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 take stock of several measures, 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.

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.

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 fot the orgchestrator, and 2 instances of type c1.xlarge for the storage+transform units. The characteristics of this profile 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 both scenari, we would retrieve an almost perfect match:

$$\begin{aligned} &\text{scenario 1: } 10[vm] \times 5[ecu] &&= \mathbf{50}[\mathbf{ecu}] \\ &\text{scenario 2: } 1[vm] \times 5[ecu] + 2[vm] \times 20[ecu] &&= \mathbf{45}[\mathbf{ecu}] \end{aligned}$$

Please note that we ignored the juju unit in our computations for both 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 both scenari, as presented above. Then we will interpret the meaning of these values, possibly confirming them with some additional smaller runs. We distinguish two different kind of measurements. On one hand we collected some data about 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 is 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;
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.

For the sake of conciseness and clarity, charts were not imported in this docuement, but left in the annexes. We will refer to the filename of the charts when we use them.

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 scenari shows equal times for the first two steps, i.e. an average of 2

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

Tab. 3.1: Amazon instance type specifications

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. However, we measured a more important 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. Table 3.1 summarize these values.

We computed that the average transcoding time of a single task was about 20m in the first scenario and about only 13m in the second. Moreover, if we look at the task status charts (s1-tasks-status.svg, s2-tasks-status.svg), we remark that 14 tasks were simultaneously computed in the first scenario, against only 9 in the second scenario. We cannot clearly explain this last result because both scenari have the same theoritical number of transcoding workers. Intuitively enough, one could have guessed that the actual number of vCPU was capping the measured concurrency, but the fact is that the second scenario, which exposes the unexpected results, does have enough vCPU to ensure such concurrency. A possible lead would be 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.

In term of cost, a simple calculation gives us a total fee of $ 1.50 for the first scenario and $ 1.05 in the second one. 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 $ 3.28 for an equivalent amount of transcoding tasks and a similar profile. Please note that in both cases, we neglect the cost of downloading back the transcoded media to our own computers.

3.2 Bottlenecks identification

As said above, we measured different values about the instances activity, while performing the transcoding tasks. Surprisingly enough, we could not identificate clear bottlenecks in any of our scenari. Instead, we remarked that we always underused our computation power. The following is a more detailed review of our results.

3.2.1 Processor

Charts s1-cpu-times.svg and s2-cpu-times.svg shows respectively the overall CPU consumption, i.e. the sum of all vCPU per instance, of both scenari. We remark the same pattern for transcoding units in both cases: CPU consumption smoothly inscreases during the task execution, but seems to be about to reach a stable value as the time goes. This behavior is hard to explain, and we couldn not get a convicing explanation so far. Less surprisingly, in the first scenario we note that storage units makes almost no use of the CPU. Orchestrator unit is not CPU aggressive neither, in both scenari.

An interesting observation could be the slightly greater CPU consumption of one transcoding unit in the second scenario. We explained this by the fact that the file which was used as the source of transcoding was stored in the same unit. Thus, the storage process consumes more resources than the one sharing the instance with the second transcoding unit.

3.2.2 Virtual Memory

Charts s1-vmem.svg and s2-vmem.svg respectively shows the virtual memory consumption of both scenari. In the first scenario, the memory consumption of transcoding units float aroung an average of about 23%. Bursts can be explained by the way ffmpeg fetches the data chunks before transcoding them. In the second scenario, we could observe a different pattern where memory consumption eventually drops brutally. If we watch the chart of tasks status (s2-tasks-status.svg), we understand that this drops matches the number of remaining tasks to process.

In both scenari, storage units clearly underuse their memory while the orchestrator makes a more decent use of it.

3.2.3 Disk and network

As we could not observe bottlenecks, disk and network io counters could only confirm our expectations regarding the behavior of the two scenari. In the first scenario, we had several transcoding units located on actual different instances than the any storage, which means a lot a small data readings on the storage. In the second scenario, storage units shared their instances with transcoding units, meaning that a fewer amount of data would have been sent over the network. Charts s1-network-io-bs.svg and s2-network-io-bs.svg are probably the most significant charts to illustrate this intuition.