Benchmarking of OSCIED

17.12.13

1 Introduction

1.1 Goals

In order to evaluate the OSCIED platform, we were asked to perform a complete benchmarking by the means of different use cases. Our goal was to measure values such as time of execution, costs or impact of topology of deployment and to reason about measured results to provide conclusion on software refinement and deployment advises.

Out of our benchmarking, we also wanted to be able to compare the running of OSCIED in a Cloud environment, compared to a deployment on private servers. Results of the comparison would enable us to identify the overhead involved by using the Cloud.

1.2 Scope

We focused our study on the transcoding workflow. It consists of sending media to OSCIED, schedule transcoding tasks and retrieve transformed medias. Publication performances were out of the scope of our tests, since we did not have reliable clients to test it.

2 Use cases

In this section we present the different use cases we used in the frame of our benchmarking, together with the profile of transcoding we used.

2.1 Tasks sets

We have performed our measurements over the following type transcoding task:

1. Convert a MXF file into a high quality H.264 MP4, targeting mobile device such as tablets

   Input Extremes_CHSRF-Lausanne_Mens_200m-50368e4c43ca3.mxf

   Profile 480p for tablets (480p/25 at 1Mbps in main profile with low-latency)

   Encoder ffmpeg, with options *-r 25 -c:v libx264 -profile:v main -preset slow -tune zerolatency -pix_fmt yuv420p -strict experimental -b:v 1000k -maxrate 1000k -bufsize 2000k -vf scale='trunc(oh*a/2)*2:min(480,iw)' -acodec aac -ac 2 -ar 44100 -ab 96k -f mp4*

We call task set the scheduling of a run of multiple transcoding tasks of the same type. For the purpose of our study, we sent a task set of 50 tasks to the plateform for each use case.

2.2 scenari of deployment

We ran two scenari of deployment, mainly varying the number of transform units, and the transform+storage units topology. Namely, we used the following deployments:

many small capacity VMs without merging where we would deploy OSCIED on 11 VMs, namely 1 juju unit, 1 orchestrator unit, 4 storage units and 5 transform units, with up to 4 concurrent transcoding workers per unit;

few high capacity VMs with merging where we would deploy OSCIED on 4 VMs, namely 1 juju unit, 1 orchestrator unit and 2 storage + transform units, with up to 8 concurrent transcoding workers per unit;

bare metal installation where we would deploy OSCIED on 3 servers, namely 1 juju + storage + transform unit, 1 orchestrator + storage + transform unit and 1 storage + transform unit, with up to 6 concurrent transcoding workers per unit.

We based the choice of these topologies on the intuition that an interesting difference would emerge between the merging and non merging approach. Indeed, in the first scenario, each VM lived separatedly from the others, thus heavily relying on network communications; in the second scenario, network communication are replaced by local data transferring, at the expense of computation resource sharing. To keep the two deployments equivalent in term of total computation power, we relied on the Amazon Elastic Compute Unit (ECU), an abstraction measurement of computer resources; it corresponds to an early 2006 1.7 GHz Xeon processor. For the first scenario, we used one instance of type c1.medium per OSCIED unit, i.e. a total of ten. For the second scenario, we used one instance of type c1.medium for the orgchestrator, and 2 instances of type c1.xlarge for the storage + transform units. We scaled our choices to correspond to the computation power of our bare metal installation, which approximatively equals to 42. The characteristics of the instance profiles we used are given in table 2.1. Please note that Amazon doesn't give further details on the terms *high* and *moderate*, regarding the network performances; instead, they advice the user to choose such characteristics with respect to their own needs evalutation.

Profile	c1.medium	c1.xlarge
Architecture	64 bits	64 bits
vCPU	2	8
ECU	5	20
Memory	1.7 GB	7 GB
Storage	1 x 350 GB	4 x 420 GB
Network	moderate	high

Tab. 2.1: Amazon instance type specifications

If we compare the ECU involved in all scenari, we would retrieve quite acceptable match:

$$\begin{aligned} \text{scenario 1: } & 10[vm] \times 5[ecu] & = \mathbf{50}[\mathbf{ecu}] \\ \text{scenario 2: } & 1[vm] \times 5[ecu] + 2[vm] \times 20[ecu] & = \mathbf{45}[\mathbf{ecu}] \\ \text{bare metal: } & 2[server] \times 2[cpu] \times 7[ECU] & = \mathbf{42}[\mathbf{ecu}] \end{aligned}$$

Please note that we ignored the juju unit in our computations for both cloud scenari, since it does not impact our benchmarking after the system has been deployed. To preserve an equal number of transcoding workers, we adapted the maximum concurrency of the transcoding workers in each instance. In other words, we allowed more workers to run simultaneously on the same instance on the second scenario, to compensate the fewer number of transconding units.

3 Results

In this section we present the results of our benchmarking. We will first go through the values we measured for all scenari, as presented above. Then we will interpret the meaning of these values, possibly confirming them with some additional runs. We distinguish two different kind of measurements. On one hand we collected some data on the instances activity, i.e. the actual machines running OSCIED, about their CPU, memory, disk, etc. On the other hand, we kept track of some information about OSCIED, i.e. the tasks and units status. Over a set of metrics we will introduce later, our goal was to compare the data collected from instances activity with the data from the OSCIED, using the latter to explain the former.

We used the following metrics to measure instances activities; please note that all these values are measured as a function of the time:

- **CPU consumption** which is the time spent by the CPU in user and system modes;
- **Virtual memory usage** which basically is the amount of consumed virtual memory;
- **Swap memory usage** which basically is the amount of consumed swap memory;
- **disk counters** which are a collection of counters about the disk activity, such as the amount of data read and written, or the time spent reading or writting data;

Scenario	**one**	**two**	**bare metal**
JuJu bootstrapping	about 2m	about 2m	-
OSCIED deployment	about 10m	about 10m	about 1h
Tasks execution	1h 10m 47s	57m 31s	20m 25s

Tab. 3.1: Amazon instance type specifications

Scenario	**one**	**two**	**bare metal**
Average task time	-	11m 1s	4m 18s
Minimum task time	-	3m 29s	2m 29s
Maximum task time	-	15m 8s	5m 17s
Average simultaneous tasks	-	10	12

Tab. 3.2: Amazon instance type specifications

network counters which are a collection of counters about the network activity, such as the amount of data received and sent.

From OSCIED, we got the status of the tasks and units in function of the time. Obviously, we could explicitly get other interesting values such as the plateform bootstrapping time of the system, i.e. the time required for OSCIED to be ready to accept tasks, and the overall transcoding time.

3.1 Performances evalutation

We decomposed each scenario in the same three steps and timed each of these steps; the reason behind this decomposing was to prevent us from redeploying everything from scratch in the event of a problem. These three steps are 1. the bootstrapping of the JuJu plateform; 2. the deployment of OSCIED plateform; 3. the execution of the tasks.

Both cloud scenari shows equal times for the first two steps, i.e. an average of 2 minutes for the bootstrapping of JuJu, and an average of 10 minutes for the deployment of OSCIED. We neglect the differences in this report because we think they are not significant enough to be relevant of some characteristics of our different topologies. In the case of the bare metal, the complete installation was about an hour, because it includes the fresh installation of Ubuntu Server on all machines. However, we measured significant difference between the time took to execute the tasks. The first scenario took 1h 10m 47s while the second performed in only 57m 31s, i.e. 13m earlier. The bare metal deployment was by far the quickest, since it took only 20m and 25s. Table 3.2 summarize these values.

We computed that the average transcoding time of a single task was about 20m in the first cloud scenario and only 11m in the second; the bare metal scenario could perform a task in 4 minutes and 18s in average. Moreover, if we look at the task status reporting, we remark that 14 tasks were simultaneously computed in the first cloud scenario, against only 9 in the second. In the bare metal scenario, an average of 12 simultaneous tasks were performed. Theses values are reported in table **??**. Unexpectingly, these results do not match the theorical number of transcoding worker, in every scenari. Investigating the possible lead that the logic of celery, the library responsible for the workers management, detects the consumption of vCPU by other processes and limits itself the concurrency, we ran the bare metal scenario with an increased concurrency of 12 per unit. New batch of results did not fully confirmed our assumption, but we did note that the number of concurrent processes increased nevertheless. Our conclusion on this observation is that it is preferable to overestimate the concurrency, because celery seems to consider that number as a maximum bound and not as a goal to reach.

In term of cost, a simple calculation gives us a total fee of \$ 1.71 for the first cloud scenario and \$ 1.25 in the second one. We based our calculations on the Amazon pricing.

$$\begin{aligned} \text{pofile cost} \times instances \times \text{overal time} &= \text{total fee} \\ 0.145[\$/h] \times 10 \times 1.18[h] &= 1.71[\$] && \text{first scenario} \\ (0.145[\$/h] + 0.58 \times 2) \times 0.96[h] &= 1.25[\$] && \text{second scenario} \end{aligned}$$

The fewer cost of the latter can be explained by, obviously, a shorter execution time, but also because it involves less instances. Interestingly enough, we can compare these cost values with the pricing of Amazon Elastic Transcoder, which would have cost about \$ 2.8 for an equivalent amount of transcoding tasks and a similar profile.

$$\begin{aligned} \text{tasks} \times \text{media duration} \times transcodingpricing &= \text{total fee} \\ 50 \times 3.73[m] \times 0.015[\$/m] &= 2.80[\$] \end{aligned}$$

Please note that in both cases, we neglect the cost of downloading back the transcoded media to our own computers.

Finally, an interesting comparison would be to compare the cost of the cloud utilization, compared to our are metal. We invite the reader to consult the survey carried out by Jeremy Edberg, who gave an estimation of the maintenance cost of a data center.

3.2 Bottlenecks identification

As said above, we measured different values about the instances activity, while performing the transcoding tasks. Intuitively, we would have thought that the network, or possibly the storage would be the bottleneck of the plateform but, surprisingly enough, we identificated that CPU load was the main problem. The following is a more detailed review of our results.

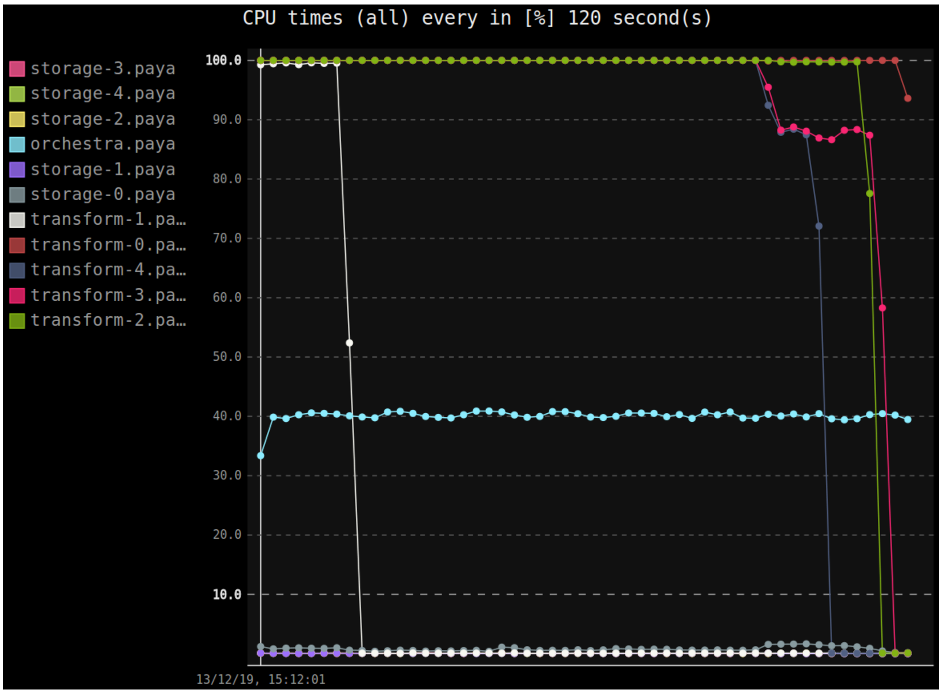

Fig. 3.1: CPU utilization over the time in first scenario

3.2.1 Processor

In both our cloud scenari, we remarked that the bottleneck was the CPU load of transcoding units, culimnating around 100% in the two use cases. However, in the bare metal deployment, units did not consumed all the CPU power. We should stress that we probably underestimated the computation power of our installation, leading to such diffences. To confirm this assumption we ran compared our results with a similar use case but with more concurrent transcoding workers. This time, the CPU utilization capped to 100%. Furthermore, if the total number of simultaneous tasks increased to 25, the total run was even slower than the former one. This convince us to point the CPU as a significant bottleneck of the plateform, which possibly overrules most of the perfomance issues. Figure 3.1 3.2 3.3 illustrate our results.

As we blended the orchestrator with the thranscoding units in the bare metal scenario, we could not identificate its own CPU consumption. However, in the two cloud scenari we remarked that orchestra consumed about 40% of its available power. This result is quite logical since both deployments used the same instance type for the orchestrator. We think that we underused the machine on which was located the orchestrator, as it will be confirmed in our review of virtual memory consumption.

Finally, cloud scenari also showed us CPU consumption of the storage units. As we can see on figure 3.1 3.2 3.3 , values were quite low and clearly showed an underutilization.

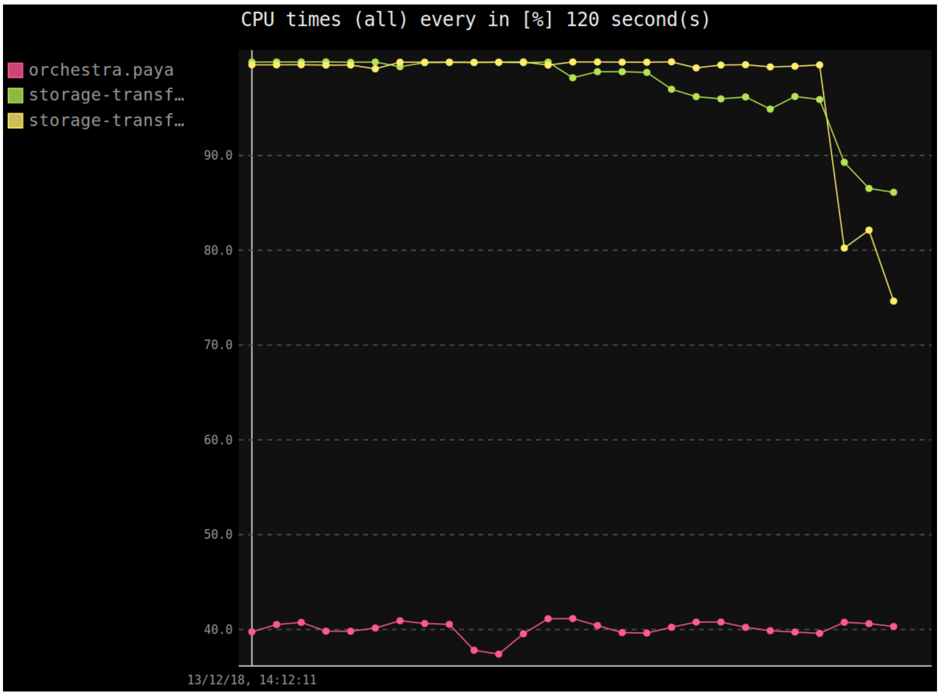

Fig. 3.2: CPU utilization over the time in second scenario

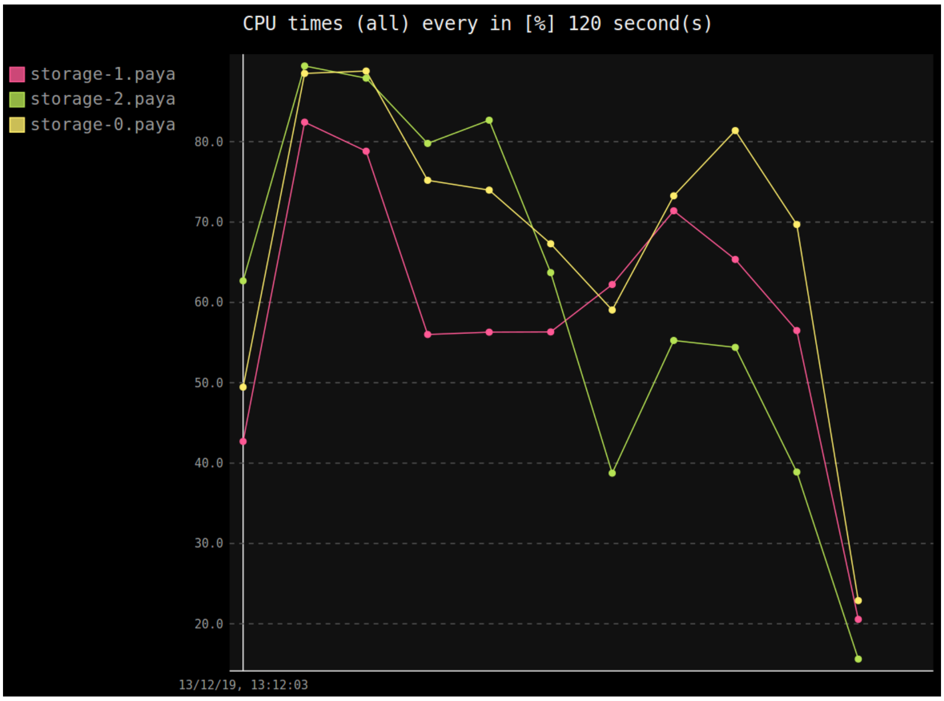

Fig. 3.3: CPU utilization over the time in bare metal

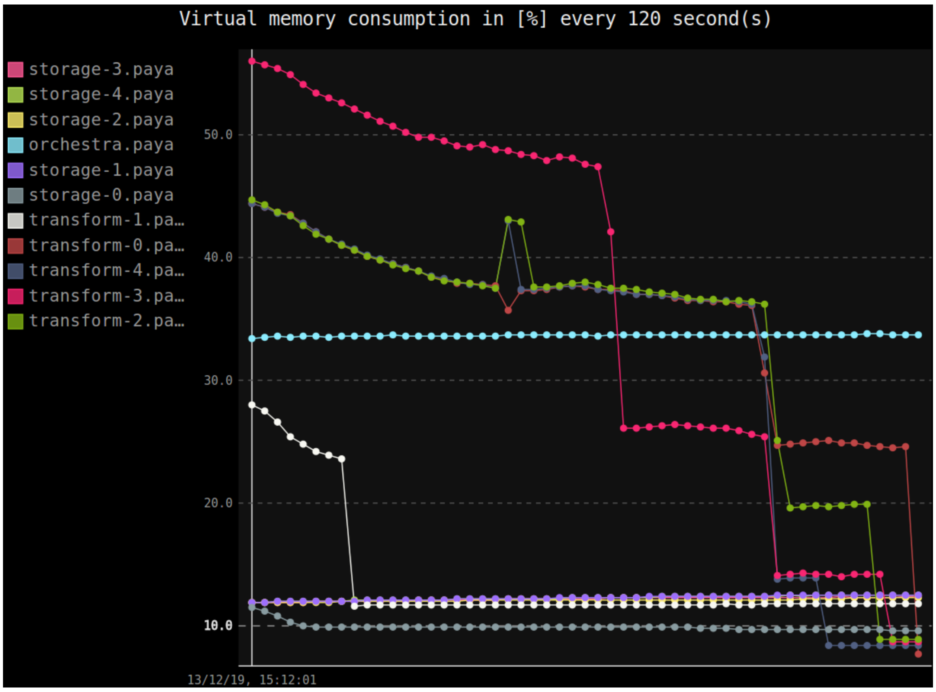

Fig. 3.4: Memory consumption over the time in first scenario

3.2.2 Virtual Memory

In all our scenari, we saw that the virtual memory consumption was not an issue, whatever the unit type. Orchestrator consumed between 30% and 35% of its available memory in both cloud scenari, which is quite expectable since it ran on the same instance type. We explain the additional memory consumption of the first scenario by the fact that it involved more units and thus more processing for the orchestrator. Values measured for storage units in the first scenario told us that memory was not an issue for this type of unit; it remained stable around 10% which convince us to neglect it in other scenari. Figure **??** depicts measured values.

Transcoding units showed the most interesting results. In all scenari we remarked that it decreased as the time went, which is explained by the rarefiction of task to perform. As we can see on figure 3.4 3.5 3.6, memory consumption evolved by huge steps. Actually, it corresponds to several tasks starting or beginning at the same time; indeed as tasks were homogeneous, it was likely that a complete batch of tasks finished at the samte time. This behavior is well observable in the second cloud scenario, were we can clearly trace the blocks of tasks.

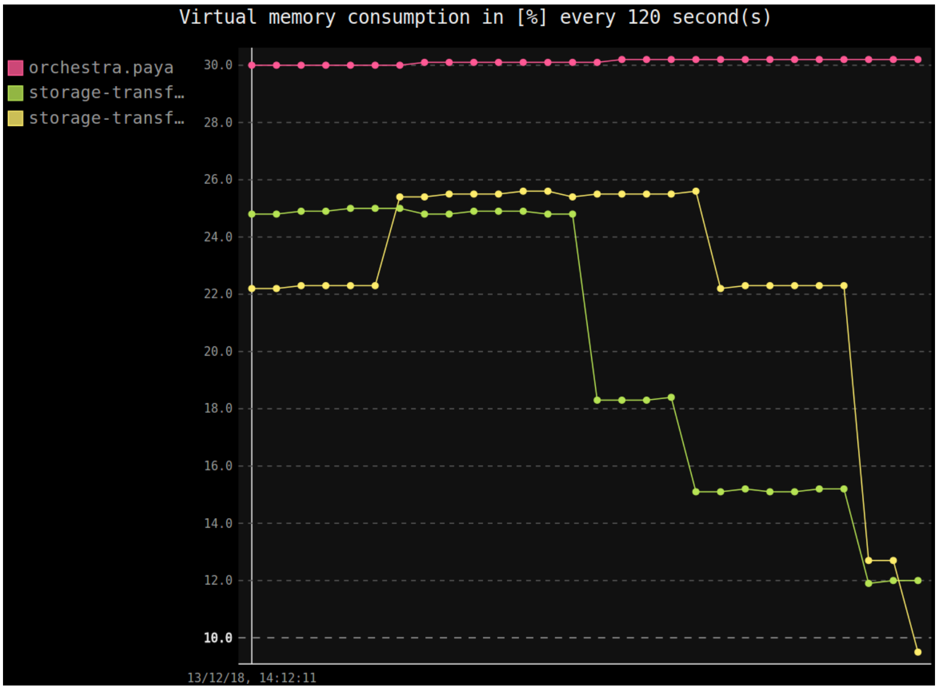

Fig. 3.5: Memory consumption over the time in second scenario

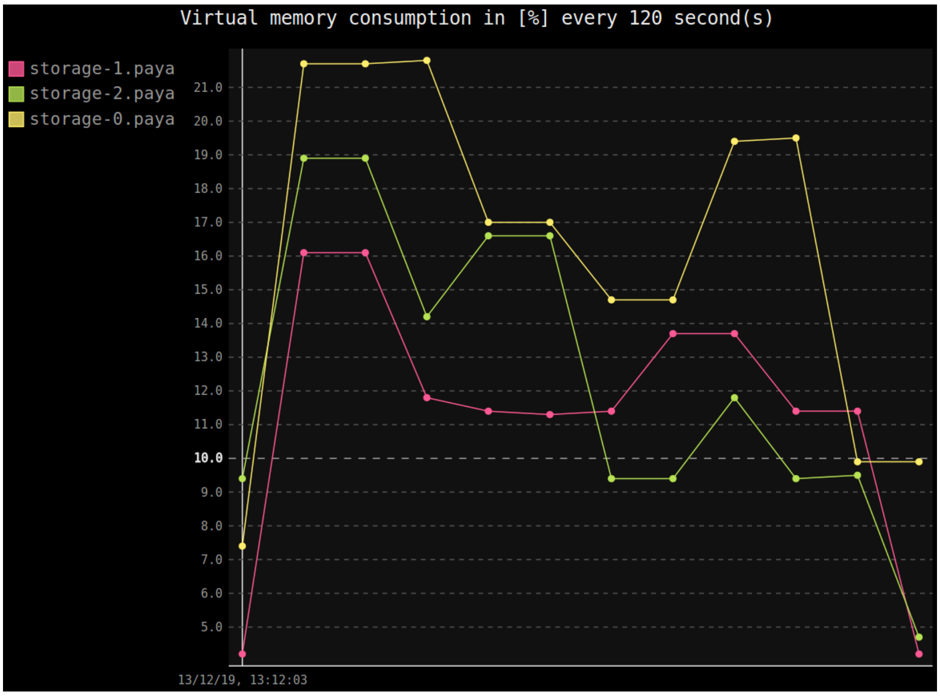

Fig. 3.6: Memory consumption over the time in bare metal

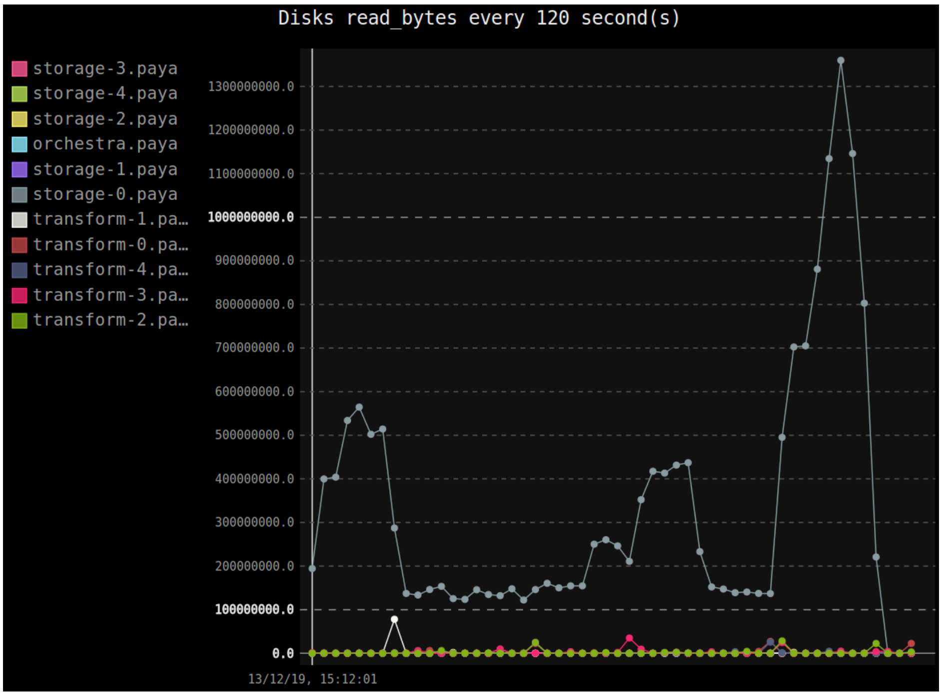

Fig. 3.7: Bytes read over the time in first scenario

3.2.3 Disk and network

We group these two metrics because they are quite related, as would lead the intuition.

A problem that arised in our scenari was induced by the fact that we had only one file to transcode. That means that even if we had several storage units, input media was actually always downloaded from the same machine. This side effect can be observed in charts reporting byted read over the time. However, in the bare metal screnario, we observed almost no reading activity. This is because we ran our test several times, to the extent that the input media was stored in the cache, and not in the hard drive anymore. Other units, storage or not, made almost no writing activity. We explain this by the fact that our plateform uses directly the storage mounts. Figure 3.7 3.8 3.9 depicts our results.

More interesting are the values of the network activity, regarding the received data. The first scenario exposed a very stable behaviour with only few pikes on the transcoding units when batches of new tasks were scheduled. Storage units were even not affected by any pikes, since every output media was equally distributed, thus producing no network congestion. On the second scenario, we remarked that the values of storage units were constantly increasing. This is because the deployment counted only two storage units, so leading to an increasing consumption of the network as more tasks were written. Figure 3.10 3.11 3.12 depicts our results.

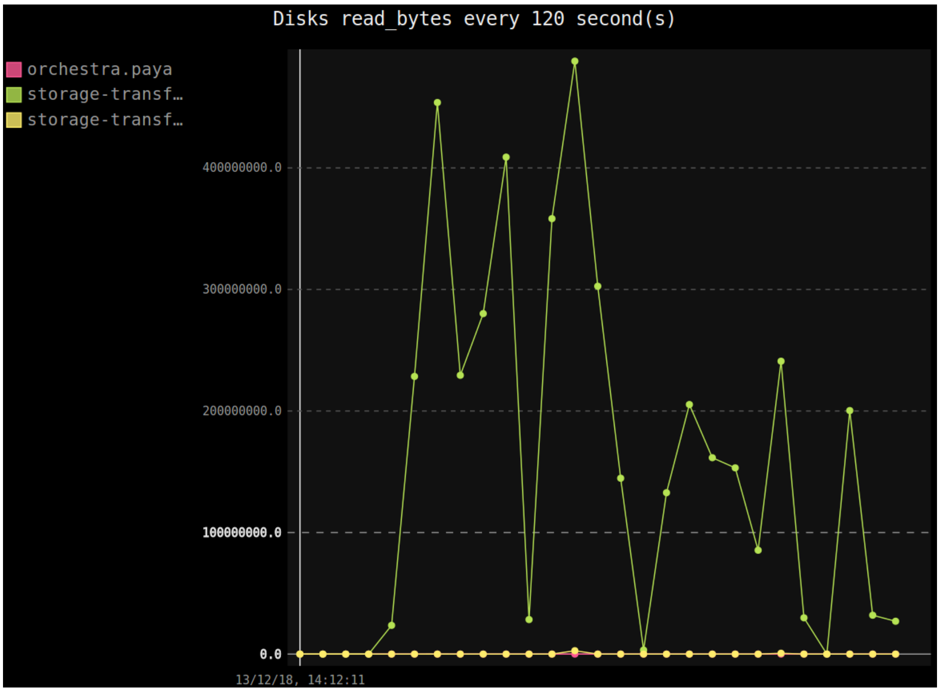

Fig. 3.8: Bytes read over the time in second scenario

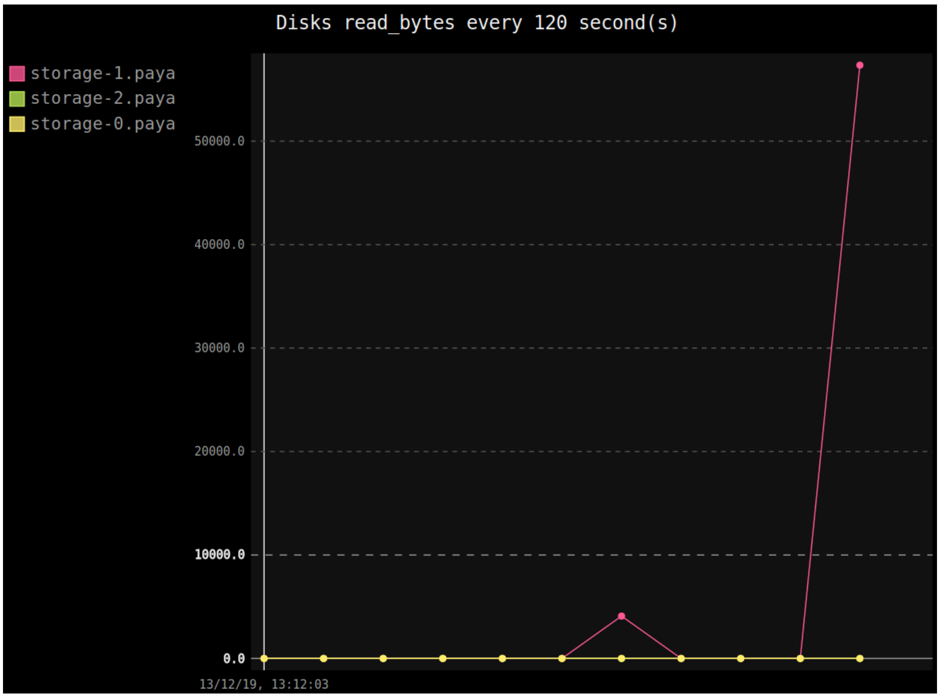

Fig. 3.9: Bytes read over the time in bare metal

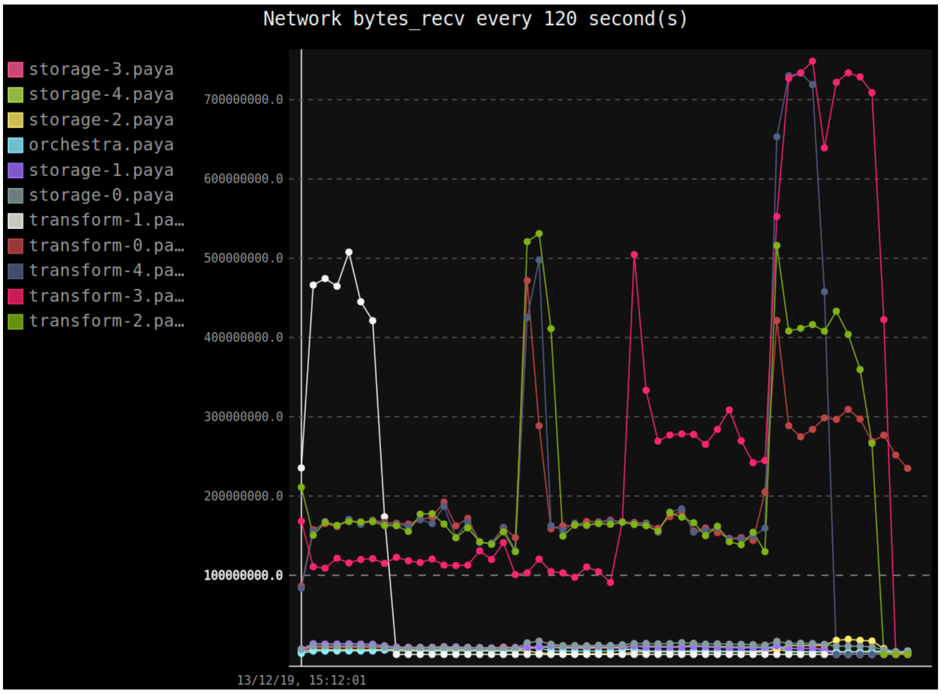

Fig. 3.10: Received bytes over the time in first scenario

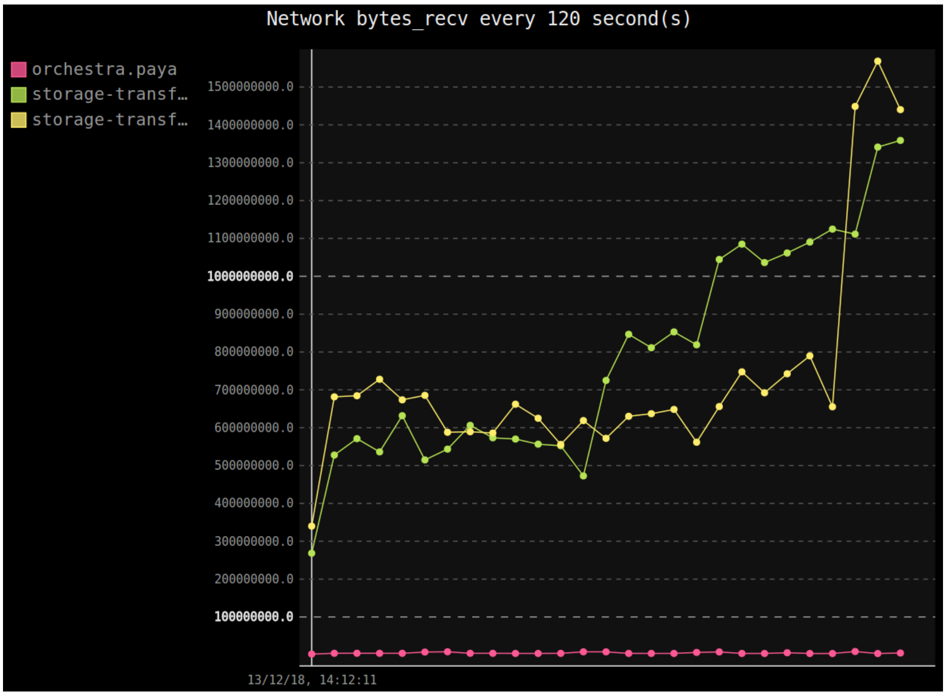

Fig. 3.11: Received bytes over the time in second scenario

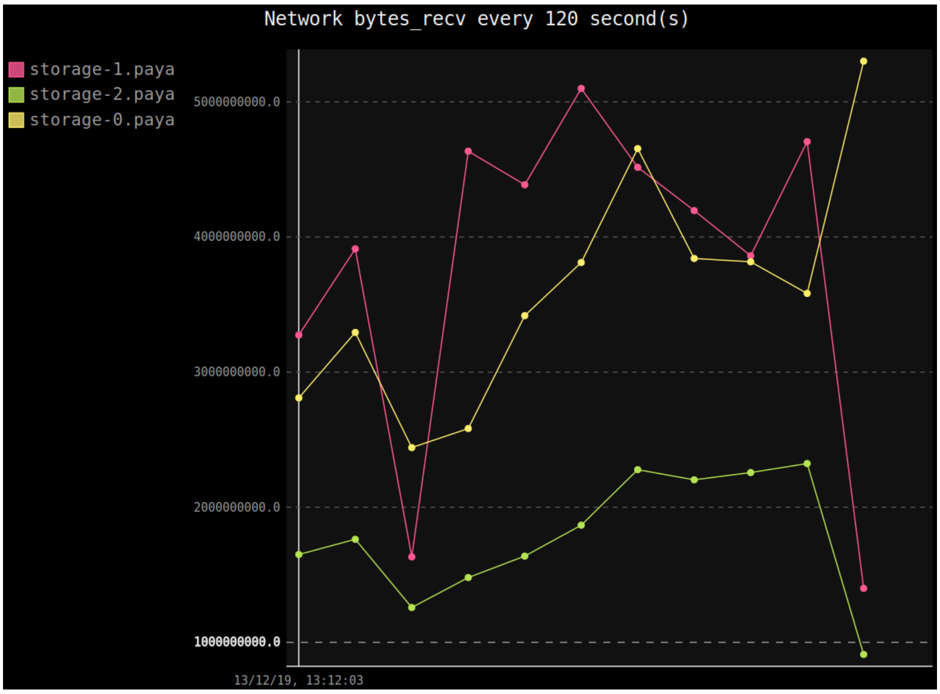

Fig. 3.12: Received bytes over the time in bare metal

Disk writing activities are quite similar in all scenari. As we explained above, output medias were equally distributed between storage units, thus explaining these results. In the second scenario, we noticed a sort of switch between the level of activity of the two storage units. We explain this behaviour by the load balencing system of Gluster FS. Figure 3.13 3.14 3.15 depicts our results.

Data sent over the network matches the observation we did earlier about the fact that only one storage unit could distribute the input file.

4 Conclusion and advices

As said in the results discussion, a surprising discover was the fact that network was not the bottleneck of the system, but CPU was. Other results were quite straightforward and followed our expectations.

We would recommand the deploy a plateform similar to what we did in the second scenario, with a more modest instance type for the orchestrator. Mixing storage units with transcoding units revealed to be a very interesting gain, especially because ffmpeg is very active in term of disk activity.

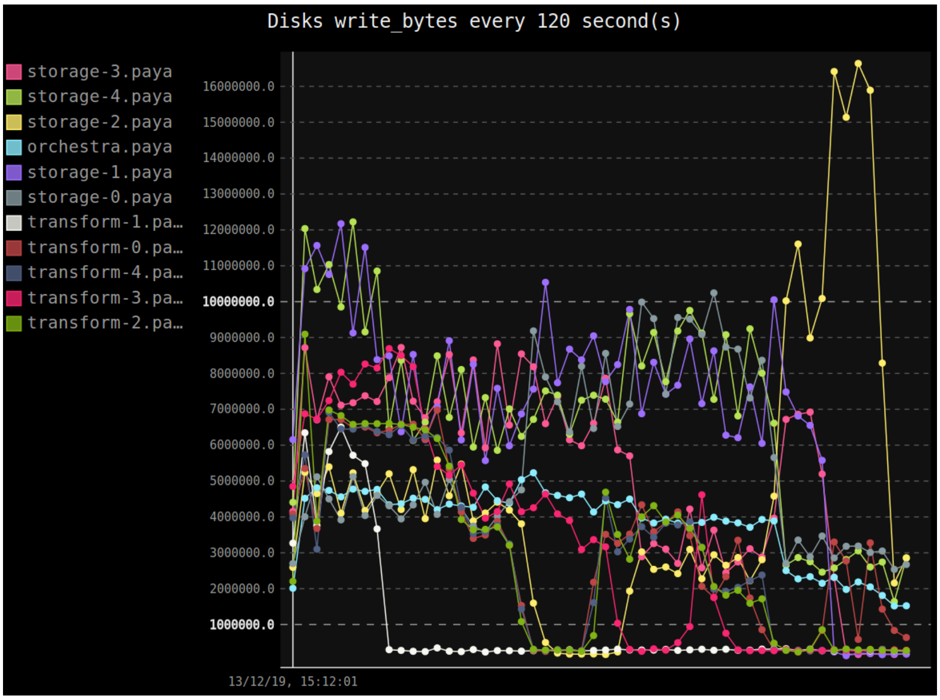

Fig. 3.13: Bytes written over the time in first scenario

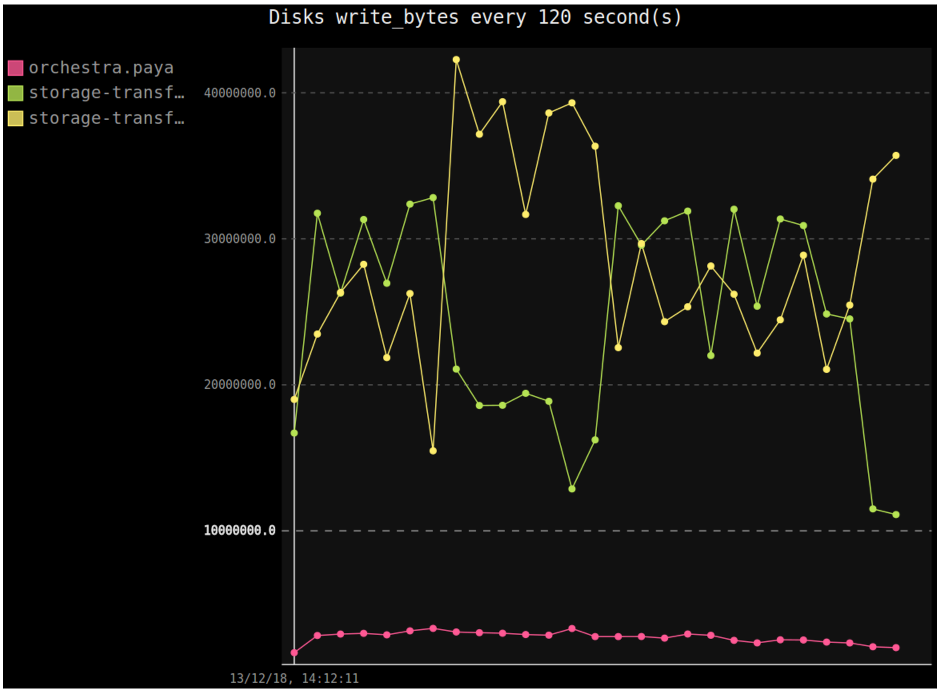

Fig. 3.14: Bytes written over the time in second scenario

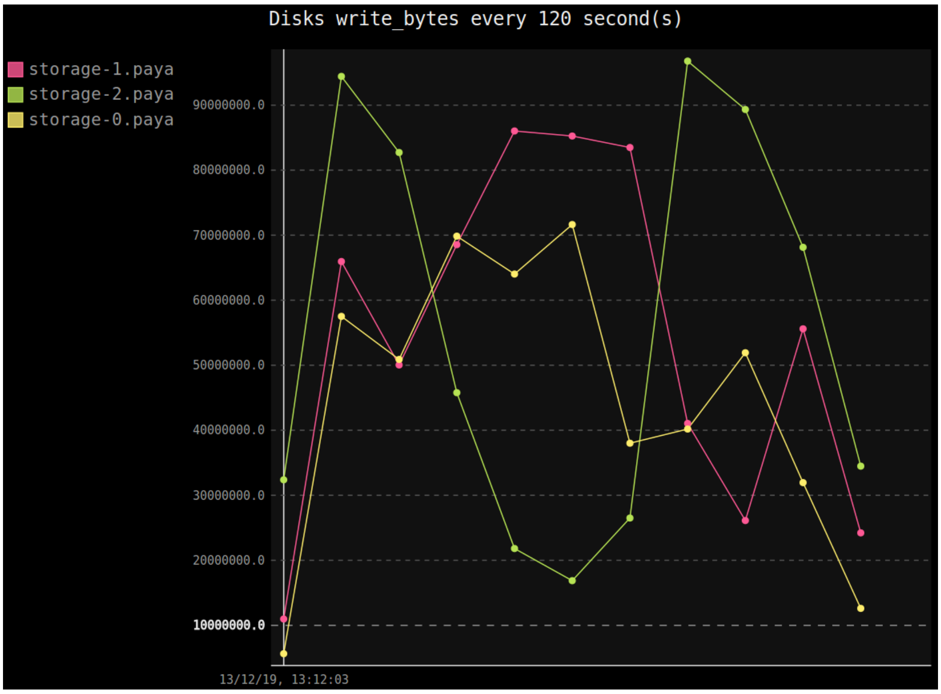

Fig. 3.15: Bytes written over the time in bare metal