I/O Scheduling Service for Multi-Application Clusters

Adrien Lebre
Adrien.Lebre@irisa.fr

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 ...)

![Matrix Diagram](image)

Matrices are stored "row by row" in files

Matrix A

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>e</td>
<td>f</td>
<td>...</td>
</tr>
<tr>
<td>g</td>
<td>h</td>
<td>i</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Matrix B

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Matrix C

<table>
<thead>
<tr>
<th>?</th>
<th>?</th>
<th>?</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>...</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Matrix A</th>
<th>Matrix B</th>
<th>Matrix C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c ...</td>
<td>1 2 3 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>d e f ...</td>
<td>4 5 6 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>g h i ...</td>
<td>7 8 9 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>... ... ...</td>
<td>... ... ...</td>
<td>... ... ...</td>
</tr>
<tr>
<td>... ... ...</td>
<td>... ... ...</td>
<td>... ... ...</td>
</tr>
</tbody>
</table>

\[ P_0 \]

1 read (n)

n read (1)
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

<table>
<thead>
<tr>
<th>Matrix A</th>
<th>Matrix B</th>
<th>Matrix C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c ...</td>
<td>1 2 3 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>d e f ...</td>
<td>4 5 6 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>g h i ...</td>
<td>7 8 9 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>... ... ...</td>
<td>... ... ...</td>
<td>... ... ...</td>
</tr>
</tbody>
</table>

Matrices are stored "row by row" in files

<table>
<thead>
<tr>
<th>Matrix A</th>
<th>Matrix B</th>
<th>Matrix C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c ...</td>
<td>1 2 3 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>d e f ...</td>
<td>4 5 6 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>g h i ...</td>
<td>7 8 9 ...</td>
<td>? ? ? ...</td>
</tr>
<tr>
<td>... ... ...</td>
<td>... ... ...</td>
<td>... ... ...</td>
</tr>
</tbody>
</table>

1 read (n)

n read (1)
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

Matrices are stored "row by row" in files

Matrix A

Matrix B

Matrix C

Matrixes are stored "row by row" in files

Matrix A

Matrix B

Matrix C

1 read (n)
1 read (n)
1 read (n)
1 read (n)
n2 * read (n)

Adrien Lebre
© Bull-ID LIPS 2006
aiOLi - workshop Phenix - December 2006
Parallel I/O - Example

No defined order between requests ⇒ many disk head movements

Matrix A

Matrix B

Matrix C

Matrix are stored "row by row" in files
Parallel I/O - Example

No defined order between requests ⇒ many disk head movements

Matrix A

\[
\begin{array}{cccc}
    a & b & c \\
    d & e & f \\
    g & h & i \\
\end{array}
\]

×

Matrix B

\[
\begin{array}{cccc}
    1 & 2 & 3 & \ldots \\
    4 & 5 & 6 & \ldots \\
    7 & 8 & 9 & \ldots \\
\end{array}
\]

=  

Matrix C

\[
\begin{array}{cccc}
    ? & ? & ? & \ldots \\
    ? & ? & ? & \ldots \\
    ? & ? & ? & \ldots \\
\end{array}
\]

Matrix are stored "row by row" in files

One HDD provides the best performances when files are accessed in a sequential and contiguous way
Parallel I/O - Example

No defined order between requests ⇒ many disk head movements

Matrix A

- a, b, c
- d, e, f
- g, h, i

Matrix B

- 1, 2, 3
- 4, 5, 6
- 7, 8, 9

Matrix C

- p_0
- p_1
- p_2

Matrix are stored "row by row" in files

Matrix A

- a, b, c
- d, e, f
- g, h, i

Matrix B

- 1, 2, 3
- 4, 5, 6
- 7, 8, 9

Similar behavior for the matrix B
"Random order" between requests

The bigger the number of requests
the bigger the potential number of seeks
⇒ performance degradation (bottlenecks)
Parallel I/O and Clusters

Parallel I/O ⇒ bottlenecks

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

Parallel I/O ⇒ bottlenecks

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

Performance constraints

- **Reduce** the number of requests:
  decrease overhead implied by the different syscalls

- Requests **Scheduling**:
  avoid expensive seeks and maximize large accesseses

- **Exploit cache** mechanisms:
  benefit from *read-ahead* strategies, ...
Solution 1: Parallel File Systems ⇒ Balance requests between several servers
Parallel I/O Solutions (1/4)

Parallel File Systems

- **Load balancing** on several servers
- 2 types:
  - Designed for "Parallel I/O": PIOUS, VESTA, ...
    (logical view/physical placement, gave up in the time)
  - More generic: PVFS, Parallel NFS, GPFS, Lustre
    (performance/coherency/fault tolerance...)
- +/- complete
- +/- efficient / +/- intrusive (dedicated APIs at client side)
- **No "real" scheduling policy** (most of them rely on low level schedulers)

**The performance depends on the striping policy of the file system**

From a general point of view, they do not take into account application striping!
Parallel I/O Solutions (2/4)

Parallel I/O ⇒ bottlenecks

Solution 2: Libraries

MPI I/O, the standard
Parallel I/O Solutions (2/4)

Parallel I/O ⇒ bottlenecks

Solution 2: Libraries ⇒ MPI I/O, the standard
Definition of access patterns ⇒ Reduce number of requests (equivalent to "views" / access vectors like for `readv/writev`)

Matrix are stored <<row by row>> in files
Definition of access patterns $\Rightarrow$ Reduce number of requests (equivalent to ”views” / access vectors like for `readv/writev`)

Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

From a global point of view, the performance are improved but:

- Sophisticated API $\Rightarrow$ Development overhead /
- Language bindings
- No global coordination $\Rightarrow$ Impact on performances

Matrix are stored <<row by row>> in files

Pattern of $P_0$
Definition of access patterns ⇒ Reduce number of requests (equivalent to ”views” / access vectors like for `readv/writev`)

Matrix A

\[
\begin{array}{ccccccc}
\text{a} & \text{b} & \text{c} & \ldots & \ldots & \ldots & \\
\text{d} & \text{e} & \text{f} & \ldots & \ldots & \ldots & \\
\text{g} & \text{h} & \text{i} & \ldots & \ldots & \ldots & \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\
\end{array}
\]

Matrix B

\[
\begin{array}{ccccccc}
1 & 2 & 3 & \ldots & \ldots & \ldots & \\
4 & 5 & 6 & \ldots & \ldots & \ldots & \\
7 & 8 & 9 & \ldots & \ldots & \ldots & \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\
\end{array}
\]

Matrix C

\[
\begin{array}{ccccccc}
? & ? & ? & \ldots & \ldots & \ldots & \\
? & ? & ? & \ldots & \ldots & \ldots & \\
? & ? & ? & \ldots & \ldots & \ldots & \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\
\end{array}
\]

Matrix are stored <<row by row>> in files

Pattern of $P_1$

\[
\begin{array}{cccc}
\text{1} & \text{2} & \text{3} & \ldots & \ldots & \ldots & \\
\text{4} & \text{5} & \text{6} & \ldots & \ldots & \ldots & \\
\text{7} & \text{8} & \text{9} & \ldots & \ldots & \ldots & \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \\
\end{array}
\]
Definition of access patterns ⇒ Reduce number of requests (equivalent to ”views” / access vectors like for `readv/writev`)

- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

From a global point of view, the performance are improved but:

- Sophisticated API ⇒ Development overhead / Language bindings
- No global coordination ⇒ Impact on performances

Pattern of $P_i$

Matrix A

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
\text{g} & \text{h} & \text{i} & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\end{array}
\]

Matrix B

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
7 & 8 & 9 & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\end{array}
\] \quad \times \quad \begin{array}{cccccc}
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\end{array}
\] \quad = \quad \begin{array}{cccccc}
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\end{array}
\]

Matrix C

\[
\begin{array}{cccccc}
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\text{...} & \text{...} & \text{...} & \text{...} & \text{...} & \text{...} \\
\end{array}
\]
Definition of access patterns ⇒ Reduce number of requests
(equivalent to "views" / access vectors like for readv/writev)

Matrix A

Matrix B

Matrix C

Matrix are stored <<row by row>> in files

Pattern of $P_p$
Parallel I/O Solutions (3/4)

Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests
  (equivalent to ”views” / access vectors like for `readv/writev`)
- Processes coordination to efficiently balance requests
  (aggregation, load balancing but still no efficient order ...)

Matrix A

Matrix are stored <<row by row>> in files

Matrix B

Matrix C
Parallel I/O Solutions (3/4)

Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests (equivalent to "views" / access vectors like for `readv/writev`)
- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

Matrix B

```
1 2 3 ... ... 4 5 6 ... ... 7 8 9 ...
```

Matrix B each access is sent in an independent and parallel way

Application Side

File System Side

```
P_0 P_1 P_i P_n ...
```

Adrien Lebre © Bull-ID LIPS 2006
Parallel I/O Solutions (3/4)

Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests (equivalent to ”views” / access vectors like for readv/writev)
- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

Matrix B

Application Side

File System Side

SEEK

each access is sent in an independant and parallel way
Parallel I/O Solutions (3/4)

Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests (equivalent to ”views” / access vectors like for readv/writev)
- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

From a global point of view, the performance are improved but:

- Sophisticated API ⇒ Development overhead / Language bindings
- No global coordination ⇒ Impact on performance

SEEK each access is sent in an independent and parallel way

Matrix B

Application Side

File System Side

each access is sent in an independant and parallel way

SEEK
Parallel I/O Solutions (3/4)
Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests (equivalent to "views" / access vectors like for `readv`/`writev`)
- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

From a global point of view, the performance are improved but:
- Sophisticated API ⇒ Development overhead / Language bindings
- No global coordination ⇒ Impact on performances

Matrix B
each access is sent in an independent and parallel way

Application Side

File System Side

SEEK
Parallel I/O Solutions (3/4)
Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests (equivalent to "views" / access vectors like for readv/writev)
- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

从全局角度来看，性能得到了改善，但：
- Sophisticated API ⇒ Development overhead / Language bindings
- No global coordination ⇒ Impact on performances

Matrix B

每个访问是以独立且并行的方式发送的

SEEK

Application Side

File System Side

Side File System

$P_0$ $P_1$ $P_i$ $P_n$
Parallel I/O Solutions (3/4)

Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests (equivalent to "views" / access vectors like for readv/writev)
- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

Matrix B

Application Side

File System Side

Use of complementary routines to define a particular order
Parallel I/O Solutions (3/4)

Libraries - MPI I/O [MPI2-97]

- Definition of access patterns ⇒ Reduce number of requests (equivalent to "views" / access vectors like for readv/writev)

- Processes coordination to efficiently balance requests (aggregation, load balancing but still no efficient order ...)

- From a global point of view, the performance are improved but:
  - Sophisticated API ⇒ Development overhead/Language bindings
  - No global coordination ⇒ Impact on performances

Use of complementary routines to define a particular order

Matrix B

<table>
<thead>
<tr>
<th>P₀</th>
<th>P₁</th>
<th>Pᵢ</th>
<th>Pₚ</th>
</tr>
</thead>
</table>

Application Side

File System Side

Each access is sent in an independent and parallel way
Parallel I/O Solutions (3/4)

Libraries in multi-application environment
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

I/O server

Appli 1

Appli 2

File 1

File 2

No informations about applications, only about files

SEEK?
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat–like operation)

I/O server

Appli 1

Appli 2

No informations about applications, only about files

File 1

File 2
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

I/O server

Appli 1

Appli 2

No informations about applications, only about files

SEEK

File 1

File 2
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

I/O server

Appli 1
Appli 2

No informations about applications, only about files
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

No informations about applications, only about files
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

I/O server

Apply 1

Apply 2

File 1

File 2

No informations about applications, only about files
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat–like operation)

No informations about applications, only about files.
Parallel I/O Solutions (3/4)

Libraries in multi-application environment

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

I/O server

Appli 1

Appli 2

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 $\Rightarrow$ mono-application efficiency
- Only through the use of the ubiquitous POSIX calls:
  open/read/write/lseek/close $\Rightarrow$ portability / simplicity
- Address requests in a global manner $\Rightarrow$ multi-application efficiency

Naive approach:
- Processing all the requests from one application before serving another one
- Not suited for a cluster $\Rightarrow$ 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 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
  \[ \Rightarrow \text{Maximize the minimum of instantaneous throughput for each application} \]

- Algorithm: "Multi-Level Feedback" variant (quantum approach)

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Processing</th>
<th>Waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>q = 10, T = 15</td>
<td>q = 10, T = 5</td>
<td>q = 10, T = 20</td>
</tr>
</tbody>
</table>

### Processing

<table>
<thead>
<tr>
<th>Pre-processing (offset dependance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
</tr>
<tr>
<td>q = 10, T = 20</td>
</tr>
</tbody>
</table>

### Scheduling

<table>
<thead>
<tr>
<th>A3</th>
<th>A1</th>
<th>A1</th>
<th>A1</th>
<th>A1</th>
<th>A2</th>
<th>A2</th>
<th>A2</th>
</tr>
</thead>
</table>

1 element requires 5 time units to be processed
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)

```
Step 2
Processing

Waiting
A1 A1 A1 A1
A2 A2 A2 A2
A2 A2 A3

Pre-processing (offset dependance)
A1 A1 A1 A1
A2 A2 A2 A2
A2 A2 A3
q = 20, T = 20
q = 20, T = 25
q = 10, T = 5

Scheduling
A1 A1 A1 A1
A2 A2 A2 A2
A2 A2 A3
```

1 element requires 5 time units to be processed
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)

Step 3

<table>
<thead>
<tr>
<th>Processing</th>
<th>Waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 A1 A1 A1</td>
<td>A2 A2 A2 A2 A2 A3 A3 A2 A3 A3 A3</td>
</tr>
</tbody>
</table>

Pre-processing (offset dependance)

- q = 40, T = 30
- q = 20, T = 20

Scheduling

- A2 A2 A2 A2 A2 | A3 A3 A3 A3 A3

1 element requires 5 time units to be processed
Multi-application Scheduling

- "Online" problem: several requests from distinct applications are delivered to the file systems.

- Wished criteria: "Efficiency" with "fairness" constraints
  \[ \Rightarrow \] 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)

<table>
<thead>
<tr>
<th>Step 3</th>
<th>Processing</th>
<th>Waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A1 A1 A1 A1</td>
<td></td>
</tr>
</tbody>
</table>

Pre-processing (offset dependance)

- q = 40, T = 30
- q = 20, T = 20

Processing | Waiting
---|---

Scheduling

- 1 element requires 5 time units to be processed
Manage Synchronous Behaviour in an Efficient Way

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

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

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

I/O server

_coordination server

Appli 1
Appli 2

Serialize and define dedicated windows

File 1
File 2
Manage Synchronous Behaviour in an Efficient Way

Synchronous behaviour
(2 concurrent applications execute a cat−like operation)

Serialize and define dedicated windows

I/O server

Coordination server

Appli 1

Appli 2

File 1

File 2
Manage Synchronous Behaviour in an Efficient Way

Synchronous behaviour
(2 concurrent applications execute a cat–like operation)

I/O server

Coordination server

Appli 1

Appli 2

Serialize and define dedicated windows

File 1

File 2
Manage Synchronous Behaviour in an Efficient Way

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

Serialize and define dedicated windows
Manage Synchronous Behaviour in an Efficient Way

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

Serialize and define dedicated windows

I/O server
Coordination server
Appli 1
Appli 2
Manage Synchronous Behaviour in an Efficient Way

Synchronous behaviour
(2 concurrent applications execute a cat-like operation)

Serialize and define dedicated windows
Manage Synchronous Behaviour in an Efficient Way

**Synchronous behaviour**
(2 concurrent applications execute a cat-like operation)

Serialize and define dedicated windows

Decrease the number of seeks and exploit read-ahead mechanism
Synchronous behaviour and Parallel I/O

Synchronous behaviour within parallel I/O

- Due to the file system granularity
  Equivalent to $n$ synchronous accesses sent in a parallel way
Synchronous behaviour and Parallel I/O

Synchronous behaviour within parallel I/O

- Due to the file system granularity
  Equivalent to $n$ synchronous accesses sent in a parallel way
Synchronous behaviour and Parallel I/O

Synchronous behaviour within parallel I/O

- Due to the file system granularity
  Equivalent to \( n \) synchronous accesses sent in a parallel way
  \( \Rightarrow \) The quantum size is adapted according to the file access history

Client side

Scheduling queue (aIOLi)

File system granularity

Aggregation process could not be exploited

Synchronous behaviour
Synchronous behaviour and Parallel I/O

Synchronous behaviour within parallel I/O

- Due to the file system granularity
  Equivalent to $n$ synchronous accesses sent in a parallel way
  ⇒ The quantum size is adapted according to the file access history
Synchronous behaviour and Parallel I/O

Synchronous behaviour within parallel I/O

- Due to the file system granularity
  - Equivalent to \( n \) synchronous accesses sent in a parallel way
  \( \Rightarrow \) The quantum size is adapted according to the file access history
Synchronous behaviour and Parallel I/O

Synchronous behaviour within parallel I/O

- Due to the file system granularity
  Equivalent to \( n \) synchronous accesses sent in a parallel way
  \( \Rightarrow \) The quantum size is adapted according to the file access history

\[
\begin{align*}
\min \text{ bound} &< \text{Quantum} < \max \text{ bound} \\
q=4 &q=3 &q=2 &q=2 \\
q=4 &q=3 &q=2 &q=1 \\
\end{align*}
\]

- Client side
- File system granularity
- Scheduling queue (aOLi)

\[\text{min bound} < \text{Quantum} < \text{max bound}\]
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
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

Observations

- Time !

32 processes deployed on 32 nodes
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!

32 processes deployed on 32 nodes
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
  \[11 < \frac{T_{Posix}}{T_{aIOLi}} < 50\]
  \[3.5 < \frac{T_{MPIIO}}{T_{aIOLi}} < 6.5\]

32 processes deployed on 32 nodes
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

![Graph showing bandwidth vs. file access granularity for different file access models with a significant improvement observed for aIOLi]

Observations
- Time!
- aIOLi provides significant improvements

\[
11 < \frac{T_{Posix}}{T_{aIOLi}} < 50
\]
\[
3.5 < \frac{T_{MPIIO}}{T_{aIOLi}} < 6.5
\]

32 processes deployed on 32 nodes
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
  
  \[ 11 < \frac{TPosix}{TaIOLi} < 50 \]
  \[ 3.5 < \frac{TMP110}{TaIOLi} < 6.5 \]
- 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
  \[11 < \frac{T_{Posix}}{T_{aIOLi}} < 50\]
  \[3.5 < \frac{T_{MPIIO}}{T_{aIOLi}} < 6.5\]
- 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

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

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

<table>
<thead>
<tr>
<th>Application details</th>
<th>Completion time</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSIX</td>
<td>MPI IO</td>
</tr>
<tr>
<td>4 decompos. 6 sequential. 6 GB</td>
<td>595</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Application details</th>
<th>Completion time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NFS POSIX</td>
</tr>
<tr>
<td><strong>read decompos. - 2GB</strong> (32 nodes, granularity=128KB)</td>
<td>490</td>
</tr>
<tr>
<td><strong>write decompos. - 2GB</strong> (32 nodes, granularity=128KB)</td>
<td>409</td>
</tr>
<tr>
<td><strong>read decompos. - 256MB</strong> (16 nodes, granularity=8KB)</td>
<td><strong>595</strong></td>
</tr>
<tr>
<td><strong>write decompos. - 128MB</strong> (8 nodes, granularity=64KB)</td>
<td>51</td>
</tr>
<tr>
<td><strong>read sequential. - 1GB</strong> (1 node, granularity=2MB)</td>
<td>558</td>
</tr>
<tr>
<td><strong>write sequential. - 512MB</strong> (1 node, granularity=2MB)</td>
<td>192</td>
</tr>
<tr>
<td><strong>read sequential. - 32MB</strong> (1 node, granularity=4KB)</td>
<td>531</td>
</tr>
<tr>
<td><strong>write sequential. - 32MB</strong> (1 node, granularity=4KB)</td>
<td>208</td>
</tr>
<tr>
<td><strong>read sequential. - 4MB</strong> (1 node, granularity=32KB)</td>
<td>57</td>
</tr>
<tr>
<td><strong>write sequential. - 4MB</strong> (1 node, granularity=32KB)</td>
<td>39</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Application details</th>
<th>Completion time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NFS</td>
</tr>
<tr>
<td>read decompos. - 2GB (32 nodes, granularity=128KB)</td>
<td>490</td>
</tr>
<tr>
<td>write decompos. - 2GB (32 nodes, granularity=128KB)</td>
<td>409</td>
</tr>
<tr>
<td>read decompos. - 256MB (16 nodes, granularity=8KB)</td>
<td>595</td>
</tr>
<tr>
<td>write decompos. - 128MB (8 nodes, granularity=64KB)</td>
<td>51</td>
</tr>
<tr>
<td>read sequential. - 1GB (1 node, granularity=2MB)</td>
<td>558</td>
</tr>
<tr>
<td>write sequential. - 512MB (1 node, granularity=2MB)</td>
<td>192</td>
</tr>
<tr>
<td>read sequential. - 32MB (1 node, granularity=4KB)</td>
<td>531</td>
</tr>
<tr>
<td>write sequential. - 32MB (1 node, granularity=4KB)</td>
<td>208</td>
</tr>
<tr>
<td>read sequential. - 4MB (1 node, granularity=32KB)</td>
<td>57</td>
</tr>
<tr>
<td>write sequential. - 4MB (1 node, granularity=32KB)</td>
<td>39</td>
</tr>
</tbody>
</table>

Time are given in seconds.
Evaluations: Multi-application Mode - Case 2

10 concurrent applications, 96 nodes, 6 GB
kernell 2.6.12, sophia cluster, NFS version 3, mpich 1.2.5, ROMIO

<table>
<thead>
<tr>
<th>Application details</th>
<th>Completion time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NFS</td>
</tr>
<tr>
<td></td>
<td>POSIX</td>
</tr>
<tr>
<td>4 decompos. 6 sequential. 6 GB</td>
<td>595</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>Completion time</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NFS</td>
<td>POSIX</td>
<td>MPI IO</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>read decompos. - 2GB</td>
<td>490</td>
<td>134</td>
<td>500</td>
</tr>
<tr>
<td>(32 nodes, granularity=128KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write decompos. - 2GB</td>
<td>409</td>
<td>107</td>
<td>604</td>
</tr>
<tr>
<td>(32 nodes, granularity=128KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read decompos. - 256MB</td>
<td>595</td>
<td>104</td>
<td>415</td>
</tr>
<tr>
<td>(16 nodes, granularity=8KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write decompos. - 128MB</td>
<td>51</td>
<td>14.5</td>
<td>247</td>
</tr>
<tr>
<td>(8 nodes, granularity=64KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read sequential - 1GB</td>
<td>558</td>
<td>143</td>
<td>54</td>
</tr>
<tr>
<td>(1 node, granularity=2MB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write sequential - 512MB</td>
<td>192</td>
<td>84</td>
<td>61.5</td>
</tr>
<tr>
<td>(1 node, granularity=2MB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read sequential - 32MB</td>
<td>531</td>
<td>48.5</td>
<td>3</td>
</tr>
<tr>
<td>(1 node, granularity=4KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write sequential - 32MB</td>
<td>208</td>
<td>47</td>
<td>6</td>
</tr>
<tr>
<td>(1 node, granularity=4KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read sequential. - 4MB</td>
<td>57</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>(1 node, granularity=32KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write sequential. - 4MB</td>
<td>39</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>(1 node, granularity=32KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Applications</th>
<th>Completion time</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NFS</td>
<td>POSIX</td>
<td>NFS+aIOLi</td>
</tr>
<tr>
<td></td>
<td>POSIX</td>
<td>MPI IO</td>
<td>POSIX</td>
</tr>
<tr>
<td>read decompos. - 2GB</td>
<td>490</td>
<td>840</td>
<td>134</td>
</tr>
<tr>
<td>(32 nodes, granularity=128KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write decompos. - 2GB</td>
<td>409</td>
<td>815</td>
<td>107</td>
</tr>
<tr>
<td>(32 nodes, granularity=128KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read decompos. - 256MB</td>
<td>595</td>
<td>728</td>
<td>104</td>
</tr>
<tr>
<td>(16 nodes, granularity=8KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write decompos. - 128MB</td>
<td>51</td>
<td>257</td>
<td>14.5</td>
</tr>
<tr>
<td>(8 nodes, granularity=64KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read sequential - 1GB</td>
<td>558</td>
<td>59</td>
<td>143</td>
</tr>
<tr>
<td>(1 node, granularity=2MB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write sequential. - 512MB</td>
<td>192</td>
<td>71</td>
<td>84</td>
</tr>
<tr>
<td>(1 node, granularity=2MB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read sequential. - 32MB</td>
<td>531</td>
<td>9</td>
<td>48.5</td>
</tr>
<tr>
<td>(1 node, granularity=4KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write sequential. - 32MB</td>
<td>208</td>
<td>9</td>
<td>47</td>
</tr>
<tr>
<td>(1 node, granularity=4KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read sequential. - 4MB</td>
<td>57</td>
<td>1.5</td>
<td>6</td>
</tr>
<tr>
<td>(1 node, granularity=32KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>write sequential. - 4MB</td>
<td>39</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>(1 node, granularity=32KB)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Conclusion

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 ⇒ *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