



I/O Scheduling Service for Multi-Application Clusters

Adrien Lebre, Guillaume Huard, Yves Denneulin

{Adrien.Lebre, Guillaume.Huard, Yves.Denneulin}@imag.fr

Laboratoire ID-IMAG (UMR 5132), Grenoble, France.

BULL - HPC, Échirolles, France.



Przemyslaw Sowa

sowa@icis.pcz.pl

Institute of Computer and Information Sciences

Czestochowa University of Technology, Poland.





Plan

- Part 1 - Parallel Input/Output and Clusters
- Part 2 - Controlling and Scheduling Multi-application I/O
- Part 3 - aIOLi, an Input/Output Scheduler for *HPC*
- Part 4 - Conclusion



Plan

Part 1 - Parallel Input/Output and Clusters

1 Introduction

- Context
- Parallel I/O

2 Parallel I/O and Clusters

- Available Solutions

3 Objectives

Part 2 - Controlling and Scheduling Multi-application I/O

Part 3 - aIOLi, an Input/Output Scheduler for *HPC*

Part 4 - Conclusion



Context

Environment

- Clusters of SMPs
- Linux
- High Performance Computing
- Intensive I/O applications
 - CPU bounded application \Rightarrow I/O bounded application
 - Remote hard drive I/O

Parallel I/O

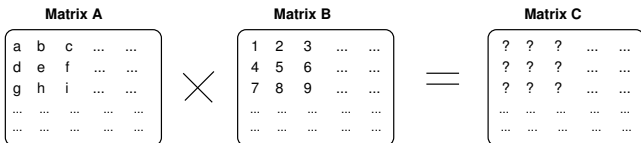
- Handling concurrent accesses to a same resource (a file)
- Accesses: different in size, in offset
Example: a parallel matrix product



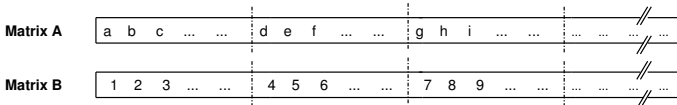
Parallel I/O - Example

Parallel matrix product

- Specific parts to fetch according to the data distribution (columns/rows, BLOCK/BLOCK ...)



Matrices are stored "row by row" in files

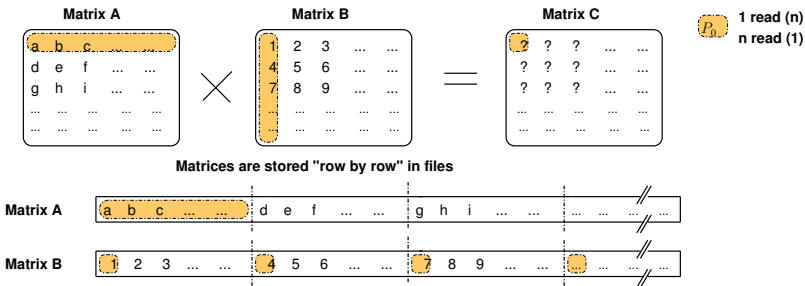




Parallel I/O - Example

Parallel matrix product

- Specific parts to fetch according to the data distribution (columns/rows, BLOCK/BLOCK ...)

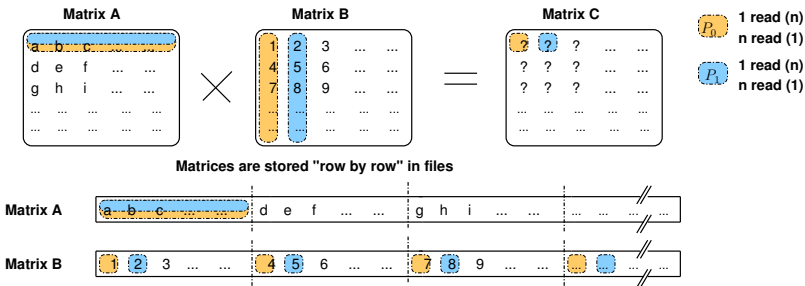




Parallel I/O - Example

Parallel matrix product

- Specific parts to fetch according to the data distribution (columns/rows, BLOCK/BLOCK ...)

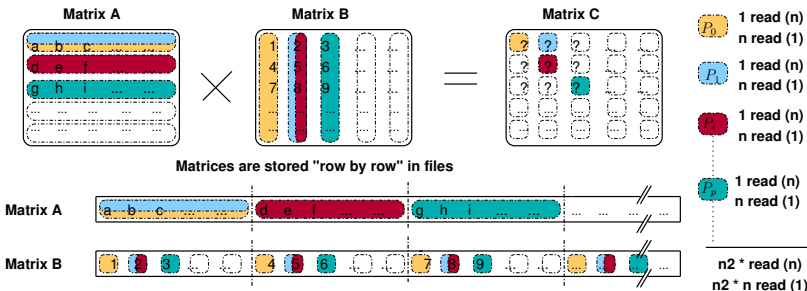




Parallel I/O - Example

Parallel matrix product

- Specific parts to fetch according to the data distribution (columns/rows, BLOCK/BLOCK ...)
- **File decomposition**: Lot of disjoint/contiguous requests at the same time
⇒ "lethal" behaviour for I/O subsystem

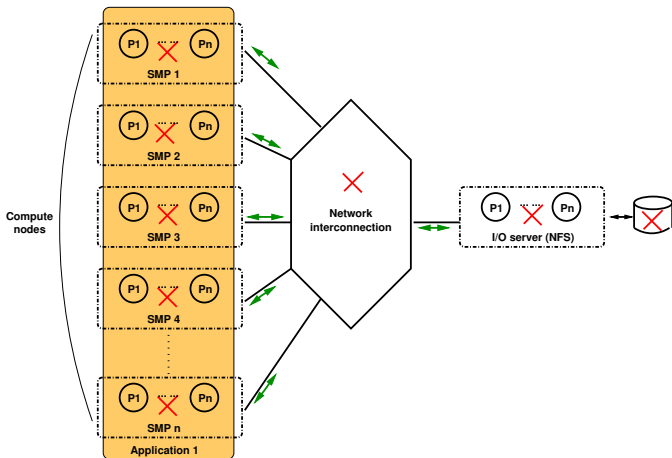


No defined order between requests ⇒ many disk head movements



Parallel I/O and Clusters

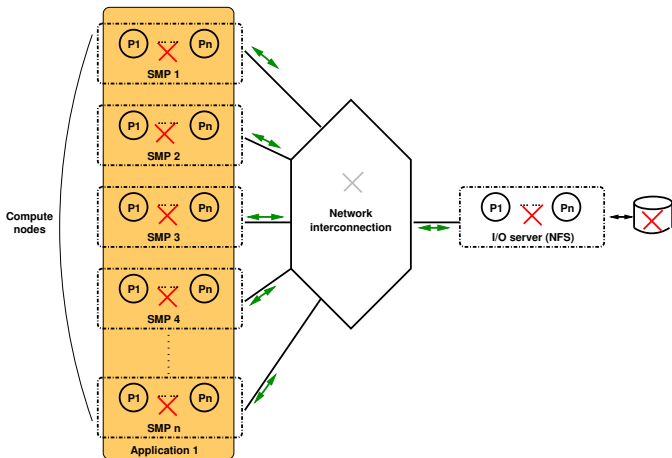
Parallel I/O \Rightarrow bottlenecks





Parallel I/O and Clusters

Parallel I/O \Rightarrow bottlenecks



Hypothesis: network has fewer impact than the I/O subsystems.



Parallel I/O Solutions (1/2)

Performance constraints

- Reduce the number of requests: decrease overhead implied by the different syscalls
- Requests **Scheduling**: avoid expensive seeks and maximize large accesses
- Exploit cache mechanisms (**read-ahead** strategies, ...)

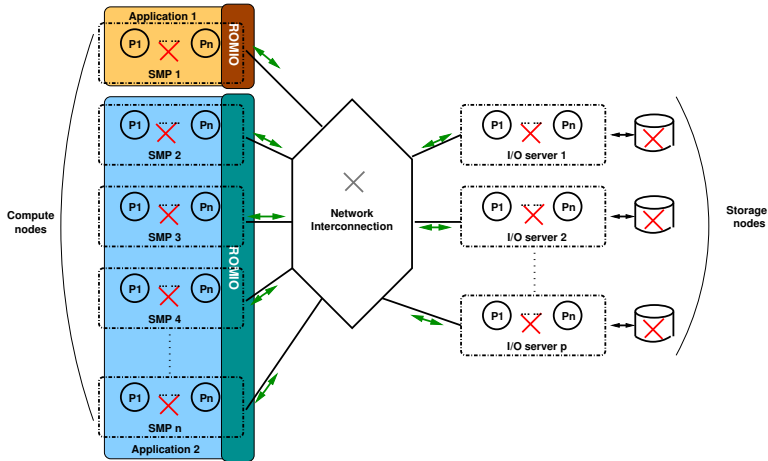
Available solutions

- **Parallel File Systems** : balance requests between several data servers
 - Designed for "Parallel I/O" (PIOUS, VESTA, ...): gave up in the time
 - More generic (PVFS, GPFS, Lustre, Parallel NFS)
 - ⇒ +/- complete / +/- efficient
- **Libraries (MPI I/O)**: Focus on **performance** and **portability** aspects
 - Sophisticated API ⇒ Development overhead / Language bindings
 - No global coordination



Parallel I/O Solutions (2/2)

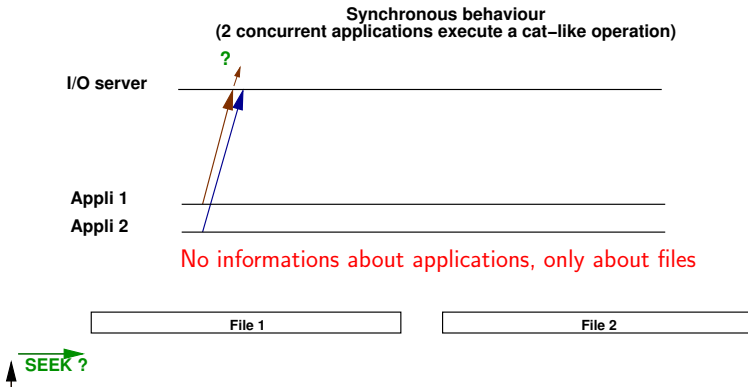
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

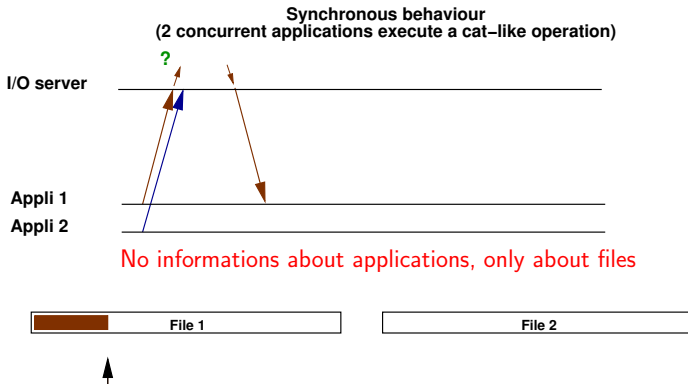
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

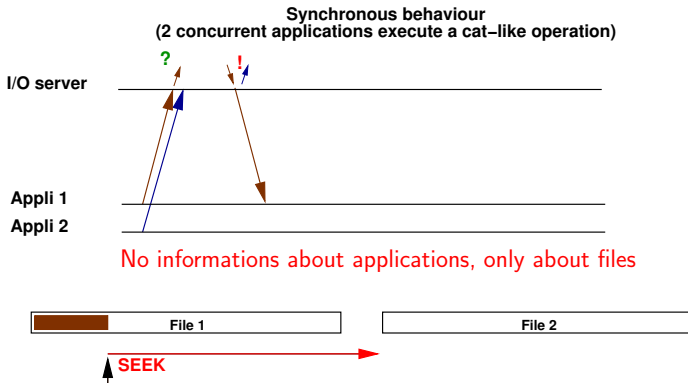
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

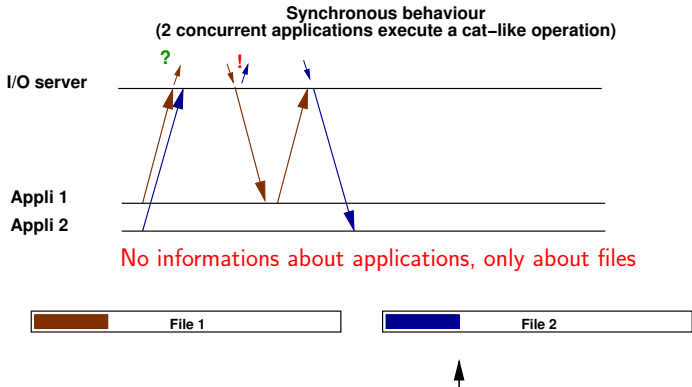
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

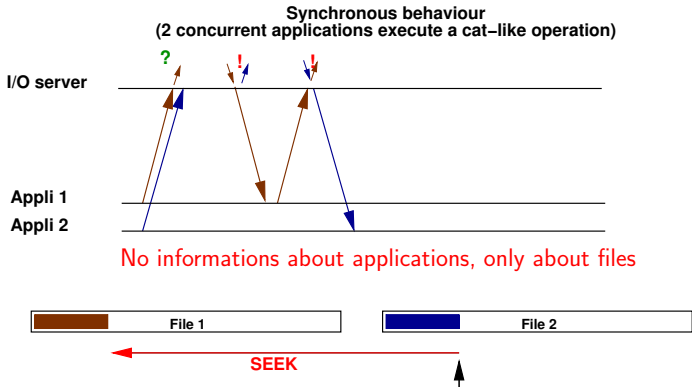
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

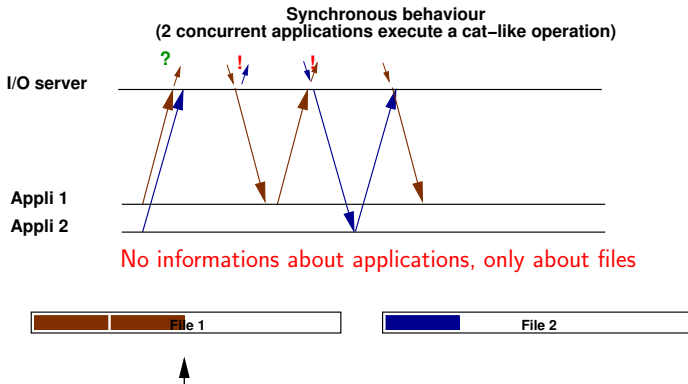
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

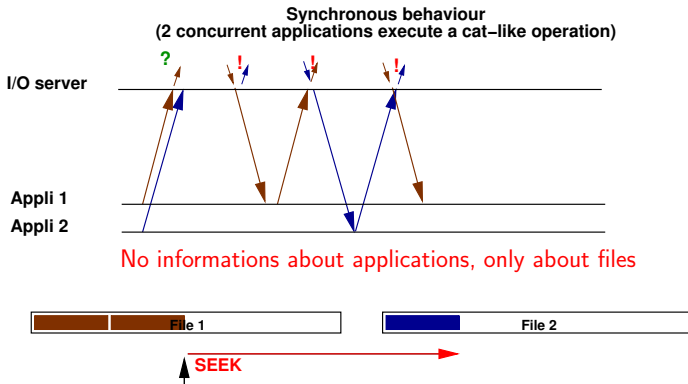
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

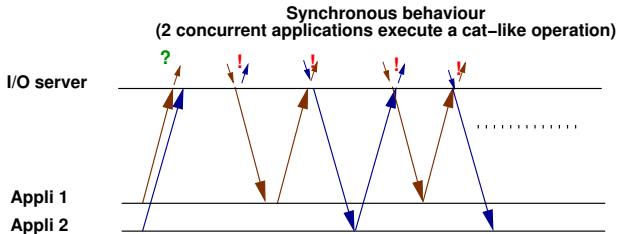
Libraries in multi-application environment





Parallel I/O Solutions (2/2)

Libraries in multi-application environment



To switch from one file to the other implies a disk head movement



Libraries are not suited

global synchronization is required



A New Approach

Objectives

- Supply **Parallel I/O** algorithms
scheduling / aggregating / overlapping access ⇒ **mono-application efficiency**
- Only **through** the use of the ubiquitous **POSIX** calls:
open/read/write/lseek/close ⇒ **portability** / **simplicity**
- **Address** requests in a global manner ⇒ **multi-application efficiency**

Naive approach:

Processing all the requests from one application before serving another one

Not suited for a cluster ⇒ **Tradeoff** between "**fairness**" and **performance**



Plan

Part 1 - Parallel Input/Output and Clusters

Part 2 - Controlling and Scheduling Multi-application I/O

4 Scheduling in Multi-application Environment

- General Algorithm

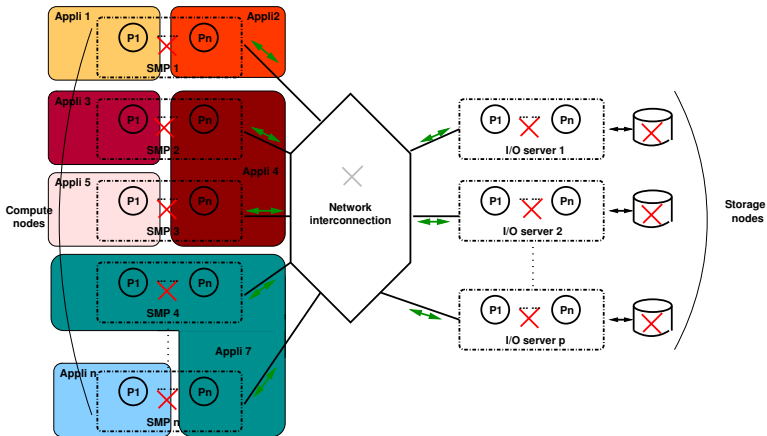
5 Synchronous Behaviour

Part 3 - aIOLi, an Input/Output Scheduler for *HPC*

Part 4 - Conclusion

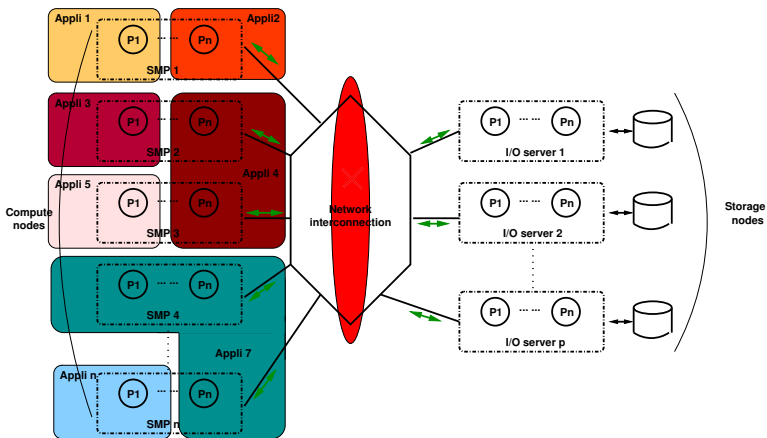


Multi-application Scheduling



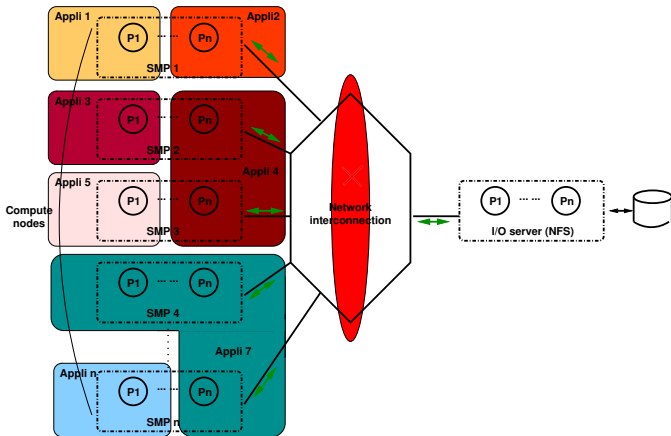


Multi-application Scheduling





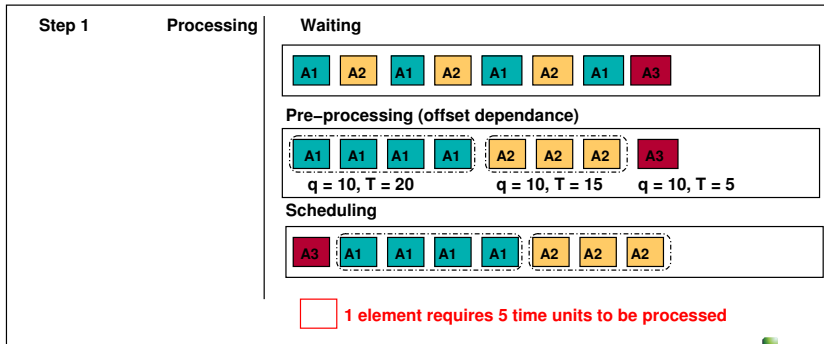
Multi-application Scheduling





Multi-application Scheduling

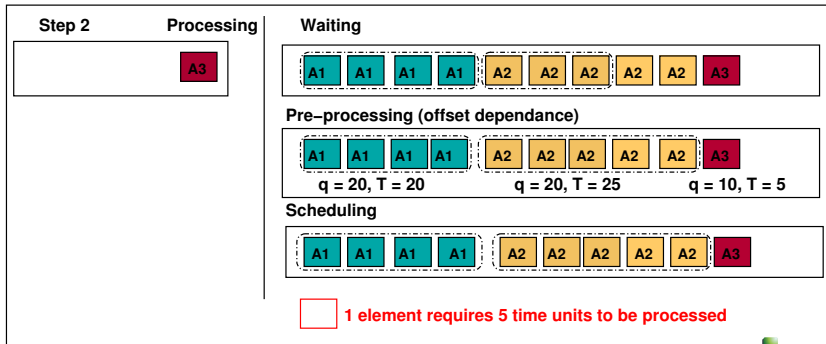
- "**Online**" problem: several requests from distinct applications are delivered to the file systems.
- Wished criteria : "**Efficiency**" with "**fairness**" constraints
⇒ Maximize the minimum of instantaneous **throughput** for each application
- Algorithm : **Multi-Level Feedback** variant (quantum approach)





Multi-application Scheduling

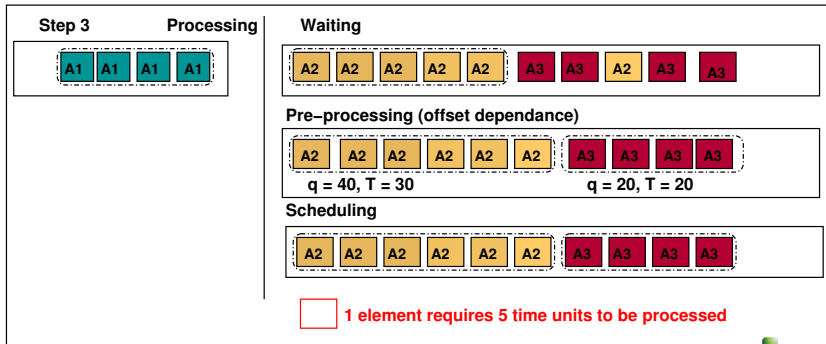
- "**Online**" problem: several requests from distinct applications are delivered to the file systems.
- Wished criteria : "**Efficiency**" with "**fairness**" constraints
⇒ Maximize the minimum of instantaneous **throughput** for each application
- Algorithm : **Multi-Level Feedback** variant (quantum approach)





Multi-application Scheduling

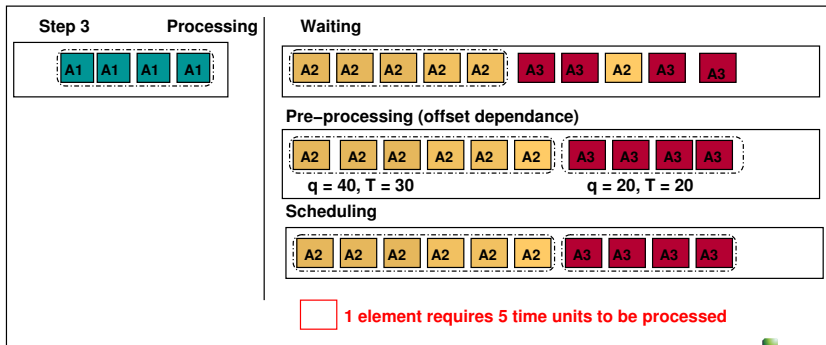
- "**Online**" problem: several requests from distinct applications are delivered to the file systems.
- Wished criteria : "**Efficiency**" with "**fairness**" constraints
⇒ Maximize the minimum of instantaneous **throughput** for each application
- Algorithm : **Multi-Level Feedback** variant (quantum approach)





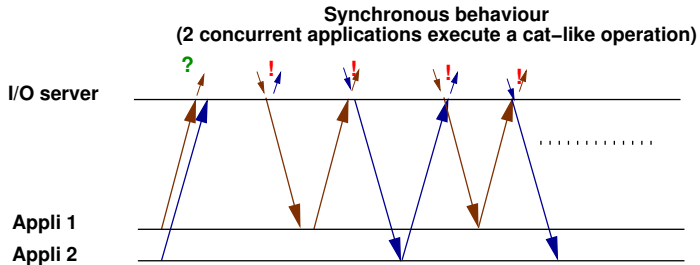
Multi-application Scheduling

- "**Online**" problem: several requests from distinct applications are delivered to the file systems.
- Wished criteria : "**Efficiency**" with "**fairness**" constraints
⇒ Maximize the minimum of instantaneous **throughput** for each application
- Algorithm : **Multi-Level Feedback** variant (quantum approach)
- The grow of a quantum could be set for each application (QoS)





Manage Synchronous Behaviour in an Efficient Way

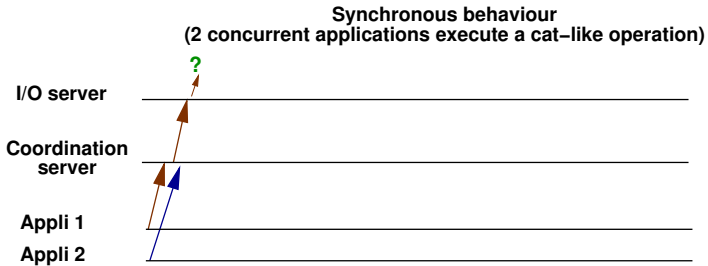


To switch from one file to the other implies a disk head movement
⇒ Serialize and define "dedicated" windows

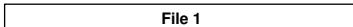




Manage Synchronous Behaviour in an Efficient Way

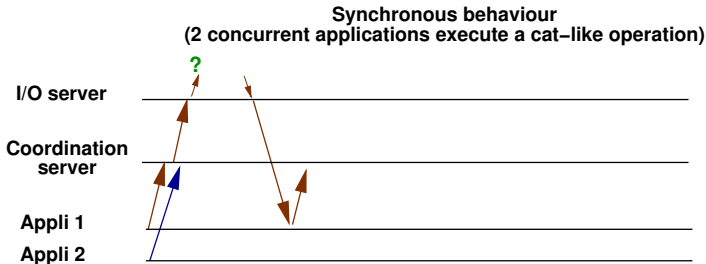


Serialize and define dedicated windows





Manage Synchronous Behaviour in an Efficient Way

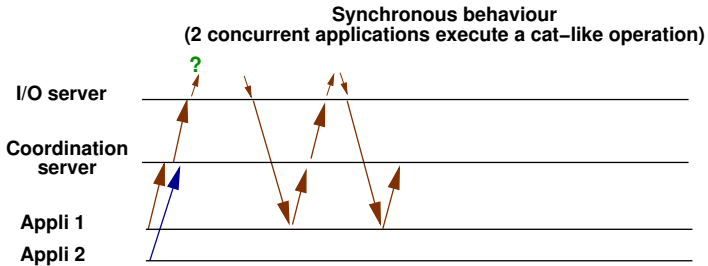


Serialize and define dedicated windows





Manage Synchronous Behaviour in an Efficient Way

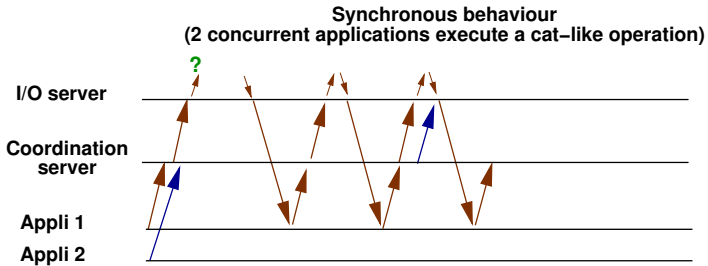


Serialize and define dedicated windows





Manage Synchronous Behaviour in an Efficient Way

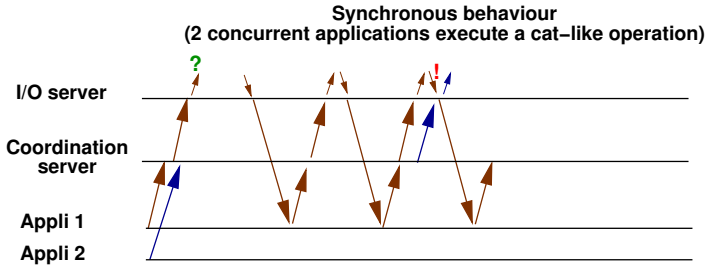


Serialize and define dedicated windows

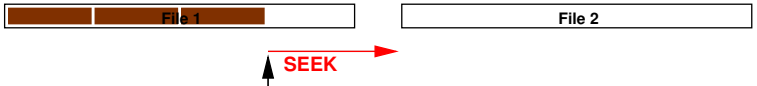




Manage Synchronous Behaviour in an Efficient Way

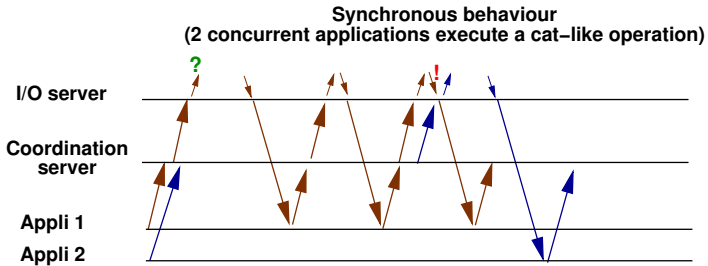


Serialize and define dedicated windows





Manage Synchronous Behaviour in an Efficient Way

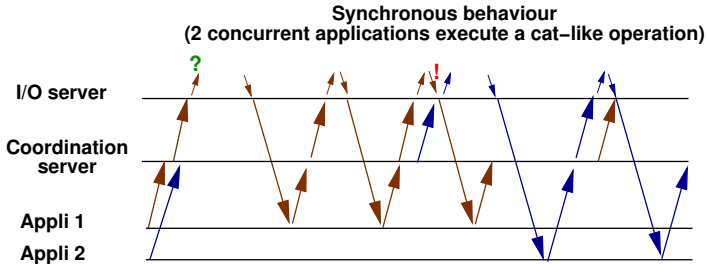


Serialize and define dedicated windows





Manage Synchronous Behaviour in an Efficient Way



Serialize and define dedicated windows



Decrease the number of seeks and exploit read-ahead mechanism



Plan

Part 1 - Parallel Input/Output and Clusters

Part 2 - Controlling and Scheduling Multi-application I/O

Part 3 - aIOLi, an Input/Output Scheduler for *HPC*

6 aIOLi - "Generic" Framework

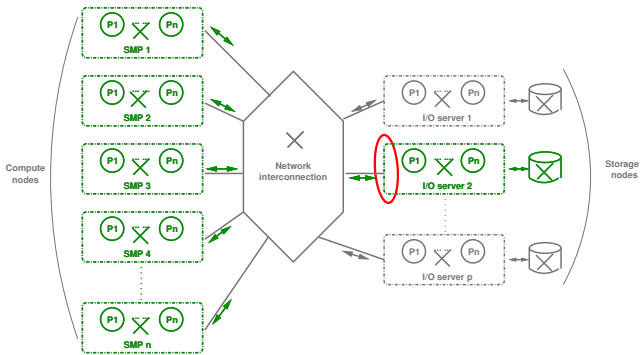
7 aIOLi - Evaluations

- Evaluations - Multi-nodes
- Eval - 2 applications
- Evaluations - 10 applications

Part 4 - Conclusion



aIOLi - "Generic" Framework



Objectives

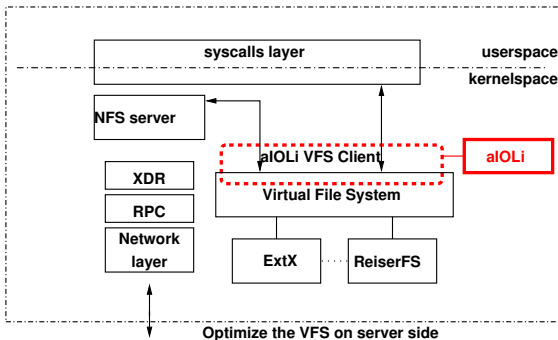
- A central point is required to apply our global strategy :
model Client/Server like PANDA (explicit centralization) is not scalable
⇒ Exploit available central point
- Major change : I/O systems become clients of our framework!



aIOli - Framework

Implementation

- *Virtual File System (server side)*

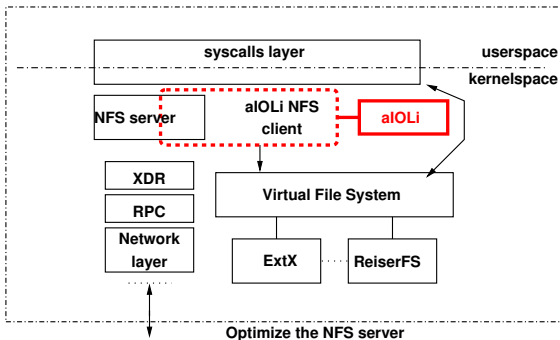




aIOli - Framework

Implementation

- *Virtual File System (server side)*
- NFS server (Version 3)





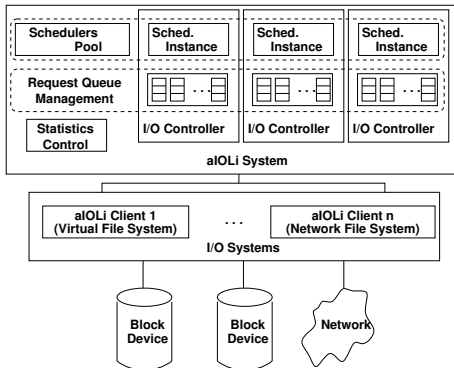
aIOli - Framework

Implementation

- *Virtual File System (server side)*
- NFS server (Version 3)

Technical aspects

- Linux kernel module - 3 functions to plug an I/O system





aIOLi - Evaluations

Platform

- Grid5000 : Sophia-Antipolis cluster (AMD 64, IDE HDD, Giga Ethernet)
- 1 to 96 nodes
- 1 dedicated NFS server
- IOR benchmark (LLNL)

Experiments

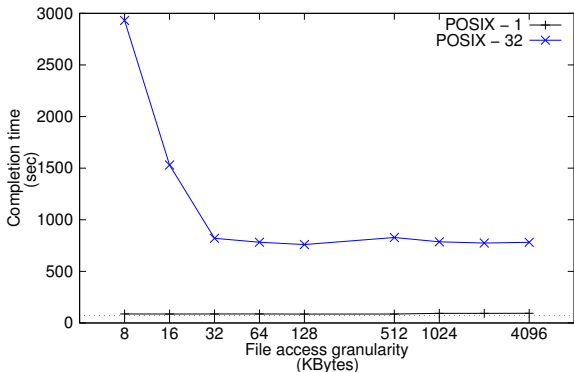
- 1 parallel application :
Valid parallel I/O detection and transparent optimisations
- 2 applications :
Analyse mutual impact and interest of a global strategy
- 10 applications :
Evaluate a "real" case



Evaluations : One Parallel Application

4 GB File decomposition including 32 MPI instances

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



32 processes deployed on 32 nodes

Observations

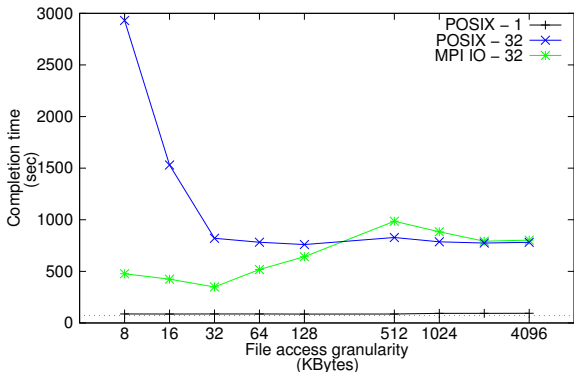
- Time !



Evaluations : One Parallel Application

4 GB File decomposition including 32 MPI instances

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



32 processes deployed on 32 nodes

Observations

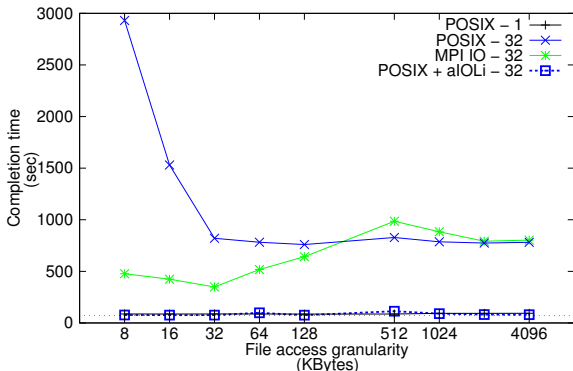
- Time !



Evaluations : One Parallel Application

4 GB File decomposition including 32 MPI instances

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



32 processes deployed on 32 nodes

Observations

- Time !
- aIOli provides significant improvements

$$11 < \frac{TP_{\text{Posix}}}{Ta_{\text{IOli}}} < 50$$

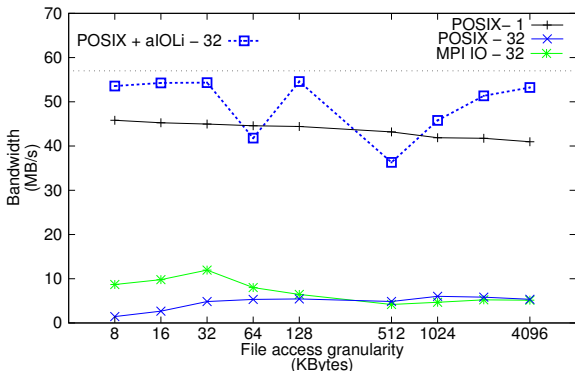
$$3.5 < \frac{TM_{\text{PIO}}}{Ta_{\text{IOli}}} < 6.5$$



Evaluations : One Parallel Application

4 GB File decomposition including 32 MPI instances

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



32 processes deployed on 32 nodes

Observations

- Time !
- aIOLi provides significant improvements

$$11 < \frac{TPosix}{TaIOLi} < 50$$

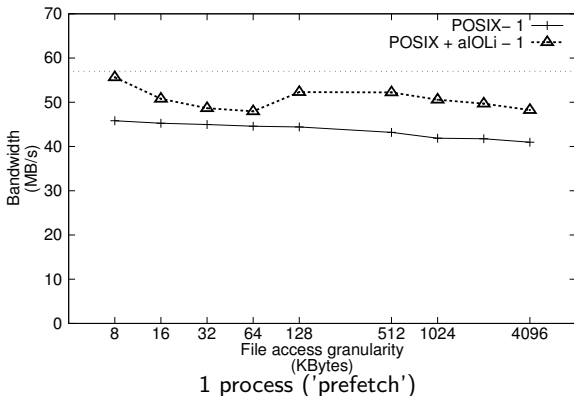
$$3.5 < \frac{TMPIIO}{TaIOLi} < 6.5$$



Evaluations : One Parallel Application

4 GB File decomposition including 32 MPI instances

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



Observations

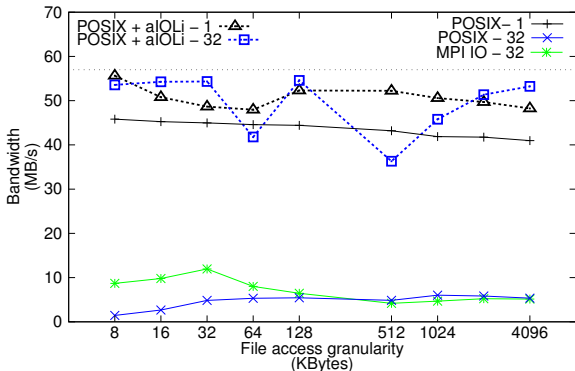
- Time !
- aIOli provides significant improvements
$$11 < \frac{TPosix}{TaIOli} < 50$$
$$3.5 < \frac{TMPIO}{TaIOli} < 6.5$$
- Synchronous behaviours benefit from aIOli



Evaluations : One Parallel Application

4 GB File decomposition including 32 MPI instances

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



Observations

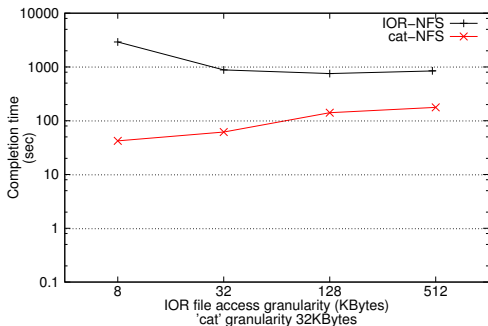
- Time !
- aIOLi provides significant improvements
$$11 < \frac{TPosix}{TaIOLi} < 50$$
$$3.5 < \frac{TMPIIO}{TaIOLi} < 6.5$$
- Synchronous behaviours benefit from aIOLi

Mono-application OK ! next step: global coordination

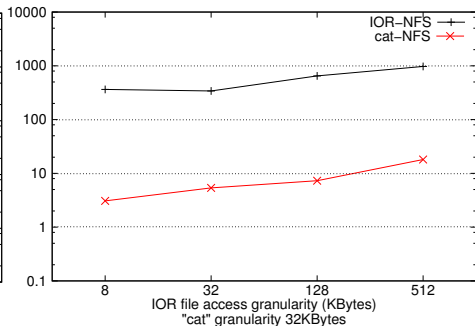


Evaluations : Multi-application Mode - Case 1

Impact of a 4 GB decomposition (32 processes - 32 nodes) over a cat of 16 MB
kernel 2.6.15, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



read - POSIX



read - MPI I/O

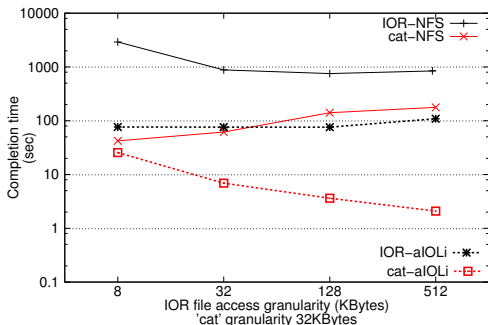
Observations

- Impact : 16MB in 100 sec (Y axis : "log" scale), The use of MPI I/O reduces the load on the server

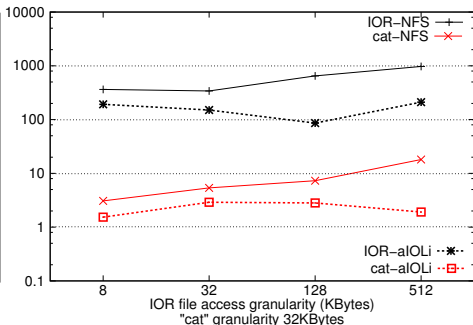


Evaluations : Multi-application Mode - Case 1

Impact of a 4 GB decomposition (32 processes - 32 nodes) over a cat of 16 MB
kernel 2.6.15, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO



read - POSIX



read - MPI I/O

Observations

- Impact : 16MB in 100 sec (Y axis : "log" scale), The use of MPI I/O reduces the load on the server
- aIOli improves the performances for both applications



Evaluations : Multi-application Mode - Case 2

10 concurrent applications, 96 nodes, 6 GB

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO

Application details	Completion time	
	POSIX	MPI IO
4 decompos. 6 sequential. 6 GB	595	840

Time are given in seconds.



Evaluations : Multi-application Mode - Case 2

10 concurrent applications, 96 nodes, 6 GB

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO

Application details	Completion time	
	NFS POSIX	MPI IO
read decompos. - 2GB (32 nodes, granularity=128KB)	490	840
write decompos. - 2GB (32 nodes, granularity=128KB)	409	815
read decompos. - 256MB (16 nodes, granularity=8KB)	595	728
write decompos. - 128MB (8 nodes, granularity=64KB)	51	257
read sequential. - 1GB (1 node, granularity=2MB)	558	59
write sequential. - 512MB (1 node, granularity=2MB)	192	71
read sequential. - 32MB (1 node, granularity=4KB)	531	9
write sequential. - 32MB (1 node, granularity=4KB)	208	9
read sequential. - 4MB (1 node, granularity=32KB)	57	1.5
write sequential. - 4MB (1 node, granularity=32KB)	39	2

Time are given in seconds.



Evaluations : Multi-application Mode - Case 2

10 concurrent applications, 96 nodes, 6 GB

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO

Application details	Completion time	
	NFS POSIX	MPI IO
read decompos. - 2GB (32 nodes, granularity=128KB)	490	840
write decompos. - 2GB (32 nodes, granularity=128KB)	409	815
read decompos. - 256MB (16 nodes, granularity=8KB)	595	728
write decompos. - 128MB (8 nodes, granularity=64KB)	51	257
read sequential. - 1GB (1 node, granularity=2MB)	558	59
write sequential. - 512MB (1 node, granularity=2MB)	192	71
read sequential. - 32MB (1 node, granularity=4KB)	531	9
write sequential. - 32MB (1 node, granularity=4KB)	208	9
read sequential. - 4MB (1 node, granularity=32KB)	57	1.5
write sequential. - 4MB (1 node, granularity=32KB)	39	2

Time are given in seconds.



Evaluations : Multi-application Mode - Case 2

10 concurrent applications, 96 nodes, 6 GB

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO

Application details	Completion time			
	NFS		NFS+aIOLi	
	POSIX	MPI IO	POSIX	MPI IO
4 decompos. 6 sequential. 6 GB	595	840	143	604

Time are given in seconds.



Evaluations : Multi-application Mode - Case 2

10 concurrent applications, 96 nodes, 6 GB

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO

Applications	Completion time			
	NFS		NFS+aIOLi	
	POSIX	MPI IO	POSIX	MPI IO
read decompos. - 2GB (32 nodes, granularity=128KB)	490	840	134	500
write decompos. - 2GB (32 nodes, granularity=128KB)	409	815	107	604
read decompos. - 256MB (16 nodes, granularity=8KB)	595	728	104	415
write decompos. - 128MB (8 nodes, granularity=64KB)	51	257	14.5	247
read sequential - 1GB (1 node, granularity=2MB)	558	59	143	54
write sequential. - 512MB (1 node, granularity=2MB)	192	71	84	61.5
read sequential. - 32MB (1 node, granularity=4KB)	531	9	48.5	3
write sequential. - 32MB (1 node, granularity=4KB)	208	9	47	6
read sequential. - 4MB (1 node, granularity=32KB)	57	1.5	6	1
write sequential. - 4MB (1 node, granularity=32KB)	39	2	19	2

Time are given in seconds.



Evaluations : Multi-application Mode - Case 2

10 concurrent applications, 96 nodes, 6 GB

kernel 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO

Applications	Completion time			
	NFS		NFS+aIOLi	
	POSIX	MPI IO	POSIX	MPI IO
read decompos. - 2GB (32 nodes, granularity=128KB)	490	840	134	500
write decompos. - 2GB (32 nodes, granularity=128KB)	409	815	107	604
read decompos. - 256MB (16 nodes, granularity=8KB)	595	728	104	415
write decompos. - 128MB (8 nodes, granularity=64KB)	51	257	14.5	247
read sequential - 1GB (1 node, granularity=2MB)	558	59	143	54
write sequential. - 512MB (1 node, granularity=2MB)	192	71	84	61.5
read sequential. - 32MB (1 node, granularity=4KB)	531	9	48.5	3
write sequential. - 32MB (1 node, granularity=4KB)	208	9	47	6
read sequential. - 4MB (1 node, granularity=32KB)	57	1.5	6	1
write sequential. - 4MB (1 node, granularity=32KB)	39	2	19	2

Time are given in seconds.



Plan

Part 1 - Parallel Input/Output and Clusters

Part 2 - Controlling and Scheduling Multi-application I/O

Part 3 - aIOLi, an Input/Output Scheduler for *HPC*

Part 4 - Conclusion

8 Conclusion

9 Current and Future Works



Conclusion

Performances and I/O in multi-application environment

- **Control and schedule** I/O requests in a **global way** :
multi-criteria problem : efficiency and fairness \Rightarrow **MLF variant** proposal

aIOli, an I/O scheduler for *HPC*

- **Generic** framework to **evaluate new strategies** for I/O scheduling
- Implementation in **kernel space** : **intrusive** from system point of view **but efficient**
- **Code** available under **GPL**
- **Joint project** since the end of 2005 with **ICIS University** (Poland)



Current and Future Works

aIOLi, works in progress

- Take into account data stripping (on RAID devices and Parallel File Systems)
- Collaboration around a parallel version of NFS to evaluate the interest of an higher I/O scheduler (Brasil/France/Poland)
- Interconnexion with *Lustre* File system

Future

- Control I/O requests at different points : multi-level scheduler on client side (compute node) / on server and hard drive side \Rightarrow *cascade scheduling*
- Exploiting the meta-node concept used by modern FS to provide consistency as a central point to plug aIOLi



Questions ?

<http://aioli.imag.fr>



LIPS Project

BULL - INRIA - ID-IMAG Laboratory - ICIS Institute

Thanks