Distributed implementation of the IO-Sets scheduling method

Louis Peyrondet

Summary

01

IO-Sets Refresher

IO-Sets

- IO scheduling method for supercomputers' Parallel File Systems
- Made by the Tadaam team here at Inria bordeaux
- Classify applications in « Sets »
- Each set have a priority assigned
- Work with 2 principles : Set Sharing and Exclusive Access
- We only have limited amount of informations about the applications

Sharing and Exclusive

-Sharing
- Every set has a assigned priority
- If multiple sets are doing I/O they'll share the bandwidth acording to their priorities

-Exclusive
- Inside each set multiple application could request I/O at the same time but we only allow 1 application to have access to the bandwidth

We end up with weighted sharing between all the sets, but an exclusive access inside each set.

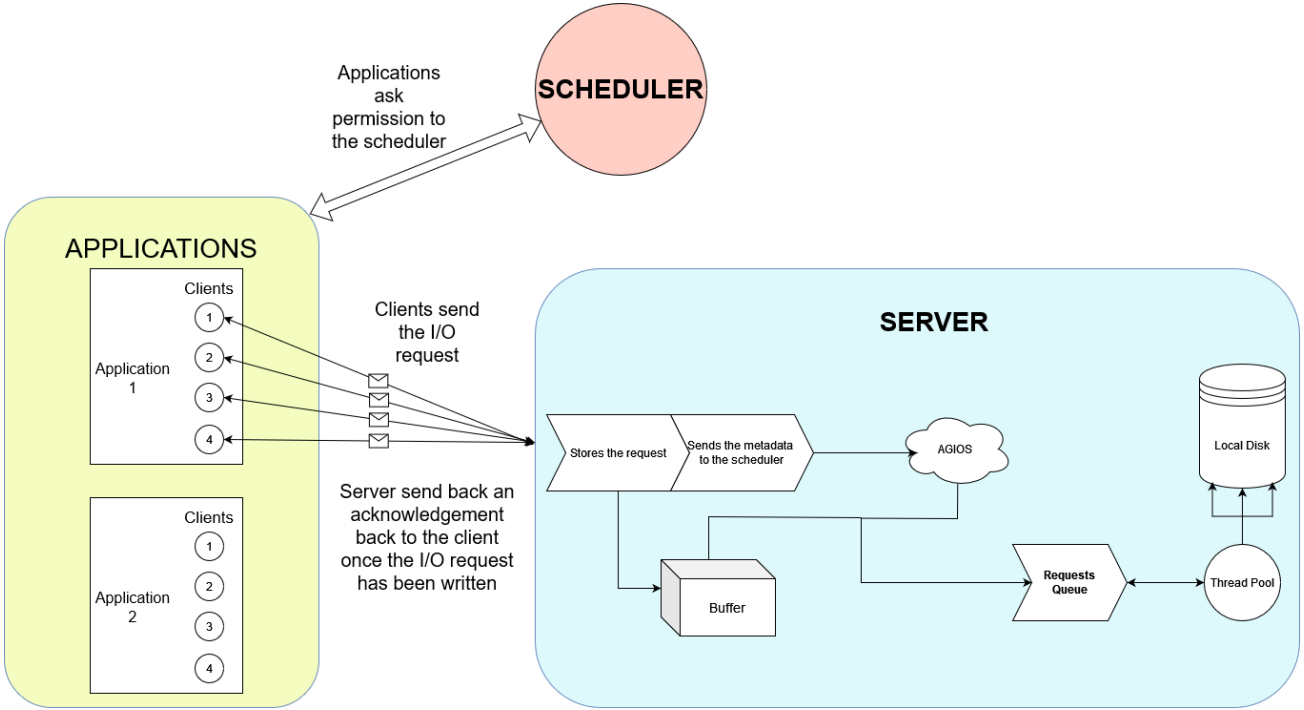
02

The IO-TestBed

What is it

- Program that I made during the internship. C langage utilizing MPI processes
- Is an environment that allows the implementation and testing of IO-Sets
- The results obtained are on a real system not a simulation
- Let you specify parameters such as the synchronization, amount of threads for concurrent writes, the scenario and implementation ...

System Architecture Diagram



Scenarios

The IO-TestBed allows you to create applications based I/O Scenario as XML files.

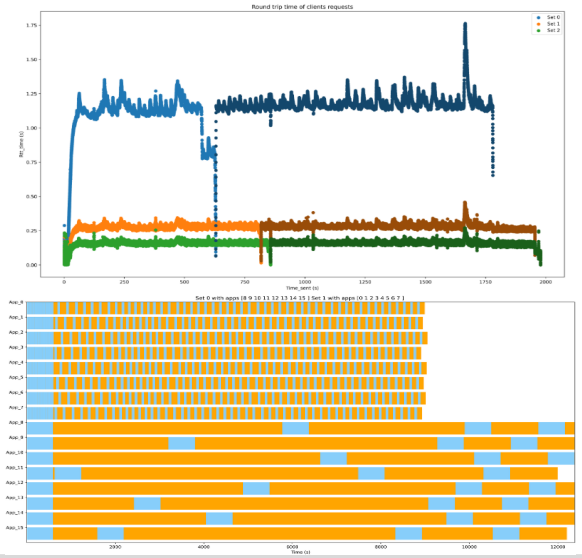
This lets you create specific situations such as many small I/O access with few large ones or vice versa.

```
<application>
    <argument name="app_1"/>           <!-- App Name -->
    <argument periods="2" />           <!-- Periodicity -->
    <argument compute="60e9" />        <!-- Compute size (flop) -->
    <argument filesize="10000000000"/> <!-- File size (bytes) -->
    <argument set="0"/>                <!-- Assigned set  -->
    <argument bd_percentage="0"/>      <!-- Bandwidth Percentage-->
    <argument start_time="0"/>         <!-- release time (s)-->
</application>
```

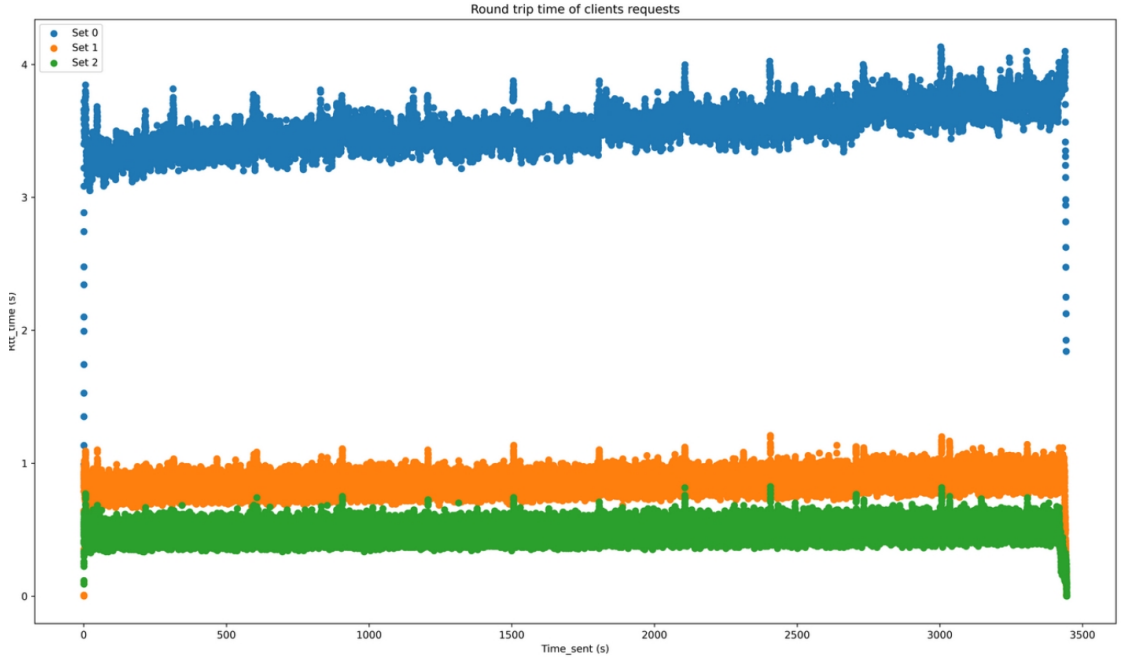
Traces and Visualisations

For each execution a variety of csv traces are generated allowing with the plotting scripts a quick pipeline from execution to visualization

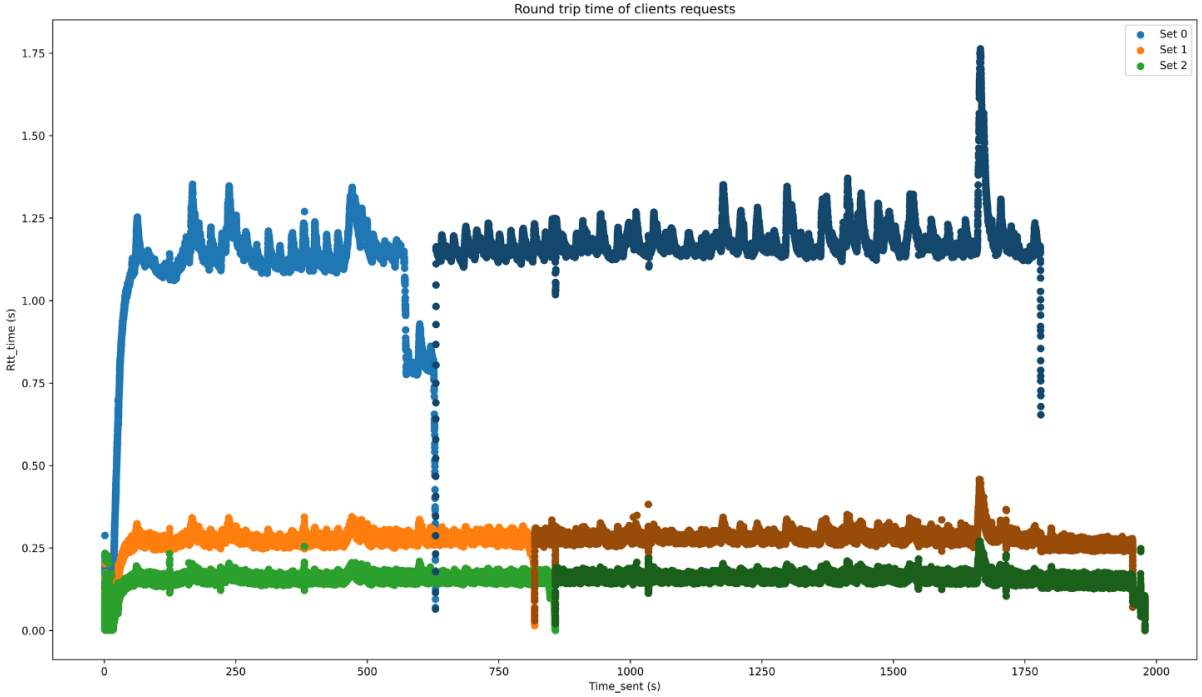
```
~$ AGIOS_CONF=~/agios.conf mpirun -np 60
build/server -threads 8 -syncronization 2
 -scenario config/standard_fast.xml -dece
ntralized 0
```



Sharing characteristic



Exclusive characteristic

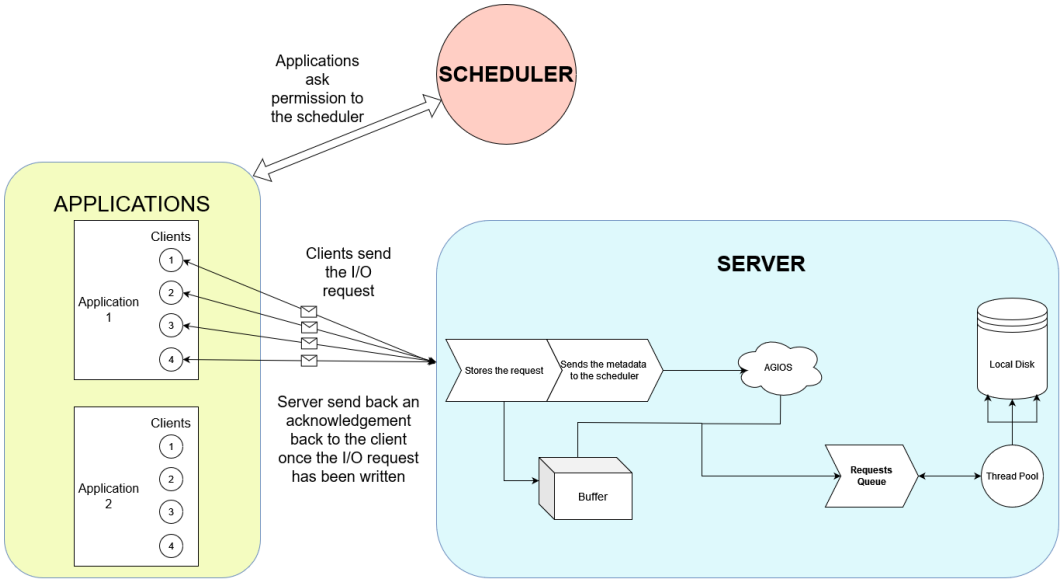


03

IO-Sets Implementation Drawbacks

Scheduler process

- Potential contention point, requires extra communications



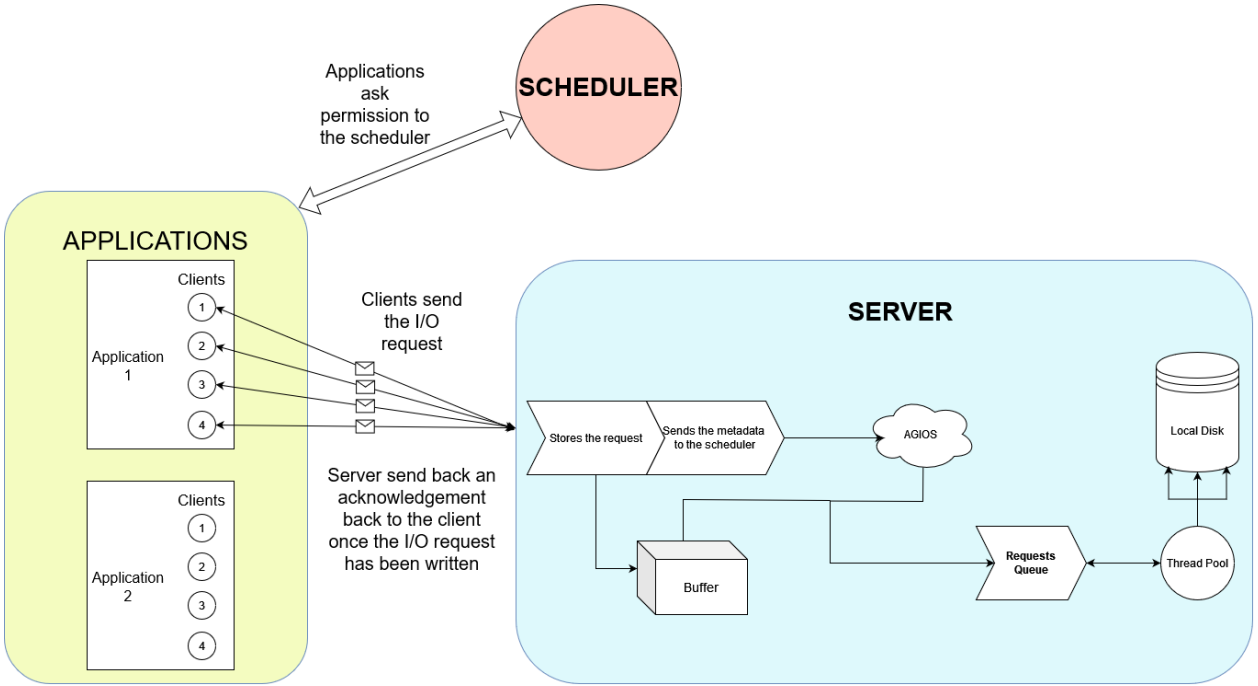
Bandwidth

- If application can only reach 10 percent of the bandwidth then scheduler still make them sequential
- Low bandwidth application actually are the majority of the application (eg: 30% uses < 10MB/s on Plafrim)
- IO-Sets would scale linearly with the amount of low bandwidth application

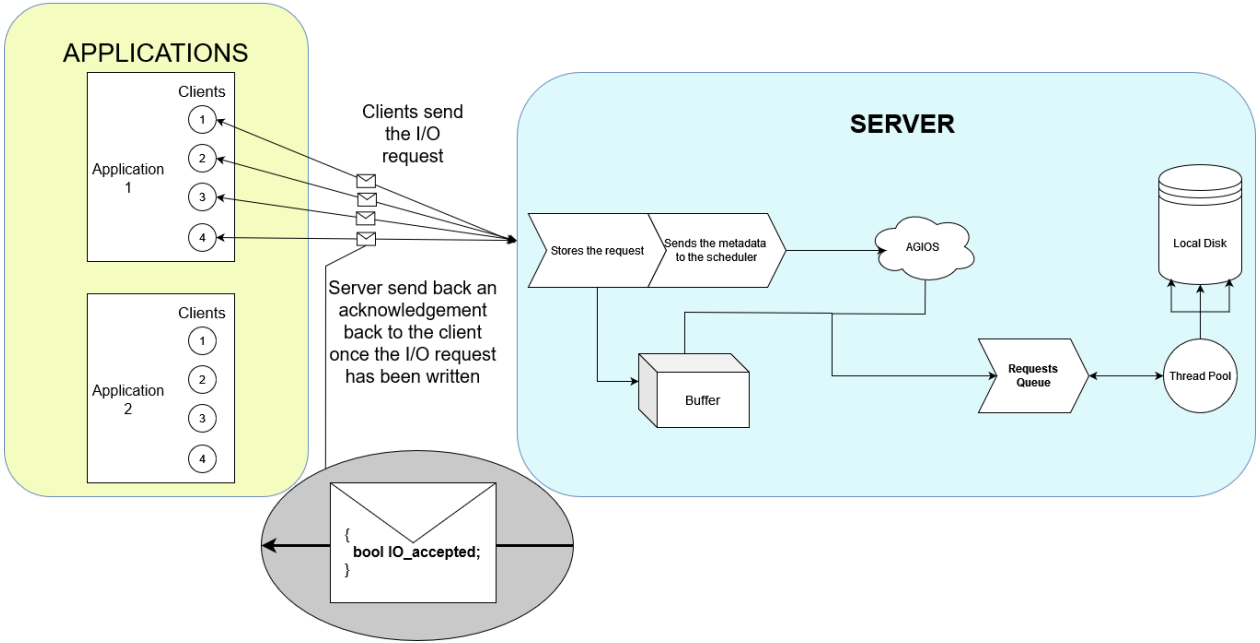
04

The New Method

Removing the scheduler



Removing the scheduler



- We need some kind of metric on the server to decide

The Server Metric

Knowing the number of sets and their respective priorities allows us to compute a table that, given the currently active sets, returns the expected percentage of total bandwidth for each set

	% Bandwidth set 0	% Bandwidth set 1	% Bandwidth set 2
000	100	100	100
001	0	0	100
010	0	100	0
011	0	~36	~64
100	100	0	0
101	~12	0	~87
110	~19.2	80	0
111	8	33	58

Bandwidth table for 3 sets with priorities {800,3300,5800}

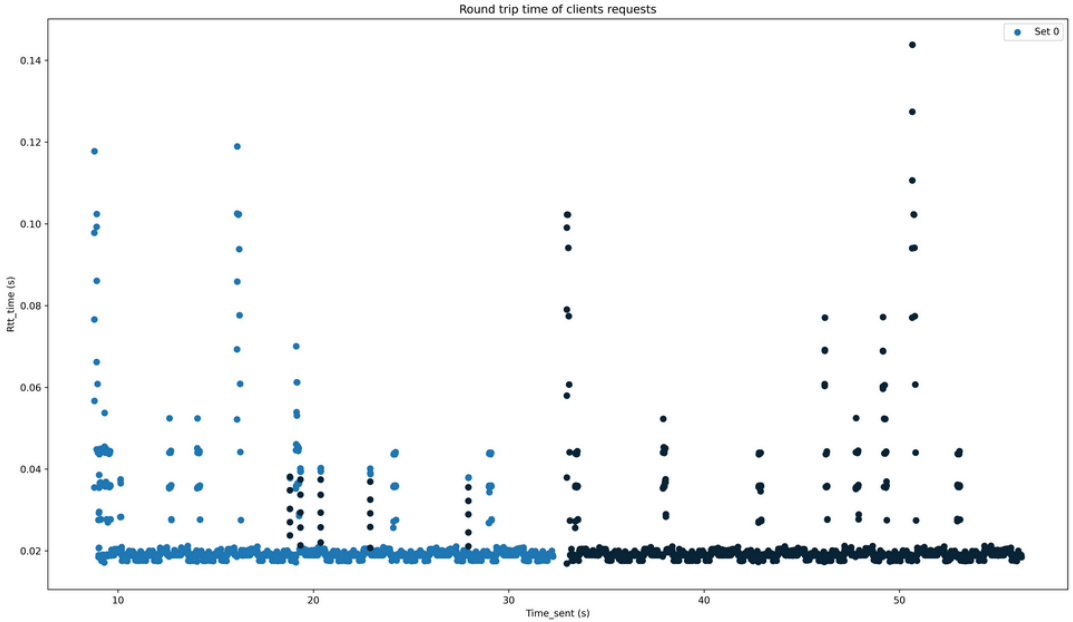
The Server Metric

- At the beginning of the execution, run a benchmark to get the maximum bandwidth possible
- Partitioning the execution into smaller time windows enables us to determine, for the current window, the bandwidth of each set in the preceding window

Algorithm 1 Server algorithm to accept or deny I/O request

$r \leftarrow I/O\ request$
$used \leftarrow get\_sets\_usage()$
$set\_current\_bandwidth \leftarrow get\_set\_bandwidth(r.set\_id)$
$set\_expected\_bandwidth \leftarrow bandwidth\_table(used,\ r.set\_id)/100 * server\_max\_bandwidth$
Return $set\_current\_bandwidth < set\_expected\_bandwidth$

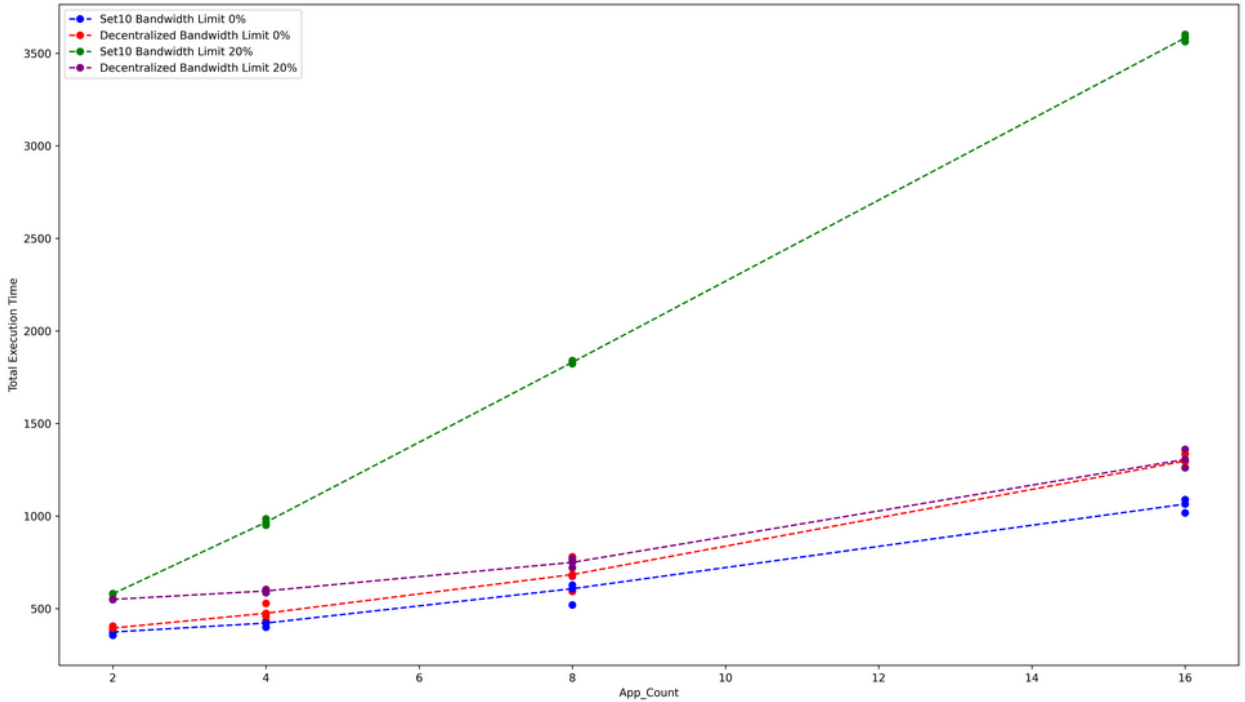
Exclusive characteristic



I/O requests of 2 applications in the same set using 100% of the bandwidth in the decentralized implementation

Results compared to original implementation

- Original implementation displays the expected linear scaling
- Decentralized version has some overhead when the applications bandwidth aren't limited
- When applications are unable to achieve maximum bandwidth, the decentralized version effectively manages the situation



05

Future Work

Future Work

- Implementing it inside a real Parallel File System like BeeGFS
- Improving the new method (lowering overhead to match original implementation, everyone asking IO at the same time)
- Server bandwidth should be updated dynamicly throughout the execution because benchmark bad

- Making the IO-TestBed more generic to allow IO-Sets implentations as dynamic libraries

Thank you for listening !

Any questions ?