application was implemented with and without READ^2 on a four-node Alpha cluster. High performance disks were emulated by graphic cards. This system characteristics are:

- A 340MB/s (M_m) bandwidth memory bus.
- A 130MB/s (M_{io}) bandwidth PCI bus.
- A Myrinet network card with a bandwidth greater than 100MB/s on each node, using the standard GM communication library [9].
- A high speed disk with a bandwidth greater than 80MB/s (M_r), emulated by a Matrox G200 graphics card.

5.1 READ^2 implementation

Without READ^2, the local graphic card (also called framebuffer) first writes data in the local main memory. Then data are stored in the local Myrinet card memory to be sent to the remote Myrinet card memory. Then data are stored in the remote main memory to be transposed. They are finally written in the remote framebuffer.

With READ^2, we modified the Myrinet GM standard communication library to let the framebuffer access to the Myrinet memory by DMA. The local framebuffer writes data in the local Myrinet card memory to be

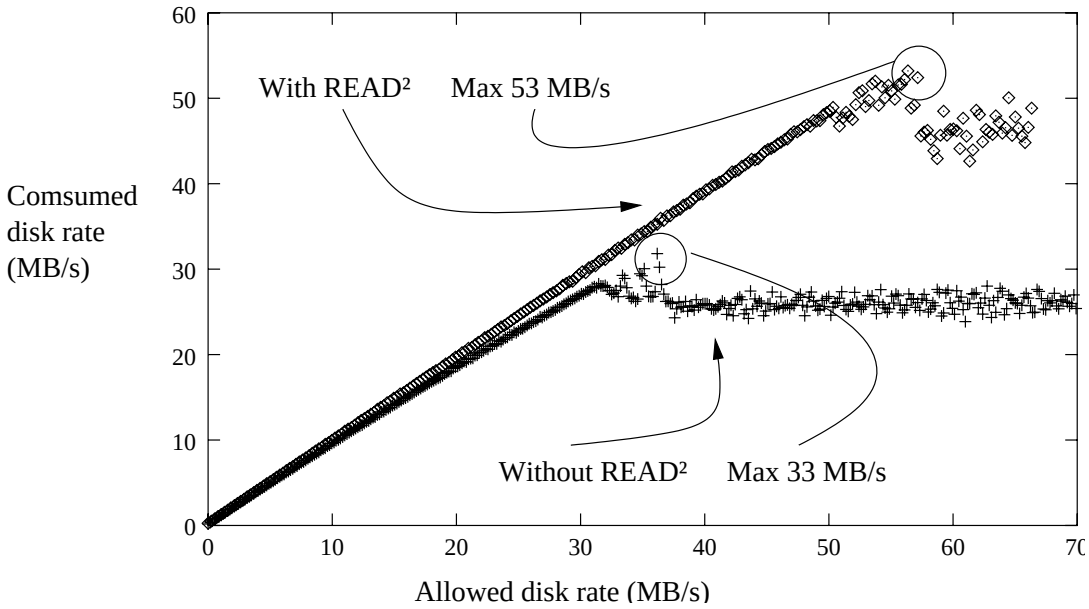


Figure 3: Maximum disk rate achievable with and without READ^2

sent to the remote Myrinet card memory. Then data are stored in the remote main memory to be transposed. They are finally written in the remote framebuffer.

The control flow is performed by the processors using pooling. The emulated disk bandwidth is controlled by waiting loops.

5.2 Experimental Results

We measured the transposition data exchange rate for different disk bandwidths, with and without READ^2. Then, the same measurements were established with an additional application running concurrently. Experience results for one node are displayed in Figure 3. Raw performance of the architecture is not achieve mainly because of the synchronisation overhead of our implementation. However, we get significant improvement:

- Without READ^2, the consumed disk rate is limited to 33 MB/s while the allowed disk rate can increase. In fact, an allowed disk rate greater than 36 MB/s gives no more improvement.
- With READ^2, disk rate consumed over disk rate allowed ratio is always grater than the ratio obtained without READ^2. In addition, The disk rate consumed reaches 53 MB/s when the allowed disc rate equals 56 MB/s.

So, READ^2 takes a greater advantage of the allowed disk rate.

A second experience introduces the execution of a concurrent application which requires only main memory access and cpu (eg filter application). Experience results are displayed in Figure 4. Plots show what are the used memory bandwith (so the speed) of the concurrent filter application according to the transposition rate. In both situations, with and without READ^2, the concurrent filter process does not modify the transposition peak bandwidth because we gave maximum priority to the transposition application.

- Without READ^2: Memory contentions appear when disk rate increase, so memory bandwidth available for filter and transposition decrease.
- With READ^2: They are less memory contentions because READ^2 needs less main memory bandwidth for remote disk accesses.

Thanks to READ^2, memory bandwith is not wasted and can be use to improve performance of concurrent application.

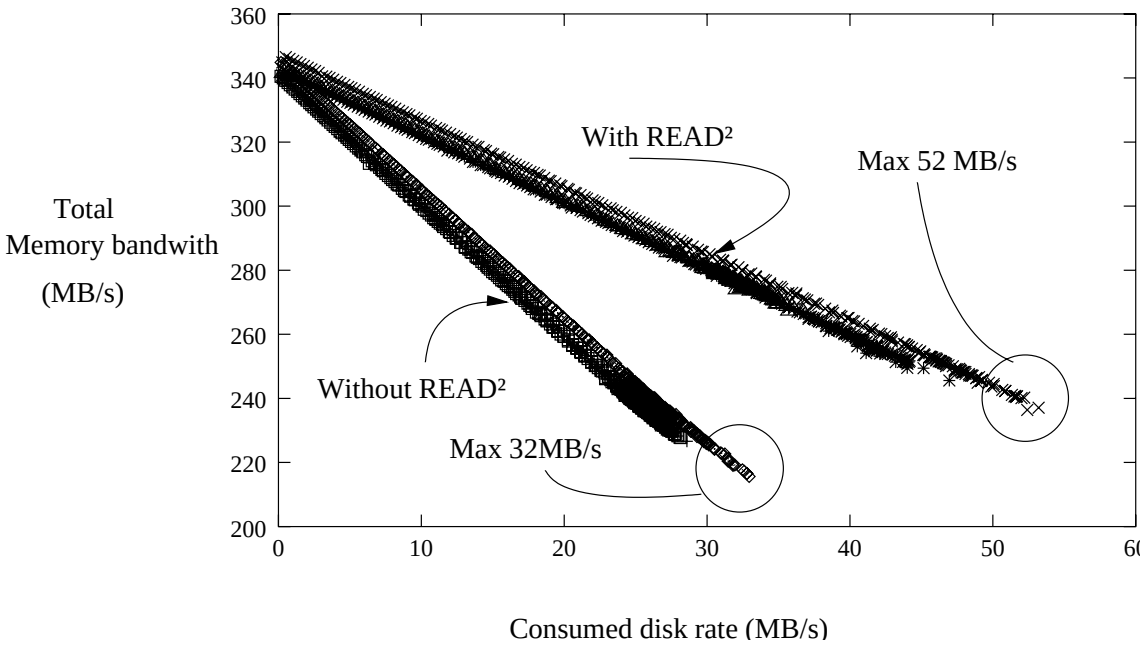


Figure 4: Memory contentions gains with and without READ^2

6 Conclusion

In this paper we investigated what are the benefits of direct remote access to distant device, called READ^2, for streaming applications on cluster. We proposed a model for streaming applications to derive the different bus usages: there are two main gains. The first one is an increase of the peak performance of streaming applications. The second one is a reduction of bus memory usage and then memory contention. These benefits were experimentally validated on a cluster where high performance disks was emulated by graphic cards.

The next step of our work is a validation of our model with real disks. Currently we have embedded a READ^2 SCSI driver in the GM firmware of Myrinet cards. We are now able to directly access to any disk from any node in the cluster. We plan to study the benefit of such an architecture for an external two-pass parallel sorting algorithm.

Another work in progress is to use the READ^2 approach for an implementation of the Global File System [10] on cluster. GFS was implemented on Fibre Channel technology where disks are directly accessed on network. GFS was ported on cluster architectures with one data server on each node (P2PS approach). We plan to analyse what will be the benefit at the system level of our new READ^2 implementation.

References

[1] Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau, David E. Culler, Joseph M. Hellerstein, and David Patterson. High performance sorting on networks of workstations. In *SIGMOD'97*, Tucson, Arizona, May 1997.

[2] Remzi H. Arpaci-Dusseau, Andrea C, Arpaci-Dusseau, David E. Culler, Joseph M. Hellerstein, and David A. Patterson. The Architectural Costs of Streaming I/O: A comparaison of Workstations, Clusters and SMPs.

[3] DAFS collaborative. *Direct Access File System version 1.0.* 2001. http://www.dafscollaborative.org/.

[4] G. A. Gibson and R. Van Metter. Network attached storage architecture. *Communication of The ACM*, 43(11), November 2000.

[5] Wolfgang Hoschek, Javier Jaen-Martinez, Asad Samar, Heinz Stockinger, and Kurt Stockinger. Data management in an internationnal data Grid project. In *GRID 2000*, Bangalore (Inde), 17-20 december 2000.

[6] Jay Huber, Christopher L. Elford, Daniel A. Reed, Andrew A. Chien, and David S. Blumenthal. PPFS: A high performance portable parallel file system. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 385–394, Barcelona, July 1995. ACM Press.

[7] Jonhatan Ilroy, Cyril Randriamaro, and Gil Utard. Improving MPI-IO Performance on PVFS. In *EuroPar'2001*, Manchester, UK, August 2001.

[8] W.B. Ligon III and R. B. Ross. An Overview of the Parallel Virtual File System. In *Proc. of the 1999 Extreme Linux Workshop*, June 1999.

[9] Myricom. *Myricom GM myrinet software and documentation*. 2000. http://www.myri.com.

[10] Kenneth W. Preslan, Adrew P. Barry, Jonatha E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steeven R. Soltis, David C. Teigland, and Matthew T. O'Keefe. A 64-bit, Shared Disk File System for Linux. In *16th Mass Storage Systems Symposium*, San Diego, March 15-18 1999. IEEE.

[11] Henri E. Bal Raoul A.F. Bhoedjang, Tim Ruhl. *Scalable Coherent Interface*. November 1998.

[12] Nisha Talagala, Satoshi Asami, David Patterson, Bob Futernick, and Dakin Hart. The Art of Massive Storage: A Web Image Archive. *IEEE Computer Journal*, 33(11), November 2000.

[13] S. Walton, A. Hutto, and J. Touch. Efficient high-speed data paths for IP forwarding using host based routers. In *1Oth IEEE Workshop on Local and Metropolitan Area Networks*, November 1998.