Collaborative, distributed and scalable platform based on mobile, cloud, micro services and containers for intensive computing tasks

David Petrocelli [1] [2], Armando De Giusti [3] [4] and Marcelo Naiouf [3]

[1] PhD Student at Computer Science School, La Plata National University, 50 and 120, La Plata, Argentina
[2] Professor and Researcher at Lujan National University, 5 and 7 routes, Luján, Argentina
[3] Instituto de Investigación en Informática LIDI (III-LIDI), Computer Science School, La Plata National University - CIC-PBA, 50 and 120, Argentina
[4] CONICET - National Council of Scientific and Technical Research, Argentina
dmpetrocelli@gmail.com, degiusti@lidi.info.unlp.edu.ar, mnaiouf@lidi.info.unlp.edu.ar

Abstract. Compute-heavy workloads are traditionally run on x86-based HPC platforms and Intel, AMD or Nvidia GPUs; these require a high initial capital expense and ongoing maintenance costs. ARM-based mobile devices offer a radically different paradigm with substantially lower capital and maintenance costs and higher gains in performance and efficiency in recent years. When compared to their x-86 brethren, they have become ubiquitous in consumer markets and are making steady gains in the server market. Given this shifting computer paradigm, it is conceivable that a cost- and power-efficient solution for our world's data processing would include those very same ARM-based mobile devices while they are idling. Given that context, we developed and deployed an auto-scalable, distributed and redundant platform on the basis of a cloud-based service managed via container orchestration and microservices that are in charge of recycling and optimizing these idle resources. We tested the platform performing distributed video compression. We concluded the system allows for improvements in terms of scalability, flexibility, stability, efficiency, and cost for compute-heavy workloads.

Keywords: Kubernetes & Containers, Pipelines, Microservices, Cloud Computing & Storage, Mobile Computing, Distributed & Collaborative Computing

1 Introduction

Developing and deploying a high-quality, distributed, collaborative, and scalable software platform to process intensive tasks requires the adoption of the newest techniques, technologies, tools, and infrastructure patterns that allow taking advantage of all the available benefits on today's computing resources (On-Premise, Cloud and Mobile). We have implemented the following set of features: a) Build lightweight and more scalable applications (Microservices), b) Integrate auto-scalable infrastructure to guarantee an cost and power efficient usage of resources (Container and Container Orchestration) and c) Reuse processing cycles from idle devices (Mobile devices)

Microservices allow for building smaller, lighter, reusable, and self-deployable software components which communicate through a simple and lightweight protocol [1]. Implementing a containerized [2] infrastructure allows microservices to be deployed and run as a naturally distributed application. This provides developers and operations engineers great benefits such as, nimble deployment of software changes simplifying the backups, replication and moving of applications and their dependencies. Operation engineers also use an orchestration layer to track which containers are running and to control, monitor, and scale (shrink or enlarge) applications [3].

In terms of computing power, mobile devices based on ARM chips have long idle periods while they are charging [4]; if properly managed them, they could become massively distributed data centers, consuming only a fraction of the energy [5] for the same computing power [6] as their traditional counterparts.

2 Collaborative, distributed and scalable platform for HPC

Based on the features we described earlier and considering our previous study [7], we developed and deployed the platform based on a model composed by a) Kubernetes container orchestration service (Cloud); b) Tasks management dockerized Microservices; c) Dockerized Queue Middleware and Database System; d) HTTP storage system; e) x86 and ARM-based mobile workers. (see Fig. 1).

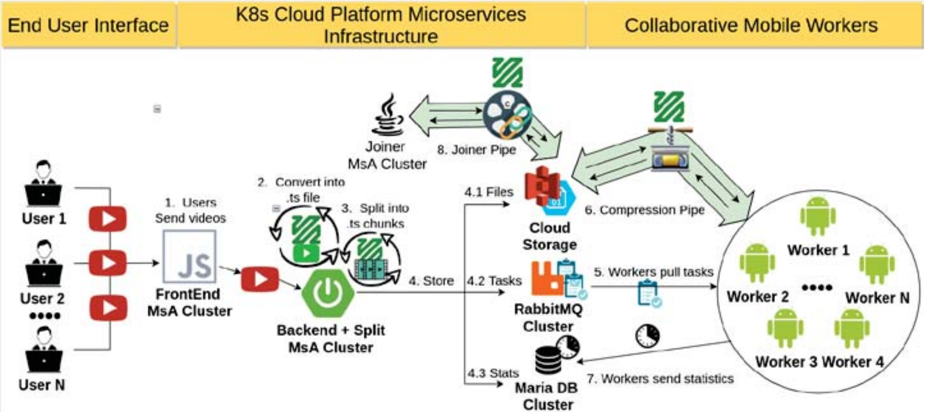


Fig. 1. Functional Diagram of the Collaborating Computing Network.

We used Kubernetes (K8s) AWS (EKS) and Azure (AKS) K8s SaaS clusters to host, orchestrate, heal and monitor our distributed Dockerized services [8]. We configured an auto-scalability mechanism based on CPU and memory container resources utilization thresholds, guaranteeing services (pods) high availability, efficient distribution and scaling (shrink/enlarge) across the cluster nodes. Finally, we configured DNS to interconnect containers, provide traffic routing and load balancing.

The front-end service allows clients to upload source video files and profile coding parameters. It also allows users to visualize information about their in-progress tasks and the result. Once tasks are received in backend microservices, source files are converted into a Video Transport Stream File (ts) due to is the recommended format for video streaming [9] and is tested impacts positively on latency, playback compatibility and viewing experience [10]. Once converted, files are split in smaller chunks [11] and stored in low-cost blob cloud storage [12]. We both use Azure Storage and Amazon S3 [12], to store data reliably and cheaply, upload and download stream files and distribute content via delivery networks (CDN) for lower latency and content caching.

Using official Bitnami RabbitMQ and MariaDB Kubernetes helm charts [13], we built an auto-scalable and fault-tolerant queue and database system where jobs and statistics are published respectively. RabbitMQ is used to securely and asynchronously store, publish and distribute backend service jobs to worker processing nodes. High Availability is guaranteed by RabbitMQ Policies and non-losing tasks are guaranteed by implementing a manual ACK mode model where should a server error, client-side issue, execution timeout happen, RabbitMQ thread moves the task back to queue. MariaDB is used to register job information (parameters, chunks, completed tasks and storage endpoint) the user interface periodically uses to be refreshed. Furthermore, it stores information about executed tasks (task, worker node and executed time). We use these statistics to evaluate the platform and worker efficiency in different scenarios. High Availability is configured via Galera active-active multi-master topology, guaranteeing scalability, smaller latencies, no slave service and no lost transactions.

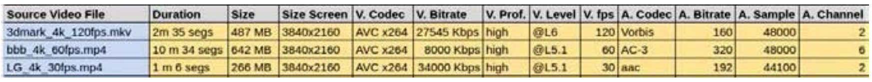
Meanwhile, workers are continuously pulling from the RabbitMQ queue to obtain tasks and compress using the FFmpeg library. When chunks are completely processed, the Joiner backend microservice unifies parts and uploads it to the cloud storage endpoint. Both worker and joiner process tasks parallelly using Linux pipelines. While the source is stream downloaded via HTTP GET curl request, is also processed by FFmpeg and parallelly streamed to the cloud storage vía HTTP PUT curl request. As a result, disk and memory operations are reduced, improving system performance.

3 Platform test and obtained results

So far, we have evaluated K8s microservices behaviour, scalability (defining auto-scale sets based on CPU and memory pod usage) and reliability. First, we forced crashing instances and verified K8s pod recreation and data integrity. Later, we stressed K8s services instances and checked horizontal scaling and load-balancing.

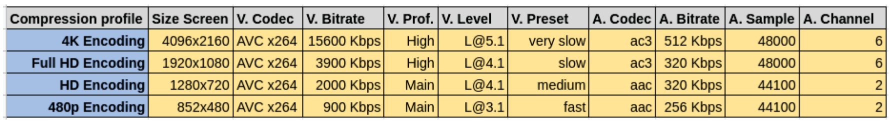
We have experimented with video compression tasks, following streaming best-practices [10][11], selecting representative source videos (see Table 1) and defining a set of h.264 compression profiles (see Table 2) to be executed on ARM-based and x86-based devices. Thus, we obtained performance and power usage metrics.

Table 1. Most relevant source videos features (codecs and bitrate) used for compression tests



Source Video File	Duration	Size	Size Screen	V. Codec	V. Bitrate	V. Prof.	V. Level	V. fps	A. Codec	A. Bitrate	A. Sample	A. Channel
3dmark_4k_120fps.mkv	2m 35 segs	487 MB	3840x2160	AVC x264	27545 Kbps	high	@L6	120	Vorbis	160	48000	2
bbb_4k_60fps.mp4	10 m 34 segs	642 MB	3840x2160	AVC x264	8000 Kbps	high	@L5.1	60	AC-3	320	48000	6
LG_4k_30fps.mp4	1 m 6 segs	266 MB	3840x2160	AVC x264	34000 Kbps	high	@L5.1	30	aac	192	44100	2

Table 2. Most relevant compression profiles properties (size, codecs, resolution and bitrate)



Compression profile	Size Screen	V. Codec	V. Bitrate	V. Prof.	V. Level	V. Preset	A. Codec	A. Bitrate	A. Sample	A. Channel
4K Encoding	4096x2160	AVC x264	15600 Kbps	High	L@5.1	very slow	ac3	512 Kbps	48000	6
Full HD Encoding	1920x1080	AVC x264	3900 Kbps	High	L@4.1	slow	ac3	320 Kbps	48000	6
HD Encoding	1280x720	AVC x264	2000 Kbps	Main	L@4.1	medium	aac	320 Kbps	44100	2
480p Encoding	852x480	AVC x264	900 Kbps	Main	L@3.1	fast	aac	256 Kbps	44100	2

4 Preliminaries conclusions

K8s architecture running platform microservices, based on the experiments we made, might be considered as an interesting infrastructure for HPC tasks. The results showed stability, scalability, good response time and efficiency for different scenarios. Regarding mobile workers, we tested ARM devices are capable of encoding video with a competitive power and cost advantage over traditional x86 architecture. We have recently improved, via pipeline implementation, the worker stability, performance and efficiency.

References

1. Fritzsch J., Bogner J. et al From Monolith to Microservices: A Classification of Refactoring Approaches. First International Workshop, DevOps 2018, France. (2018)
2. Matthias K., Kane S. Docker: Up & Running - 2nd Edition, Shipping Reliable Containers in Production. O'Reilly Media. (2018)
3. Dobies J., Wood J. Kubernetes Operators - Automation the container Orchestration Platform. O'Reilly Media. (2020)
4. Chethan K, Chiranthan H and D'Silva K. A Distributed Computing Infrastructure Using Smartphones. International Advanced Research Journal in Science, Engineering and Technology JSS Academy of Technical Education Vol. 4, Special Issue 8 (2017)
5. Pramanik P., Sinhababu N et al. Power Consumption Analysis, Measurement, Management, and Issues: A State-of-the-Art Review of Smartphone Battery and Energy Usage. IEEE Access 7. DO 10.1109/ACCESS.2019.2958684. (2019)
6. Andatech - The Snapdragon 865 Performance Preview: Setting the Stage for Flagship Android 2020, https://bit.ly/2ZzFdd6, last accessed 2020/05/20
7. Petrocelli, D., De Giusti, A. E. & Naiouf, M. Hybrid Elastic ARM&Cloud HPC Collaborative Platform for Generic Tasks. Springer. Communications in Computer and Information Science. Cloud Computing and Big Data (2019)
8. Hausenblas M., Schimanski S. Programming Kubernetes: Developing Cloud-Native Applications. O'Reilly Media. (2019).
9. ETSI - Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream, https://bit.ly/36segJI, last accessed 2020/05/20
10. 2019 Global Media Format Report, https://bit.ly/2ZBmZIi, last accessed 2020/20/05
11. Dash Industry Forum - Guidelines for Implementation: DASH-IF Interoperability Points, https://bit.ly/3ghFxTT, last accessed 2020/05/20
12. Daher Z., Hajjdiab H. Cloud Storage Comparative Analysis Amazon Simple Storage vs. Microsoft Azure Blob Storage. Int. Journal of Machine Learning, Vol. 8, No. 1. (2018)
13. 13. Bitnami Helm Charts, https://bit.ly/3gsNci7, last accessed 2020/20/05.