6. Limitations

After prototypes implementation and performance evaluation, it is possible to highlight some bottlenecks that could reduce the architecture efficiency, and some limitations that restrict its usage. It is also possible to identify some improvement aspects that could be addressed in further researches, focusing in increase reliability and scalability of the proposed architecture.

One of the main limitations of the Split&Merge architecture, when deployed in the Cloud, is the bandwidth availability between the Cloud platform and the content owner. In fact, this limitation is not directly related with the architecture itself, but, in some cases, it could turn the chosen for public cloud unfeasible. In the scenario in which our tests were performed, this bandwidth issue was impeditive for the adoption of proposed approach in the Globo.com video compression case.

In fact, since the high resolution video, with high data rates, must be uploaded into the Cloud, this approach requires a really high bandwidth availability between the server were content is stored and the virtual machine in the Cloud in which videos will be processed. Ideally the bandwidth should be at least 4 times greater than content data rate, adding only 25% of content duration of overhead in the total time required to obtain the encoded video. As shown in Figure 13 of section 5.1, in the Globo.com case, these high definition videos must be transferred from Globo.com's datacenter to Amazon AWS platform, using the public internet links for this task. In this specific case, the Globo.com's datacenter is in Brazil, and should send videos to Amazon's cloud platform in US, using intercontinental pipes which are frequently overloaded, and where it is not possible to reach even the 25Mbps of data rate used in the original videos. In this scenario, the overall time required to get the encoded content is higher than if the content is processed using the traditional approach, which basically discards all encoding gains obtained by the Split&Merge architecture itself, due to network limitations.

However, the scope of this limitation is restricted to this usage scenario. If the architecture was deployed in a private cloud, inside the content owner's datacenter, or if there is a high bandwidth availability between the Cloud and the content owner, for example, using a dedicated and/or a private link, this network bottleneck could be removed, and all benefits of Split&Merge could be really obtained. Furthermore, the telecommunications infrastructure is constantly evolving, frequently increasing the network capacity. These investments basically means that this will not be an issue anymore in the next few years, and the architecture would be deployed and used as proposed.

Besides this network bottleneck, there is another limitation that could impact in the overall architecture performance, and it is related specifically with the split and the merge steps. In the way which architecture was conceived, all operations of the split and the merge steps were executed sequentially by one single sever (or virtual machine in the Cloud deploy). This means that if the split or the merge steps were complex, with a high computational cost, the performance of process will be reduced.

In the both cases analyzed in previous section, the computational cost of split and merge is directly associated with the input size. For video compression, longer the video, more chunks will be generated, more computational cost will be required to seek, calculate and find chunks. In this case, longer inputs will result in an increased duration of the split step. The same analysis works for the merge step. With more chunks, a longer video stream will be produced, and more computational cost will be needed to order, synchronize and remix the content.

In this approach, performing the split an the merge steps in one single server, as more complex are the performed operations to fragment the input and to combine them to get the output, more the performance will be impacted and lower will be the benefits obtained by the Split&Merge approach. In fact, better results will be obtained for simple split and merge tasks, and complex chunk processing.

Another identified limitation appears when the Split&Merge is deployed in public clouds. As a network bottleneck could be found outside the Cloud infrastructure, when uploading the content, another network issue could degrade the architecture performance, but, this time, inside the Cloud.

During the process step, a master node continuously communicate with the worker nodes, to obtain the processing status, to delegate new tasks, and to perform failover controls, retrieving the state of each node. Working with the approach of one node by each chunk, one can have hundreds or even thousands of working nodes, running in the same Cloud infrastructure. In the limit, the internal network of the Cloud platform will be overloaded by the messages exchanged between the architecture's virtual machines, which basically could stop all content processing. A good approach to tackle these bottlenecks is to constantly monitor the network path between the nodes, controlling overloads by shutting virtual machines down. In this case, having less nodes then chunks could be the best alternative to not increase significantly the time required to obtain the desired results.

Finally, the last important limitation is related with the heterogeneity of public Cloud platforms. When deploying the Split&Merge in a public Cloud environment, one has to perform several customizations to use the API services provided by the platform to allow resources administration. This basically means that each service provider has its proprietary API to, for example, start and stop virtual machines, or to store and retrieve contents from the storage. Deploying the architecture in a different Cloud platform usually means that all interfaces to perform these simple tasks must be rewritten in the application side, which, in some cases, could be really complex and time consuming.

Today there is no standardization of these public APIs to manage the Cloud resources, and, as result, each provider adopts the technologies and formats that fit better in their needs. This problem became even more critical if we consider that each platform uses different authentication/authorization processes, and even different protocols to allow users to perform these basic tasks.

One alternative to address this issue is to build a middleware that is capable to abstract the Cloud platform interfaces from the applications development, and which is maintained by the service providers themselves. In this scenario, the Cloud platforms will became responsible to develop and maintain a common interface for developers to perform this basic set of operations, such start and stop servers.

With this approach, the choice between one Cloud platform and other could be performed at runtime, using the provider with better quality of service at that specific moment, since, by the application point of view, the operations for servers administration will be the same, independent of who is providing the service. Other service providers, such that in the telecommunications area, already do this same approach, abstracting from the device manufacturers the differences between their infrastructures.