5 An Architecture for Distributed High Performance Video Processing in the Cloud

5.1 Introduction

As computer systems evolve, the volume of data to be processed increases significantly, either as a consequence of the expanding amount of information available, or due to the possibility to perform highly complex operations that were not feasible in the past. Nevertheless, tasks that depend on the manipulation of large amounts of information are still performed at large computational cost, i.e., either the processing time will be large, or they will require intensive use of computer resources.

In this scenario, the efficient use of available computational resources is key, and creates a demand for systems that can optimize the use of resources in relation to the amount of data to be processed. This problem becomes increasingly critical when the volume of information to be processed is variable, i.e., there is a seasonal variation of demand for processing. Such variable demand can have different causes, such as an unanticipated burst of client requests, a time-critical simulation, or a high volume of simultaneous video uploads, e.g. as a consequence of a public contest. In these cases, there are moments when there is very low demand and the resources are almost idle while at other moments, there is processing demand that exceeds the resource capacity, and which may cause undesirable delays.

Moreover, from an economical perspective seasonal demands do not justify a massive investment in infrastructure, just to provide enough computing power for peak situations. In this light, the ability to build adaptive systems, capable of using on demand resources provided by Cloud Computing [ARMBRUST 2009][VOGELS 2008][MILLER 2008] is very interesting.

The remainder of this chapter is structured as follows. In the next section (5.2) we discuss computing on demand and the Map-Reduce [DEAN 2008] paradigm, in section 5.3 we introduce the Split&Merge [PEREIRA 2010] architecture, in section 5.4 we discuss fault tolerance issues and compare be-

tween private cluster and public cloud implementations of the Split&Merge architecture. In section 5.5 we conclude and discuss further work.

5.2 Background and Problem Statement

The distribution of tasks in a cluster for parallel processing is not a new concept, and there are several techniques that use this idea to optimize the processing of information. The Map-Reduce paradigm [DEAN 2008], for example, is a framework for processing huge datasets of certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster. It consists of an initial Map stage, where a master node takes the input, chops it into smaller or sub-problems, and distributes the parts to worker nodes, which process the information; following there is the Reduce stage, where the master node collects the answers to all the sub-problems and combines them to produce the job output. The process is illustrated in figure 5.1.

A popular Map-Reduce implementation is Apache's Hadoop [BIALECKI 2011], which consists of one Job Tracker, to which client applications submit Map-Reduce jobs. The Job Tracker pushes work out to available Task Tracker nodes in the cluster, which execute the map and reduce tasks.

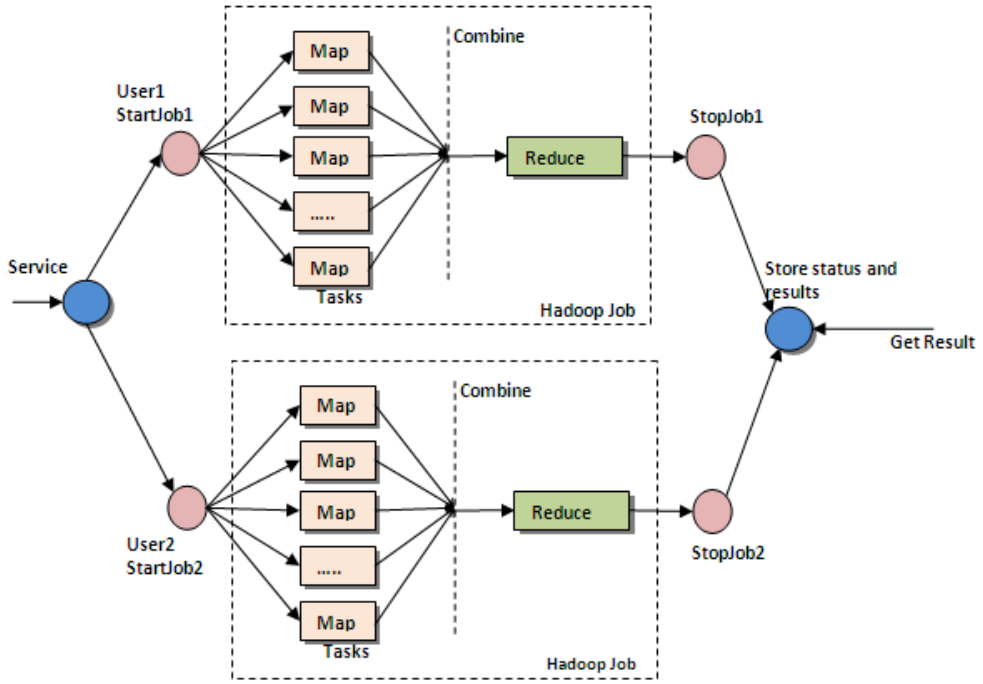

Figure 5.1. Map Reduce architecture.

However, despite being a very appealing and efficient technique for processing large volumes of data, there are a number of challenges associated with the deployment of Map-Reduce architectures. One of them is the required infrastructure. To make the process truly effective, one needs several machines acting as nodes, which often requires a large upfront investment in infrastructure. This point is extremely critical in situations where the processing demand is seasonal. In addition, fault tolerance issues and the need of a shared file system to support mappers and reducers make the deployment of a Map-Reduce architecture complex and costly.

In cases where there is a seasonal computation demand, the use of public Clouds, for information processing and storage, is emerging as an interesting alternative. The Hardware as a Service (HaaS) paradigm relieves the burden of making huge investments in infrastructure, and at the same time supports on-the-fly resizing of resources, and adaptation to current needs.

With a public Cloud, one can quickly make provision for the resources required to perform a particular task, and pay only for the computational resources effectively used. This is good solution, not only because it deploys faster, as opposed to having to order and install physical hardware, but it also optimizes overall costs, as resources can be released immediately after the task is completed.

One of the largest HaaS providers in the public Cloud is Amazon AWS, with its Elastic Cloud Computing (EC2) and Simple Storage Service (S3) services. Amazon EC2 is a web service interface that provides resizable computing capacity in the cloud, allowing ~~a~~ complete control of computing resources and reducing the time required to obtain and boot new server instances. This feature is of particular interest because it allows applications to quickly scale up and down their processing and storage resources as computing requirements change. Amazon S3 provides a simple web services interface that can be used to store and retrieve data on the web, and provides a scalable data storage infrastructure.

In the specific case of applications requiring parallel processing using Map-Reduce architecture, one may also use the Elastic Map Reduce, which implements a hosted Hadoop [BIALECKI 2011] framework running on the infrastructure of Amazon EC2 and Amazon S3.

In this thesis we are interested in computer systems that are capable of being deployed both in private clusters, and in public Cloud providers, under the HaaS

paradigm. In this scenario, the Map-Reduce architecture is an interesting approach, once it is versatile enough to be deployed in both environments. As stated above, Amazon itself already has a Cloud Map-Reduce platform through Elastic Map Reduce. However, the Map Reduce architecture isn't generic enough to be used in all classes of problems that deal with large amounts of data to be processed, once there are some issues that are not addressed efficiently, such as the use of different Reduce algorithms for some specific pieces of information, or the chunk ordering before the Reduce step. A good example where Map-Reduce could be generalized is the compression of high definition video files, which requires intensive information processing. In this compression process, streams of audio and video are processed with different algorithms, and there is a great correlation between subsequent video frames, especially when there is temporal compression. The order in which pieces of audio and video are recombined after having been processed must also be taken into account so as to avoid that significant distortions are incorporated in the output. Moreover, issues such as fault tolerance and scalability need to be thoroughly considered, so that the proposed architecture becomes robust enough to meet the requirements of different video compression applications. These issues will be further discussed in more detail in section 5.4.

In the next section we propose an architecture for video processing, which is sufficiently flexible to be deployed in either a private cluster, private or public Cloud environment, that provides a HaaS platform.

5.3 Distributed Video

Video compression refers to reducing the quantity of data used to represent digital video images, and is a combination of spatial image compression and temporal motion compensation. Video applications require some form of data compression to facilitate storage and transmission. Digital video compression is one of the main issues in digital video encoding, enabling efficient distribution and interchange of visual information.

The process of high quality video encoding is usually very costly to the encoder, which, and require a lot of production time. When we consider situations where there are large content volumes, this is even more critical, since a single video may require the server's processing power for long time periods. Moreover, there are cases where the speed of publication is a critical point. Journalism and

breaking news are typical applications in which the time-to- market the video is very short, so that every second spent in video encoding may represent a loss of audience.

Figure 5.2 shows the speed of encoding of a scene, measured in frames per second, with different implementations of the H.264 compression standard [MPEG4-10 2003]. We note that the higher the quality, i.e., the bitrate of the video output, the lower the speed of encoding.

In order to speed up encoding times, there are basically two solutions. The first one is to augment the investment in encoding hardware infrastructure, to be used in full capacity only at peak times. The downside is that the infrastructure will be idle the remaining of the time. The second solution is to try and optimize the use of available resources. The ideal scenario is to optimize resources by distributing the tasks among them evenly. In the specific case of video encoding, the intuitive solution is to break a video into several pieces and distribute the encoding of each piece among several servers in a cluster. The challenge of this approach is to split, as well as merge video fragments without loss in synchronization.

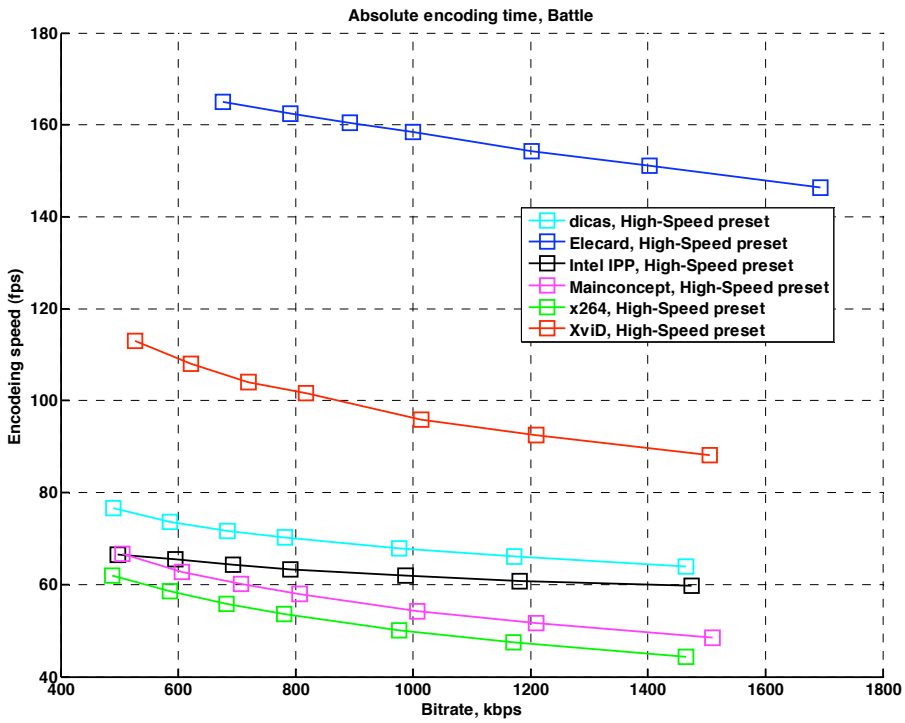

Figure 5.2. Encoding speed for different H.264 implementations [VATOLIN2009].

Given a video encoded in H.264, for example, a split in a frame other than the key-frame (B or P frames) could be disastrous, because the remaining frames depend on information registered in the key frame to be regenerated. Furthermore, the synchronization between audio and video can be greatly affected, since the frame size of each one may not be equal.

5.4 The Split Step

In what follows we describe a technique for reducing video encoding times, based on distributed processing over cluster or cloud environments, implemented using the Split&Merge architecture and illustrated in figure 5.3. The fragmentation of media files and the distribution of encoding tasks in a cluster consist of an advanced solution for increasing the performance of encoding, and an evolution of the simple distribution of single complete video encoding tasks in a cluster or cloud. The idea is to break the media files into smaller files so that its multiple parts can be processed simultaneously on different machines, thereby reducing the total encoding time of the video file.

The problem in the case of video files is that, unlike a text file, we can not split it anywhere. As discussed above, if the video to be encoded already provides some form of temporal compression, then it would be necessary to first identify its key-frames, so that the cuts are made at their exact positions. Furthermore, to avoid synchronization between audio and video problems, we must separate the two, so that they can be independently compressed.

Situations where the original video does not show temporal compression are special cases where the video can be split at specific frame numbers or at regular intervals. The important point here is to ensure that no frame coexists in more than one chunk, and that no frame is lost.

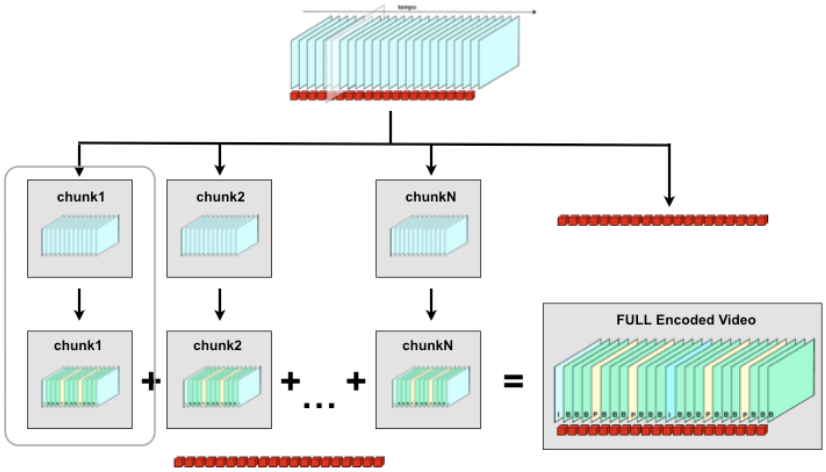

Figure 5.3. The proposed Split&Merge architecture instantiated to video encoding.

Assuming an input video in high definition at 1080p, with 29.97 frames per second, encapsulated in AVI and compressed with MJPEG codec (no temporal compression), and an audio stream stereo PCM, with sampling rate of 44100Hz, the first step in split task would be separate video stream from the audio stream. This is because video encoding is much more complex, and requires much more computer resources than audio encoding. Because the overall impact to the performance is very small, the audio stream is processed in one piece (no fragmentation). Furthermore, if processed together, chunks containing both audio and video may generate various synchronization problems, since audio frames do not necessarily have the same temporal size than video frames. We thus avoid processing both streams simultaneously, for it may generate audible glitches, delays and undesirable effects.

After splitting audio from video, the video stream must be broken at regular intervals. Note that this is only valid in the case where there is no temporal compression at the input. The ideal here is to make chunks with a constant amount of frames, and not based on runtime. When using a time shift split, it is important to make sure that there is no loss or duplication of frames in chunks.

A key point in the fragmentation of the input video is to determine the size of the chunks to be generated. This decision is closely related with the output that is generated, that is, the video codec and compression parameters passed to it in the processing step. This is because, after processing, the chunks will present a key- frame in its beginning and end. Fragmentation in chunks performed indiscriminately, will produce an output video, after the merge, with an excess of key-

frames, which reduces the efficiency of compression. To have an idea, it is frequent the use of spacing between key-frames of 250 frames, when we have a video with 29.97fps. Thus, if in the split step chunks are generated with less than 250 frames, we will inevitably reduce the efficiency of the temporal compression of the encoder. A good approach is to perform the split so that the number of chunks generated is equal to the number of nodes available for processing. However, when we use an elastic processing structure, we can further optimize this split, analyzing what is the optimum amount of chunks to be generated, which certainly varies according the duration of the video, and the characteristics of the input, and output to be produced.

To have this optimized split, would be necessary to implement a decision-making algorithm to evaluate the characteristics of input and output, choosing what size of fragment will use resources more efficiently, producing a high quality result and with an acceptable response time. The implementation of this algorithm is quite desirable in order to improve the efficiency of the process, however, it is beyond the initial scope of this work.

When we split a video file into several chunks, or smaller files, we must repair the container of them, rewriting the header and trailer, most of the time. This process can be avoided with a very interesting method. When we refer to split the video, we are actually preparing the data to be distributed in a cluster, and to be processed in parallel. If in the split step, instead of breaking the video file, we just identify the points of beginning and end of each chunk, then it would not be necessary to rewrite the container, which would consequently reduce the encoding time. The disadvantage in this case would be that all nodes should have read access to the original file, which could be implemented through a share of the file system, as an NFS mount, or even through a distributed file system with high read throughput.

5.5 The Process Step

Once video is fragmented, the chunks generated should be distributed among the nodes to be processed. In the specific case of video compression, this process aims at reducing the size of the video file by eliminating redundancies. In this step, a compression algorithm is applied to each chunk, resulting in a compressed chop of the original video.

The process of chunk encoding is exactly equal to what would be done if the video were processed without fragmentation, i.e. it is independent of the split and the amount of chunks generated. However, if the option to simply mark the points of beginning and end of chunks was used during the split, then the processing step should also have read access to all the original video, and must seek to the position of the start frame, and stop the process when the frame that indicates the end of the chunk is achieved.

There are several open source tools for video compression, among the most popular, FFmpeg [TOMAR 2006] and mencoder, which are compatible with various implementations of audio and video codecs. It is possible, for example, use mencoder to implement the processing step, performing a compression of a high-definition video, generating an output that can be viewed on the Internet, or even on mobile devices that have a UMTS [GANESH 1999] or HSDPA [HOLMA 2006] connectivity. In this case, we could use the H.264 Baseline Profile with 280kbps, and a 480x360 resolution, performing, therefore, an aspect ratio adjustment. Using the marking approach in the split step, the processing step could be implemented as follows:

```
mencoder
  -ofps 30000/1001 -vf crop=${WIDTH}:${HEIGHT},scale=480:360,harddup
        ${ORIGINAL_INPUT_FILE}
  -ss ${CHUNK_START}
  -endpos ${CHUNK_END}
  -sws 2 -of lavf
  -lavfopts format=mp4,i_certify_that_my_video_stream_does_not_use_b_frames
  -ovc x264
  -x264encopts
psnr:bitrate=280:qcomp=0.6:qp_min=10:qp_max=51:qp_step=4:vbv_maxrate=500:vbv_
bufsize=2000:level_idc=30:dct_decimate:me=umh:me_range=16:keyint=250:keyint_min
=25:nofast_pskip:global_header:nocabac:direct_pred=auto:nomixed_refs:trellis=1:bframe
s=0:threads=auto:frameref=1:subq=6
  -nosound
  -o $( printf %04u ${CHUNK_ID} ).mp4
```

In addition to processing the video chunks, it is also necessary to process the audio stream, which was separated during the split step. Audio compression is a

simple process, with low computational cost, and can be performed with the same tools used for video compression:

```
mencoder
  ${ORIGINAL_INPUT_FILE}
  -ovc raw -ofps 30000/1001
  -oac mp3lame
  -af lavcresample=22050,channels=1
  -lameopts cbr:br=32
  -of rawaudio -o audio.mp3
```

At the end of the processing step, we have all the compressed chunks, as well as the audio stream. To obtain the desired output, we must merge all fragments, thus reconstructing the original content.

5.6 The Merge Step

The merge step presents a very interesting challenge, which consists of reconstructing the original content from its parts, so that the fragmentation process is entirely transparent to the end user. This means that not only the joining of the fragments of video should be perfect, but also that the audio and video must be fully synchronized. Note that the audio stream was separated from the video before the fragmentation process took place. As compression does not affect the length of the content, in theory after merging the processed chunks, we just need to realign the streams through content mixing.

The first phase of the merge step is to join the chunks of processed video, which can be accomplished easily by ordering the fragments and rewriting the container.

```
mencoder
        ${LIST_OF_VIDEO_CHUNKS}
        -ovc copy -nosound -of lavf
        -lavfopts
        format=mp4,i_certify_that_my_video_stream_does_not_use_b_frames
        -o video.mp4
```

Following, we remix the audio stream with the video, synchronizing the contents, and generating the expected output.

```
mencoder
        video.mp4
        -audio-demuxer lavf
        -audiofile audio.mp3
        -ovc copy -oac copy -of lavf
        -lavfopts
        format=mp4,i_certify_that_my_video_stream_does_not_use_b_frames
        -o ${OUTPUT}
```

After the split, process and merge steps, implemented using the proposed architecture, we created a fully parallel and distributed video compression process, where the different pieces of content can be processed simultaneously in a cluster or, alternatively, using resources in the Cloud.

5.7 Performance Tests

In order to validate the proposed architecture we experimented using Amazon's AWS services. We deployed an instance application responsible for the encoding different sequences of videos, evaluating the total time required for the encoding process, and comparing it with the total time spent in the traditional process, where the video is encoded without fragmentation, i.e. all content is rendered on a single server.

For these tests, we selected sequences of high-definition video, with different durations, and encoded with MJPEG 25Mbps, 29.97fps, and audio PCM/16 Stereo 48kHz. The video output of the compression process was set to be an H.264 Baseline Profile, with 800kbps, 29.97fps, and with a resolution of 854x480 (ED), and audio AAC, 64kbps, Mono, 44100Hz.

To deploy the infrastructure, we chose the instance type m1.small for all instances of servers, which has the following characteristics, according to Amazon:

- 1.7 GB memory
- 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)
- 160 GB instance storage (150 GB plus 10 GB root partition)
- I/O Performance: Moderate

In figure 5.4, as follows we depict the comparison between the total times, measured in seconds, required for the encoding of different video sequences, us-

ing the proposed Split&Merge implementation, and using the traditional compression process. In this test scenario, we worked with chunks of fixed size (749 frames), and with one node per chunk.

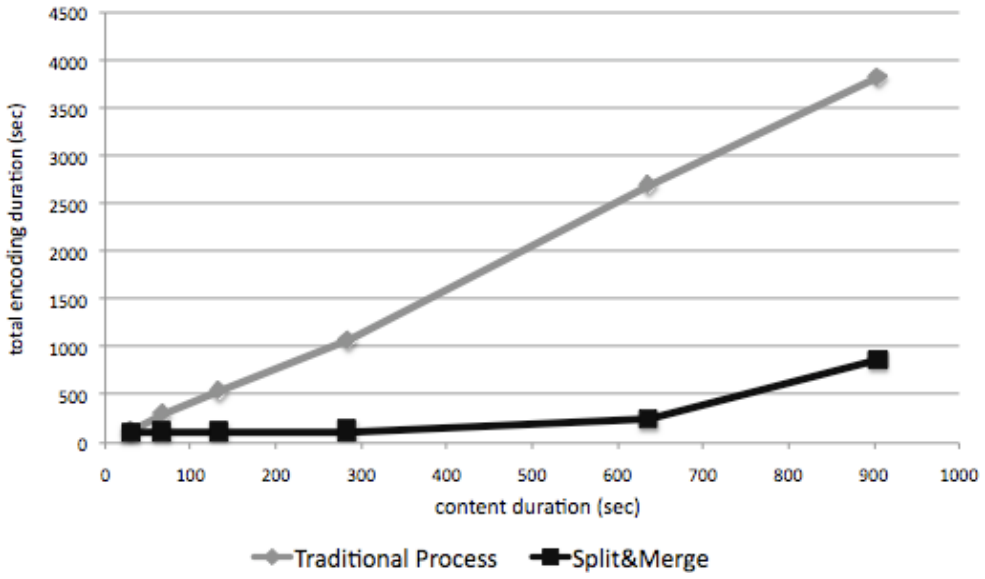

Figure 5.4. Total encoding times for different sequence durations (s).

As we see, while the total encoding time using the traditional process, grows linearly with increasing duration of the video input, the Split&Merge, average process times remain almost constant for short duration videos. In fact, the total CPU time consumed, which is the sum of the CPU usage in all nodes, will be greater in the Split&Merge approach, however, the distribution of processing among several nodes for parallel execution will reduce the total encoding duration. This result is quite significant when one considers videos of short and average duration. In the case when we have a video about 10 minutes long, the total time for encoding using the technique of Split&Merge is equivalent to less than 10% of the total time spend using the traditional process, which is extremely interesting for applications where time to market is vital, as breaking news. If we consider another application, such as the production of soccer matches, which have videos about 2 hours long, the gain would be even more significant if we increase the amount of available nodes by provisioning of servers in the Cloud. In this case, we could reduce the total production time from several hours to a few minutes. However, with the increasing of the number of chunks being processed

simultaneously, the efficiency of the Split&Merge tends to be reduced due the network I.O rates, as we could see with the sequence of 900 seconds duration.

Whereas, in the limit, the elastic capacity of a public Cloud tends to be infinite, i.e. the amount of resources available is unlimited (including networks), then we can say that it is possible to encode all the video content of a production studio collection, with thousands of hours of content, in a few minutes, by using the approach of Split&Merge deployed in a Cloud, which certainly would take hundreds of hours using the traditional process of coding in a private cluster.

Another key point that must be considered is the cost of the Split&Merge approach deployed in the public Cloud, against the cost of having a private infrastructure dedicated to this task. Taking into account the results of the tests above, and also a production of 500 minutes a month, we will have, at the end of one year, an approximate cost of $25,000 using the platform of Amazon AWS, with the advantage to be possible to produce all content in a few minutes. This value is comparable to the cost of a single server, without even considering the depreciation and maintenance, which makes the architecture of Split&Merge deployed in the public Cloud not only efficient in terms of processing time, but also in cost of deploy and operation.

Considering an optimal situation where there are unlimited resources available, it is possible to use the experimental results to predict the total cost and number of nodes needed to encode videos of different categories. Table 5.1, bellow, compares the traditional encoding process with the proposed Split&Merge approach. In this example we set the total encoding time to 2 minutes, and explore several different scenarios, i.e. advertisements, breaking news, TV shows and movies or sports matches, respectively. We are also using the cost per minute, although Amazon's minimum timeframe be one hour, considering scenarios where there are a great number of videos to be processed, so, consequently, the machines are not shut down after a single process.

Input Video Duration	*Traditional Encoding Duration*	*S&M Encoding Duration*	*Number of S&M Nodes*	*S&M Encoding Cost Using EC2 (in US dollar)*
30 sec.	2 min.	2 min.	1	$0.003
5 min.	19 min.	2 min.	10	$0.03

30 min.	112 min.	2 min.	57	$0.16
2 hour	7.5 hour	2 min.	225	$0.63

Table 5.1. Comparison between the traditional encoding process and the split & merge approach.

Note that the Split&Merge approach, deployed in a public Cloud, reduces the total encoding time for a 2-hour video from 7.5 hours to 2 minutes, with the total processing cost of $0.63. If we extrapolate these numbers for the Super Bowl XLIV [STLTODAY 2011], it is possible to encode the 3.3 hours match for $1.03, in only 2 minutes, as opposed to 12.2 hours, if we opted for the traditional process.

5.8 Discussion

Scalable and fault tolerant architectures that support parallel processing of large volumes of data in Cloud environments, are becoming increasingly necessary to enable flexible, robust and efficient processing of large volumes of data. In this paper we are specifically interested in architectures that deal with large-scale video processing tasks, as most of the existing video processing techniques do not consider parallel computing in the cloud. In what follows we discuss the requirements for such architecture, focusing on the dynamic deployment of additional computer resources in the Clouds, as the means to handle seasonal load variations.

The first requirement for such architecture is portability, i.e., it should be designed in a way that it works on different platforms. Therefore, it should be simple and componentized, so that it can be deployed in hybrid infra-structures, including machines in private clusters, as well as in public Clouds. With approach it would be possible, for example, to combine servers in a local cluster, to instances running on Amazon EC2, simultaneously processing tasks. An alternative scenario is to have the processing done locally, using servers in a private cluster, but using a public Cloud for storage. There are several other possibilities, but the fundamental point is that portability is central to secure flexible solutions that combine private and public Cloud resources. These are commonly referred to as hybrid cloud computing [CEARLEY 2009].

To make this possible, all components of the architecture should be service oriented, i.e., they must implement web services that allow functional architectur-

al building-blocks to be accessed over standard Internet protocols, independently of specific platform and programming languages. This is a key feature when dealing with services in the Cloud. In the case of Cloud services provided by Amazon, for example, it is possible to manipulate data stored in Amazon S3, and to provision resources in Amazon EC2, scaling up and down, and deploy virtual machines, using programs that communicate through REST web services [ZHANG 2004]. Thus, a task processing architecture should provide a standard service-oriented interface for scheduling and manipulation of jobs. The communication amongst its internal components should, of course, make use of the same standard. This makes deployment more flexible, and also facilitates the extension of existing features, addition and/or removal of software components, as needed.

If we analyze Map-Reduce in its essence, we note that process optimization is achieved by distributing tasks among available computing resources. The possibility of breaking down the input, and processing its parts in parallel, is the key to reducing overall processing times. In the particular case of video processing, special care must be given to the processes of merging and recombining processed fragments, so as to preserve their original order, avoid synchronization problems, and quality loss.

5.9 Fault Tolerance

To understand how the Split&Merge architecture deals with possible failures in its components, we need to detail the implementation of redundancy mechanisms, component behavior, and information exchange. The first point is the way in which messages are exchanged. We advocate in favor of a service-oriented architecture, based on exchange of messages through REST web services [FIELDING 2000].

The typical Map-Reduce implementation provides a single master node, responsible for the scheduling tasks to worker nodes (responsible for doing the processing). Communication between workers and the master node is bidirectional: the master node delegates tasks to workers, and the workers post the execution status to the master. This type of architecture has received severe criticism, as a single failure can result in the collapse of the entire system. Conversely, worker failures could happen without ever being detected.

The Split&Merge architecture tackles this problem by coupling a service to the master node that periodically checks the conditions of its workers. This ensures that the master node, which controls the entire distribution of tasks, is always able to identify whether a node is healthy or not. This simple mechanism can be further refined as necessary, e.g., adding autonomic features, such as monitoring workers to predict when a particular worker is about to fail, isolating problematic nodes, or rescheduling tasks. Of course, care must be taken to avoid overloading the master node with re-scheduling requests, and additional overhead caused by recovery and prevention mechanisms.

Another issue addressed by the proposed Split&Merge architecture is related to the fact that there is a single point of failure on the master node. The main challenge in having two active masters is sharing state control between them. More specifically, state control sharing means that, whenever it delegates a task, the master node responsible for such operation must inform its mirror which task has been delegated to which worker node, so that both are able understand the processing status post from the worker(s). Sharing of states can be implemented through several approaches, but our choice was to use master replication using a common database. Apart from the simplicity of state sharing, with this solution we also secure control state persistence, which means that in case of failure, we may resume processing of the chunks from the last consistent state.

We must pay attention to how workers read the input and write processing results, which translates to the problem of ensuring file system reliability. In the cases where a private cluster is used, we opted for a shared file system, e.g. NFS, HDFS (that uses a distributed architecture) [BORTHAKUR 2008] or MogileFS. They seem a natural choice as Distributed file systems, in general, already incorporate efficient redundancy mechanisms. In cases where the Cloud is used, storage redundancy is already fully transparent, which greatly simplifies the deployment architecture. However, we must note that data writing and reading delays in Cloud storage systems are significantly higher, and often depend on the quality of the connection among nodes and the servers that store content.

In the following two sections we compare between two possible implementations of the Split&Merge architecture.

5.10 Private Cluster

Defining a flexible and robust architecture is not enough, as we must also guarantee that it is suited to the environment where it is going to be deployed. When considering deployment in a private cluster, we identify a number of characteristics in the environment that may interfere in the architecture efficiency, and thus require subtle changes, to enable more rational use of available resources.

One of the main features of a private cluster is the control over the environment and the ability to customize it. It is possible to have an accurate idea of the computational power of each component, and identify existing bottlenecks, which greatly facilitates the processes of deployment, management and maintenance of existing resources. Figure 5.5 illustrates the proposed architecture, instantiated to a cluster environment.

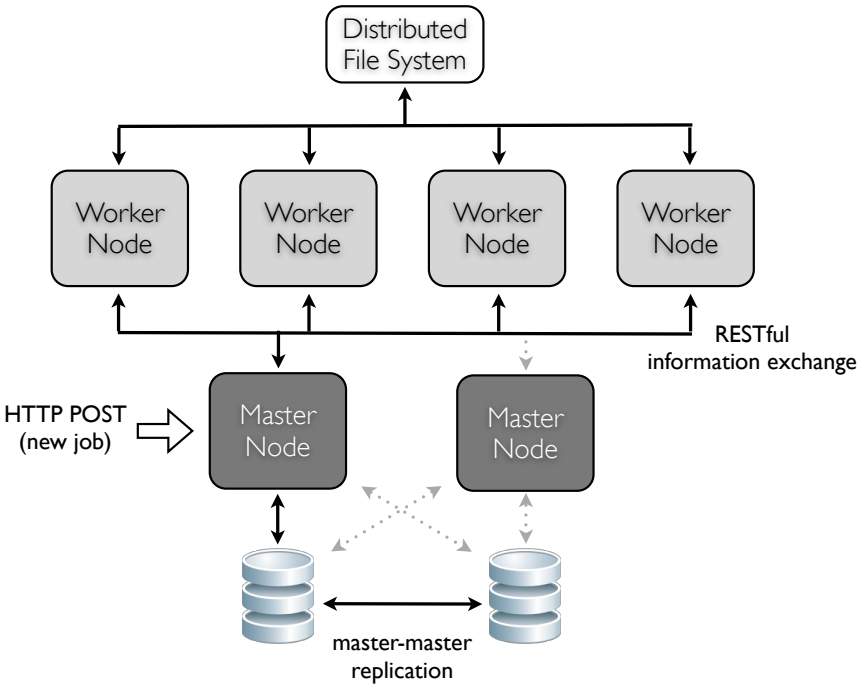

Figure 5.5. Private cluster Split&Merge architecture.

The great disadvantage of a cluster consists in the investment necessary to set it up, which is not justifiable in cases of seasonal demands. The deployment of applications using the proposed architecture, with a minimum redundancy of as described in the previous session, would require at least 4 servers, which is a significant investment when we consider that, together with the cost of hardware,

there are maintenance, depreciation, electricity / cooling, human resources, amongst other additional costs.

A major advantage of using a private cluster consists, in most cases, in the guaranteed and exclusive bandwidth available between the nodes of the architecture. This point is important for applications that handle large volumes of data because they usually are I.O. intensive, both in terms of file system and network-oriented I.O., especially when using a shared file system. In such cases, scalability may be limited by network bottlenecks. Therefore, for such applications a good connectivity infrastructure between the components is paramount.

5.11 Public Cloud Deployment

In cases where there is a floating demand or services, or when sudden changes to the environment dictate the need for additional resources, the use of public Cloud Computing platforms to launch applications developed using the Split&Merge architecture becomes extremely interesting. The "pay-as-you-go" business model provides a series of advantages: there are no fixed costs, no depreciation, and it does not require a high initial investment. Furthermore, it is totally elastic, i.e., it is possible to add and remove workers at any time, according to demand. If there is no demand, all workers can be turned off, on the fly, by the master node. Cost of operation is thus minimal. Even masters may be disconnected, and re-connected, manually, which makes the total cost of operation in idle situations very low. In Figure 5.6 we illustrate the Split&Merge architecture when used with public Cloud Environments. Because we opted for the Amazon Web Services (AWS) for our experiments, the examples used throughout the text refer to their services. The Split&Merge architecture, however, is general enough to accommodate other choices of cloud service providers. Figure 5.6 illustrates the proposed architecture, instantiated to Amazon cloud services.

To enable the use of Amazon Web Services to deploy applications using of the proposed architecture, we first need to build an image (AMI) for EC2 instances, one that corresponds to one full installed and configured worker. This way we ensure that new instances can be started in a state of readiness. We also need an image for the master node.

For our storage needs, we use Amazon S3. In this case, redundancy and availability concerns are transparent and delegated to the Cloud provider. We also

use the Amazon Relational Database Service that implements a simple relational database.

An important point to consider when making a deployment in a public Cloud service is data delivery and recovery in the Cloud storage. It is relevant because, in addition to paying for data transfer, the network throughput is limited to the bandwidth availability between the destination and origin, Internet links, in general.

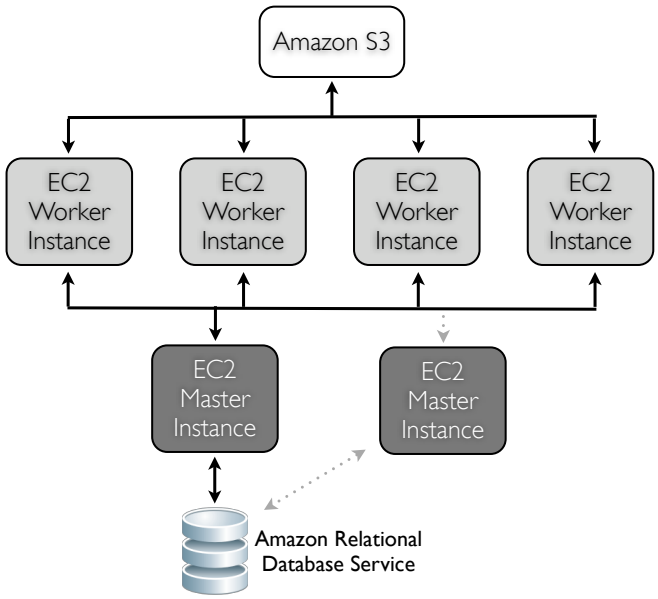

Figure 5.6. Public cloud Split&Merge architecture (depicted using Amazon S3).

5.12 Conclusion

With increasing demand for processing of large volumes of information, the need for systems able to meet this demand efficiently also increases, which promotes the research on architectures and techniques to optimize use of available resources. In addition, the emergence and evolution of services in the Cloud, which allows computing resources to be used on demand, increases flexibility and makes the processing of large datasets more accessible and scalable, especially when we have seasonal demands.

There are now various techniques and paradigms aimed at the optimization of computing resources usage, and the improvement of performance in information processing. Among them, we highlight the Map-Reduce, however, we understood that for some situations it could be generalized and simplified. A specific case that was presented was video compression, and, for this type of processing,

we propose a more efficient architecture and that can be deployed both in private clusters, as in the Cloud

It is important to note that the techniques used in the split and merge steps are hotspots, i.e., the techniques can be exchanged and customized as needed. That ensures flexibility, adaptation, extensibility and generality to proposed architecture. In the case of video processing, it is paramount to allow a choice among codecs, containers, audio streams, and different splitting techniques. Let's take the cases where the input video has no temporal compression, MJPEG [SYMES 1998] for example. In such cases, the split operation can be performed at any video frame. Conversely, cases when the input is a video with a temporal compression that uses p-frames only – H.264 [MPEG4-10 2003] Baseline Profile for example – it is mandatory to identify the key-frames before splitting.

The generalization of this idea, lead to an architecture in which it is possible to isolate, and modify the implementations for the split, process and merge steps. The current implementation encompasses a great plethora of techniques that one can choose from, ranging from video compression, to image processing, through the fragmentation of their macro-blocks, to simple word counts. The choice of technique can be done at scheduling time, the implementation embeds the technique of choice in a web service at run time.

We also pointed out that the deployment in a private cluster is only justified in situations where there is a constant demand for processing. The alternative, therefore, to situations where the demand is seasonal, is using a public cloud platform, such as Amazon, where one pays for the exact amount of resources used.

We validated the proposed Split&Merge architecture and verified its behavior of processing a large volume of information, both in the private cluster and public cloud scenario, by instantiation of a distributed video compression application. As a result we were able to dramatically reduce video encoding times. In fact, we demonstrated that if we scale up the architecture to one worker per 250-frame chunk, we can guarantee fixed encoding times, independently of the size of the video input. This is extremely interesting for content producers, because it is possible to establish entry independent SLAs for video encoding services.