Optimization of Data Transfer for 100 Gb/s Ethernet

AUGUST 2018

AUTHOR:
Massimiliano Galli

CERN EP-LCB

SUPERVISORS(S):
Niko Neufeld
Tommaso Colombo

CERN openlab

Abstract

In 2019 the LHCb experiment will go through an important upgrade, that will improve performance in many fields. One oh these fields is the DAQ system: it consists of a big flow of data that comes from the network and reaches a set of disks, where data are stored. This flow of data needs to be placed on the hardware in order to optimize the performance: high bandwidth and low CPU utilization are two key ingredients to achieve this goal. The first part of this project is about this: we realize a very basic test setup, consisting of two servers connected through a 100 Gbit/s Ethernet Interface and we optimize the data transfer in terms of bandwidth and CPU utilization.
In the second part a set of disks is connected to one of the two servers, in order to simulate a typical storage system. After optimizing the data transfer, a series of measurements of bandwidth as a function of the packet size is performed: the reason for this is to study the behaviour of the Data Direct Input/Output (DDIO) technology, that allows data packets of certain sizes to go directly from network to L3 cache, without touching the main memory; this should, in principle, enhance the performance in some cases.

Contents

1. The Test Setup

The first thing we did was realize a test setup that has both a network and a storage system part. A reproduction of the setup is shown in Fig. 1.1: it consists of two servers, each one having 40 CPUs distributed on two NUMA nodes, connected through a 100 Gbit/s Ethernet Interface; one of the servers is also connected to 96 disks, arranged in two parts controlled by two different controllers. The maximum allowed transfer rates are shown in Fig. 1.1. The following are the technical specifications of the different parts:

- CPUs : Intel Xeon CPU E5-2630 v4 2.20GHz
- NICs : Mellanox Technologies MT27700 Family [ConnectX-4]
- Disk controllers: LSI Logic / Symbios Logic SAS3008 PCI-Express Fusion-MPT SAS-3
- Disks : TOSHIBA MG04ACA6 FS3B

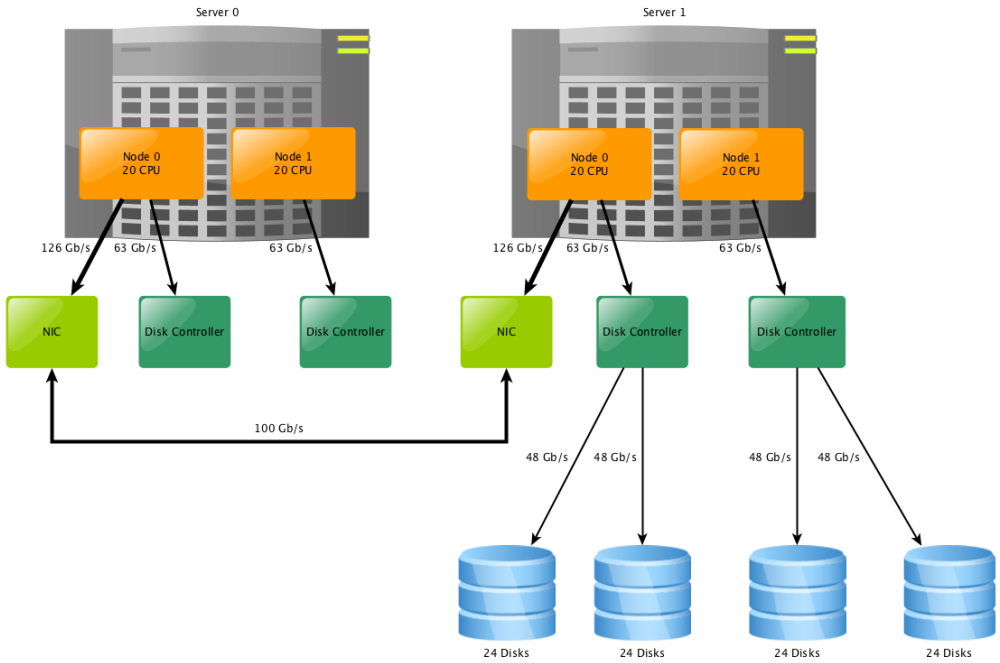

Figure 1.1: Reproduction of the test setup used in the project.

2. Network Optimization

2.1 Measures in the Default Configuration

In order to find which parts needed to be tuned to encrease performance, we first performed some measurements with the servers in their default configuration. The results of the measurements of bandwidth and CPU utilization are shown in Figs. 2.1 and 2.2: since both the numbers of parallel streams used seem to have side effects (4 streams do not reach the maximum bandwidth, while 12 streams imply a high CPU utilization), the following step was to configure the machines in order to reach the maximum bandwidth whit just 4 streams.

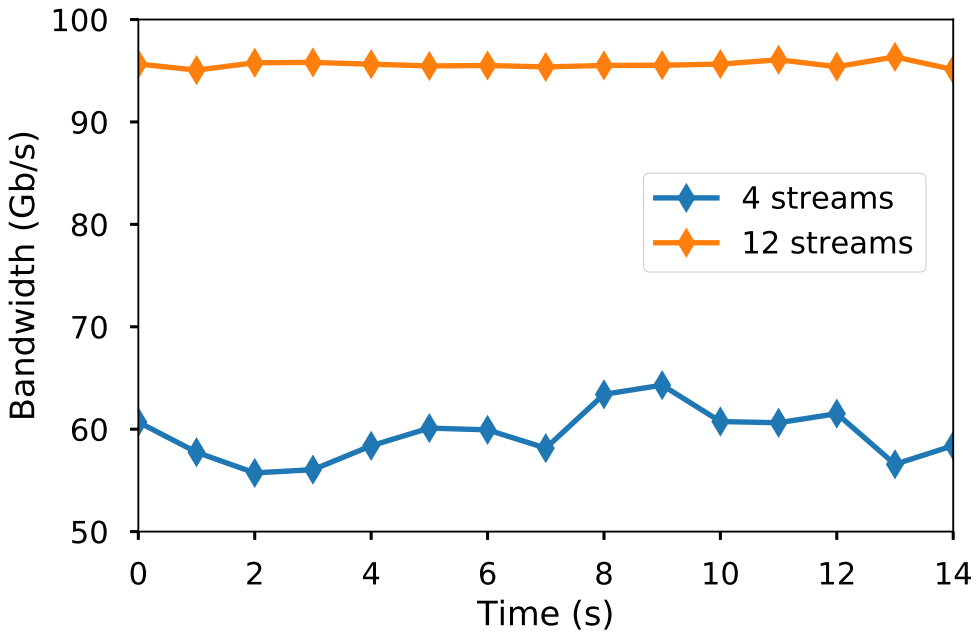

Figure 2.1: Measurements of bandwidth with the two machines in their default configuration; the orange line shows the level achieved using 12 parallel streams of data, the blue line the level achieved using only 4 streams of data.

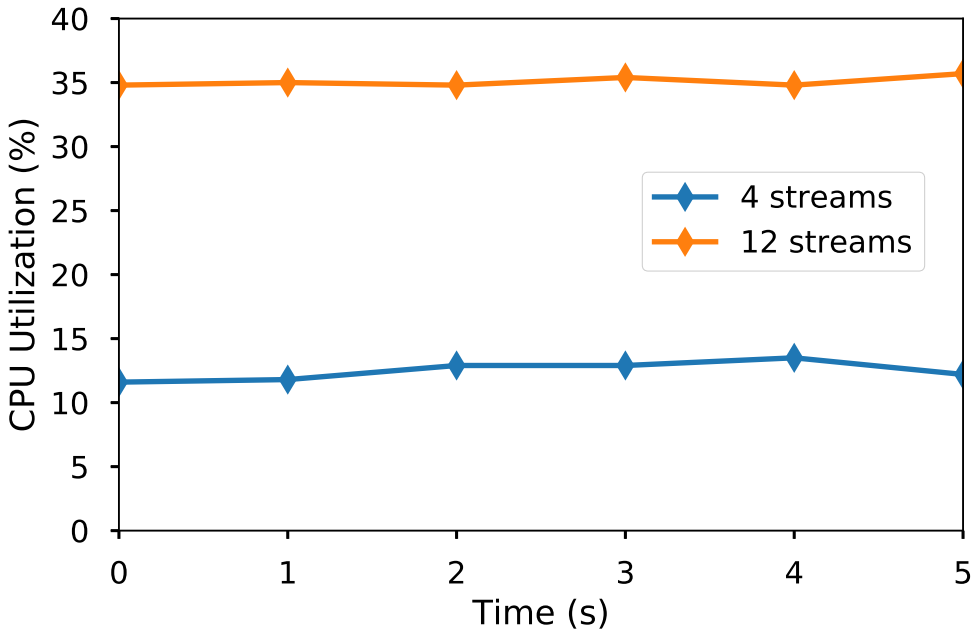

Figure 2.2: Measurements of CPU utilization with the two machines in their default configuration; the orange line shows the level achieved using 12 parallel streams of data, the blue line the level achieved using only 4 streams of data.

2.2 Tuning Procedure

Here we summarize the main steps of the procedure followed to encrease the performance with four parallel streams of data:

Disable IRQbalance IRQbalance is a command line tool that distributes hardware interrupts across processors to improve system performance. It runs by default, but it needs to be disabled in order to make a customized tuning.

Reassign CPU-IRQ affinities By modifying a series of scripts, it is possible to make different CPUs handle different hardware interrupts. In this specific case, we need to handle the interrupts coming from the NIC: we assign them to the first four CPUs in the NUMA node connected to the NIC, in order to make the data flow travel the shortest possible path.

Assign port-queue priority NICs have ring buffers, called RX (receive) and TX (transmit), that are used to store incoming packets until they can be processed by the device driver. Using Ethtool, it is possible to assign data packets coming from a specific port to a specific ring buffer: in this way the packets are not spread through all the RXs. In this specific case, we assign the packets coming the ports were the four streams are placed to the first four ring buffers.

Application pinning The last operation to do is forcing the benchmark tool (Iperf) to be executed on a set of CPUs which is not the one we assigned the IRQs to. In this specific case, we bound Iperf to the second four CPUs in the NUMA node connected to the NIC.

2.3 Results

The measurements of bandwidth and CPU utilization after the tuning, compared with the results obtained in the default configuration, are shown in Figs. 2.3 and 2.4. On both sides they are very good, since the bandwidth is almost the maximum allowed and the CPU utilization is lower than the one reached by the four streams in their default configuration.

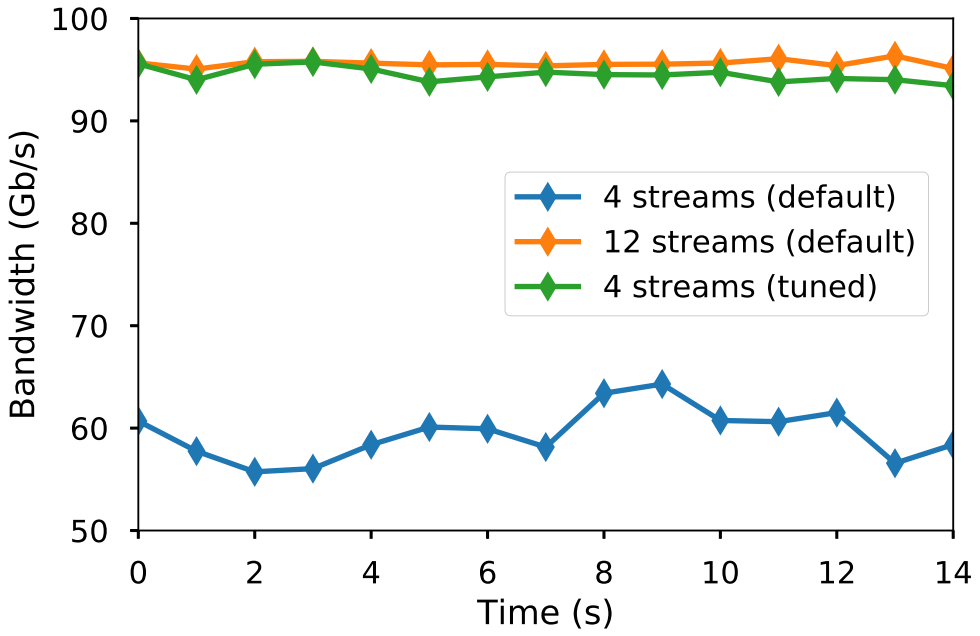

Figure 2.3: Measurements of bandwidth before and after the tuning. The green line is the level reached with four streams after the tuning, the orange and blue lines are the same shown in Fig. 2.1

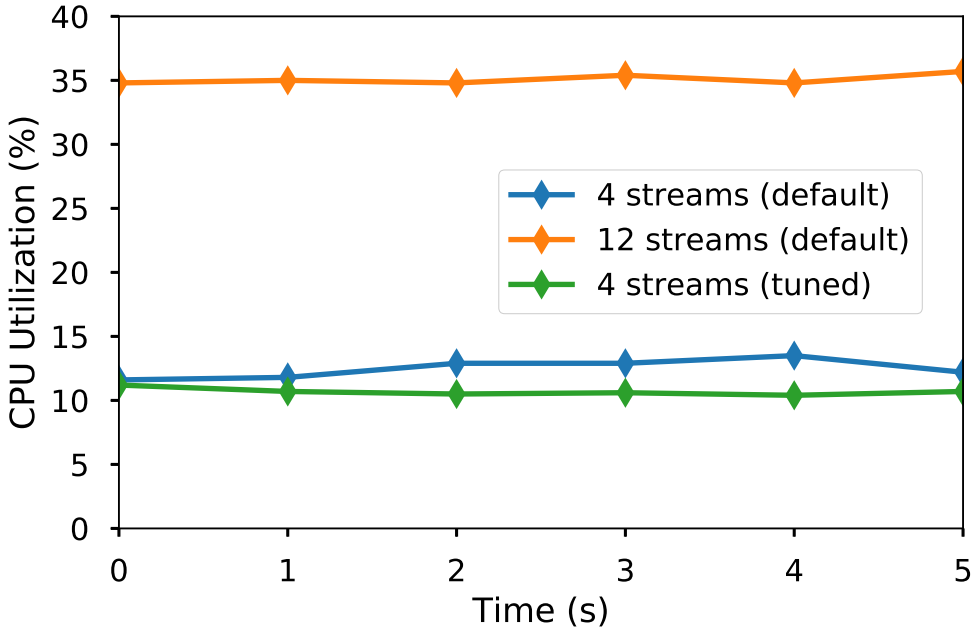

Figure 2.4: Measurements of CPU utilization before and after the tuning. The green line is the level reached with four streams after the tuning, the orange and blue lines are the same shown in Fig. 2.2

3. Storage System and DDIO analysis

In the second part of the project we wrote two programs able to read and write from the disks using 96 streams of data (one for each disk). We will call them ”read program” and ”write program”, depending on the action that they perform. By looking at the curves of the bandwidth as a function of the buffer size, we should also be able to say if and under which circumstances DDIO encreases the performance.

3.1 Read

In the read program a server is executed on the Server 0 (see Fig. 1.1) while 96 clients are executed on the Server 1. Every stream reads from a disk and sends the content to Server 0.
In this case, a high efficiency in terms of throughput is achieved by simply using the function `sendfile()` instead of the combination of `read()` and `write()`, since the user space is not touched.

3.2 Write

In this case we still have one server on 0 and 96 clients on 1, but now the server continuously sends data to every client, that instead receives and writes to the disk.
In this case, a high efficiency in terms of throughput can be achieved by using asynchronous I/O operations. Since we could not be able to finish the program on time, the measurements presented here were performed with simple `read()` and `write()` operations.

3.3 Results

In this section we show the measurements of bandwidth as function of time and buffer size, for both the read and write programs.

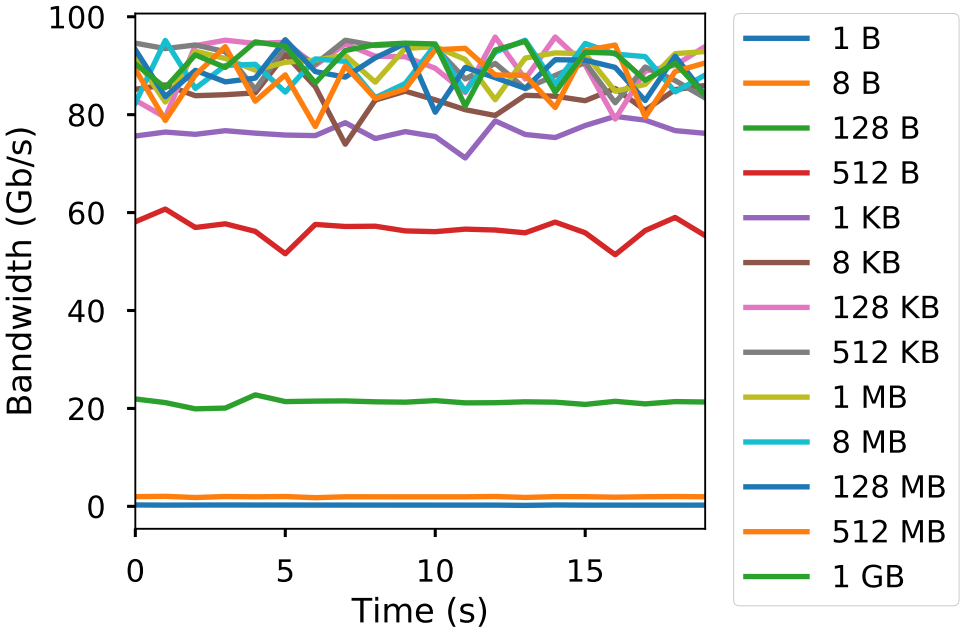

Figure 3.1: Bandwidth as function of time for different buffer sizes going from 1 B to 1 GB in the read program.

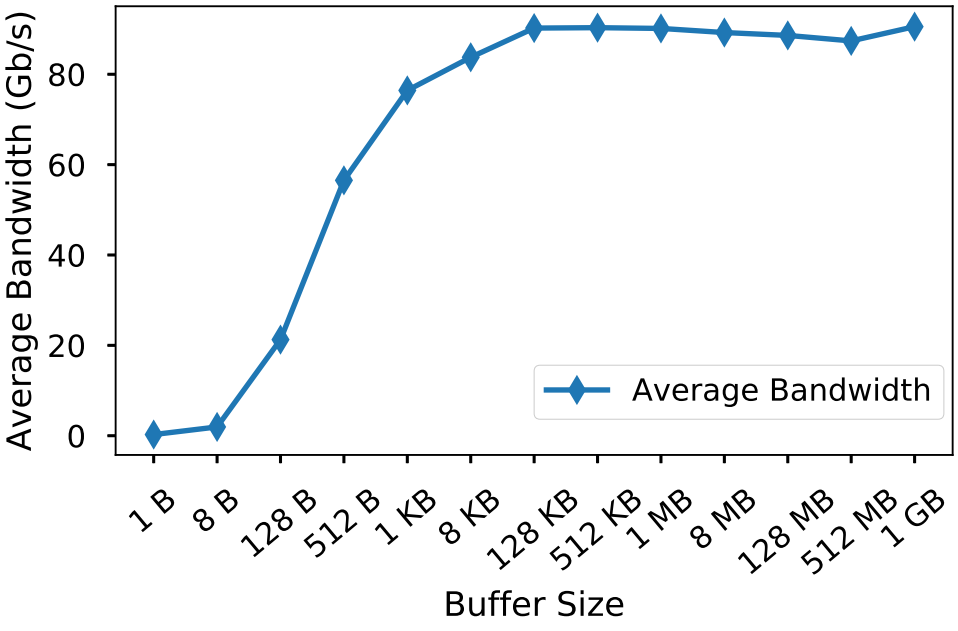

Figure 3.2: Bandwidth as function of the buffer size in the read program.

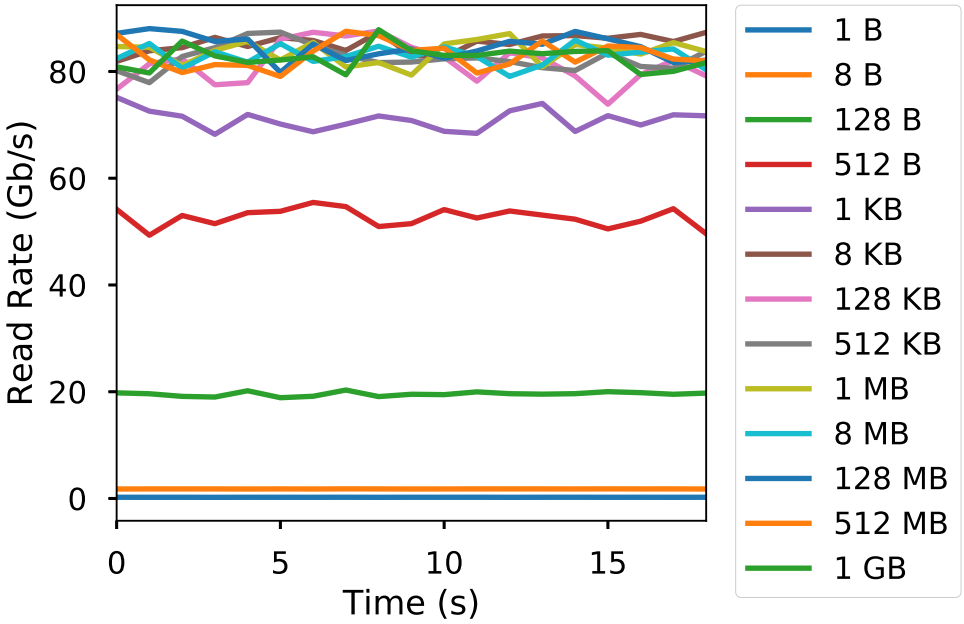

Figure 3.3: Read rate as function of time for different buffer sizes going from 1 B to 1 GB.

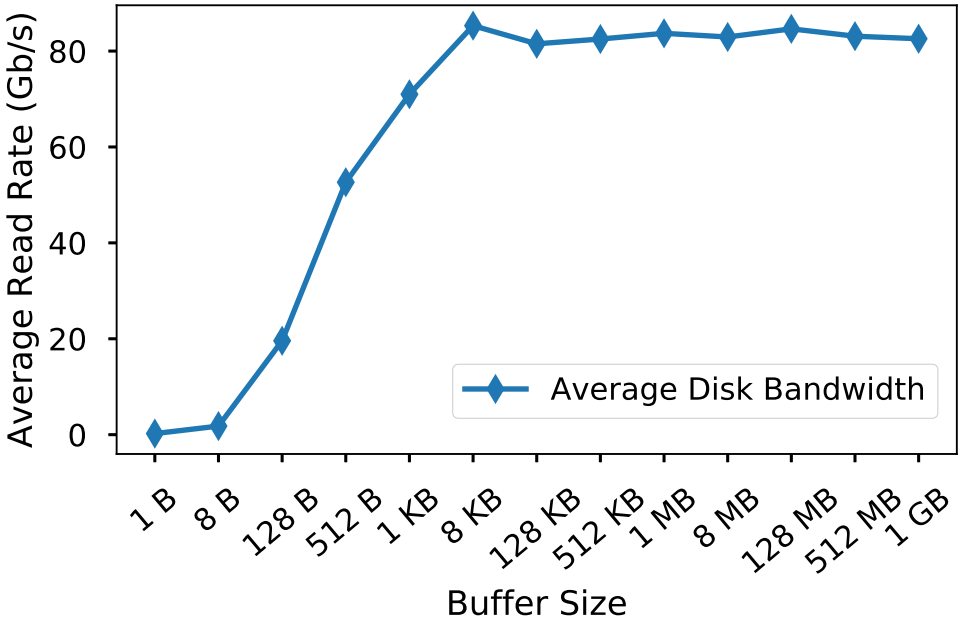

Figure 3.4: Read rate as function of the buffer size.

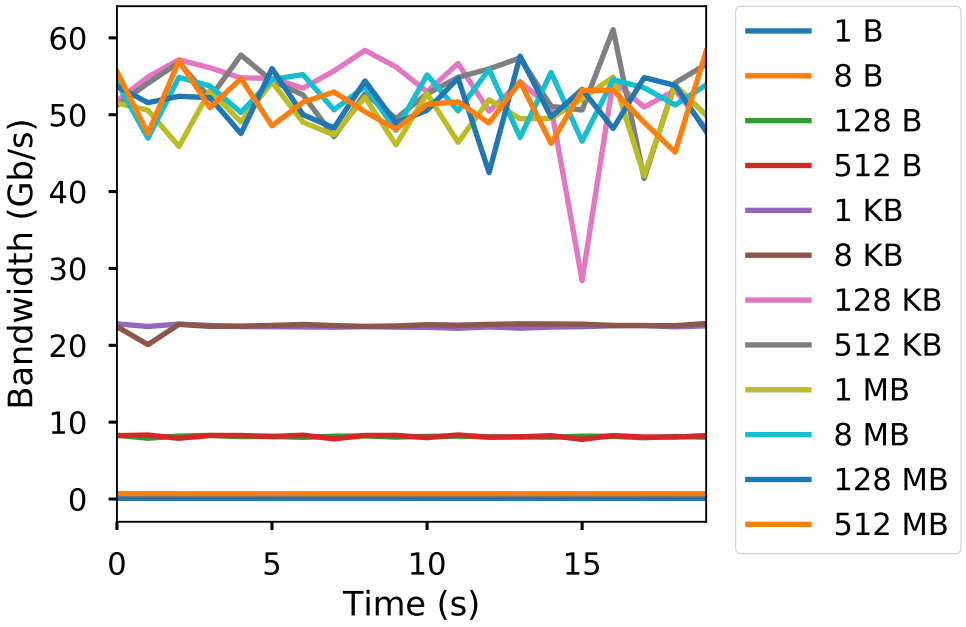

Figure 3.5: Bandwidth as function of time for different buffer sizes going from 1 B to 512 MB in the write program.

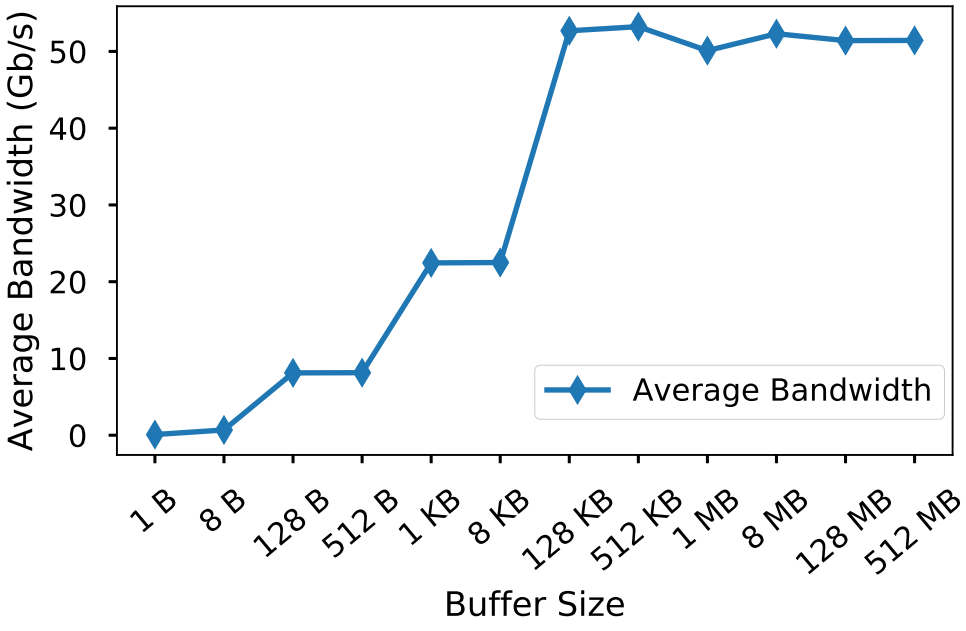

Figure 3.6: Bandwidth as function of the buffer size in the write program.

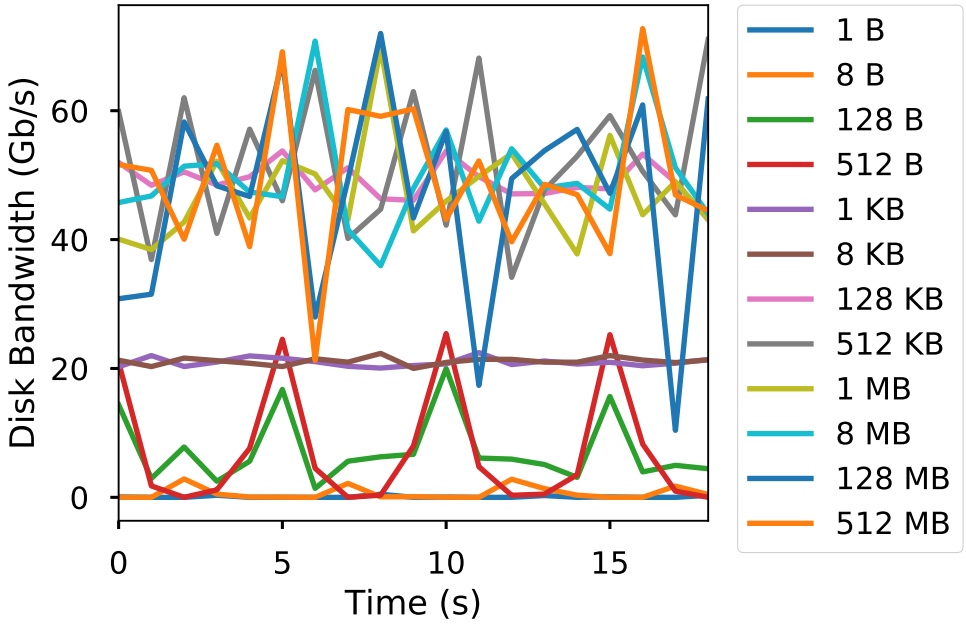

Figure 3.7: Write rate as function of time for different buffer sizes going from 1 B to 512 MB.

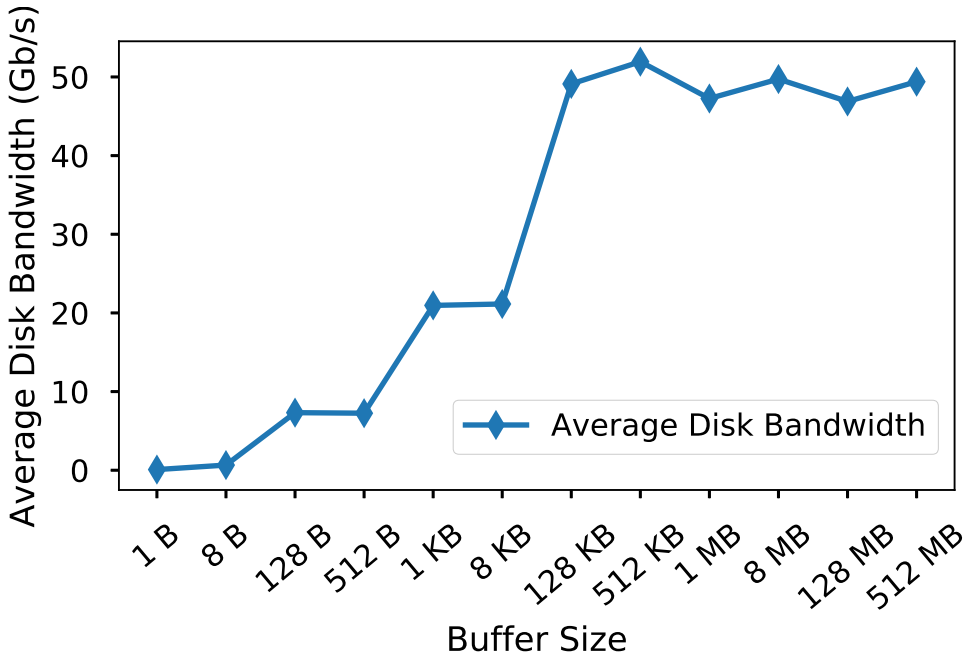

Figure 3.8: Write rate as function of the buffer size.

4. Conclusions

This project was divided into two main parts: in the first one we developed a way to optimize data transfer between two servers, using a low number of parallel streams and decreasing the amount of CPU utilization.
In the second part we performed some measurements of bandwidth and read/write rates as a function of sent package sizes. Since after a size of approximately 128 KB the function becomes almost flat, we can assert that the improvement expected from the DDIO technology was not observed.

Bibliography

[Bainbridge(2015)] Jamie Bainbridge. Red hat enterprise linux network performance tuning guide, 2015. URL https://access.redhat.com/sites/default/files/attachments/20150325_network_performance_tuning.pdf.

[Intel(2015)] Intel. Intel data direct i/o technology overview, 2015. URL https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/data-direct-i-o-technology-overview-paper.pdf.

[Jorgensen(2016)] Beej Jorgensen. Beej's guide to network programming, 2016. URL http://beej.us/guide/bgnet/pdf/bgnet_USLetter.pdf.

[Linux(2016)] Ref Hat Enterprise Linux. Deployment guide, 2016. URL https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch-disk-storage.