Parallelizing Video Transcoding Using Map-Reduce-Based Cloud Computing

Feng Lao, Xinggong Zhang * and Zongming Guo
Institute of Computer Science & Technology
Peking University, Beijing 100871, P.R. China
Email: {laofeng, zhangxg, guozongming}@pku.edu.cn

***Abstract*— Due to the complexity of video coding, fast transcoding is still a challenge. Various parallel coding methods have been proposed. In this paper, we present a parallel transcoding system over Map/Reduce cloud computing architecture. Input video sequences are divided into segments, and mapped to multiple computers. The sub-tasks are launched in parallel with processing results concatenated to the final output sequences. For heterogeneous clips, computing capacity, and task-launching overhead, the task scheduling over cloud is an NP-hard problem. We propose a low-complexity heuristic algorithm, Max-MCT, to find out the optimal solutions for task scheduling. By estimating the low-bound of finish time, we transform the problem into a virtual knapsack problem. But it is not an optimal solution for the original problem therefore we use a minimal complete time (MCT) algorithm to minimize the entire finish time. We carry out extensive experiments on numerical simulations. The results verified that our algorithm outperforms the existing algorithms.**

I. Introduction

Recent years, there has been a growing demand for high quality video, which leads to advances of coding technology, such as H.264, MPEG-4 and MPEG-2 and so on. And various environments usually require different coding formats. This results in the demand of fast transcoding. However, due to the complexity of video coding, fast transcoding remains a problem to be explored. This gives rise to parallel transcoding. There have been many efforts devoted to parallel transcoding over multi-core processor, such as [1] [2] [3]. But due to specified hardware, the parallel transcoding over multi-core processor is hard to extend. Cloud computing, as an emerging technology, can utilize computing power of thousands of computers. It is a potential platform for parallel transcoding.

Cloud computing consists of a cluster of distributed computers. Each of them can be engaged in their tasks independently and all of them together provide a great parallel computing power. Since the computers can be heterogeneous, cloud computing is extendable and relatively inexpensive. Map/Reduce is a distributing cloud computing model. As showed in figure 1, in this model, the input data would be divided into many splits. These splits would be submitted as sub-tasks and handled by mappers in multiple computers. The output of mappers would be handled by reducers to provide the final output.

There have been some research efforts devoted to cloud computing. [4] and [5] divide the video file into small segments and transcodes them in parallel. They focused on handling the length of segments to optimize the transcoding speed. [6] proposed CloudStream, a cloud-based proxy that transcodes streaming video in real time. They formulated the transcoding process as an on-line scheduling problem and gave two mapping options to optimize transcoding speed and reduce the transcoding jitters. However, they did not consider the overhead when launching a sub-task.

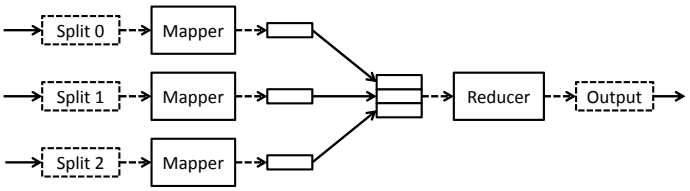


Fig. 1. Cloud Computing Architecture

For Map-Reduce-based cloud computing here, we mainly focus on the scheduling for sub-tasks to multiple computers. [7] compared 11 heuristic algorithms and found that Min-Min performed the best. Moreover, when the transcoding time is in proportion to segment complexity, Min-Min algorithm is equal to minimal complete time (MCT) algorithm. The MCT algorithm assigns segments according to descending complexity order. Each time, it assigns the segment to the computer with the shortest time to complete the transcoding. Obviously, it would average the finish time of the computers. However, this may result in placing too many segments and task-launching overhead in powerful computers.

In this paper, We introduce a Map-Reduce-base cloud transcoding system, which divides video sequence into segments and submits them as sub-tasks. We formulate the scheduling as an NP-hard problem. Considering overhead to launch sub-tasks, we propose a heuristic task scheduling algorithm, named Maximizing Minimal Complete Time (Max-MCT), which includes two procedures: virtual knapsack and MCT procedures. At first, we convert our problem to a virtual knapsack problem and assign complex segments to powerful computers, so that these computers would not waste too much capacity on task-launching overhead. For residual pieces, we employ MCT algorithm to assign them to the computers with the minimal complete time, which would average the finish time of the computers. To verify our algorithm, we carry out

*Corresponding author

978-1-4673-0219-7/12/$31.00 ©2012 IEEE

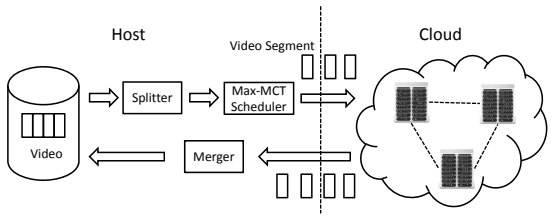


Fig. 2. System Architecture

extensive numerical experiments. The results show that our algorithm outperforms the existing algorithms significantly.

The rest of this paper is organized as follow. Chapter two is our system architecture. Problem formulation is presented in chapter three. We propose Max-MCT algorithm in chapter four and experiment results are given in chapter five.

II. System Architecture

We now present our Map-Reduce-based cloud transcoding system, which consists of a host and a cloud. As showed in figure 2.

Upon user or operator request, the splitter at the host gets the input video sequence from storage and divides it into several video segments. To insure the independency of the segments, video sequence should be divided in between GOPs. For the case of Open-GOP, in which there exists dependency between GOPs, we duplicate one GOP after segmentation point. This technique had been introduced in [4]. The difference is that we do not limit the length of segments. Moreover, the content of each segment is also various. Therefore, the complexity of segments is heterogeneous. There have been many efforts to estimate the complexity of video sequence, such as using time motion metric, spatial detail metric and length and so on. These can be achieved by profiling [8] [9]. Here, we just assume we can obtain the complexity of each segment. After generated, all segments are mapped to multiple computers by Max-MCT scheduler, and then pushed to the cloud.

The cloud consists of a cluster of computers with different computing power. The computer capacities can be estimated and normalize by the historical transcoding record. Base on the Map/Reduce model, we submit segments as sub-tasks and assign them to computers according to Max-MCT scheduler. In each computer, all sub-tasks must be processed sequentially, without preemption. To launch a sub-task, computers have to obtain the video segment from the distributed file system of the cloud and prepare the input data. These are irrelevant with the computer capacity. We denote them as a constant task-launching overhead. The cloud transcoding finishes when all computers finish their sub-tasks.

After the cloud transcoding, all transcoded segments are available at the cloud. The merger at the host downloads these segments and concatenates them together to provide the output video sequence.

III. Problem formulation

For a given video sequence, the time spent for dividend and concatenation are constant. We only consider the transcoding process at the cloud and formulate it as a scheduling problem.

As present above, we are given n different segments, $J = (1, 2, ..., n)$ with different complexity, $C = (c_1, c_2, ..., c_n)$. Each segment must be processed without preemption until its completion. We also have m computers with different capacity, $P = (p_1, p_2, ..., p_n)$. And the task-launching overhead is $t_{overhead}$. The transcoding time is proportional to segment complexity and inversely proportional to computer capacity. So that the transcoding time spent for segment i on computer j can be computed as $t_{ij} = c_i/p_j + t_{overhead}$.

After some scheduling strategy, each computer would have several segments. We denote the set of segments on computer j as S_j. Then the finish time of set S_j is

$$f_{S_j} = \sum_{i \in S_j} t_{ij} = \sum_{i \in S_j} c_i/p_j + |S_j| \times t_{overhead}$$

The scheduling strategy can be denoted as $L = \{S_1, S_2, ..., S_m\}$. The assignment of segment i is $A(i) : J \to L$, which means the mapping of segments to computers. The goal here is to find the scheduling strategy to minimize the entire finish time, which is bounded by the maximal finish time of all the computers. So our problem can be formulated as

$$\min_{L} \max_{S_j \in L} f_{S_j}$$

s.t.

$$L = \{S_1, S_2 ... S_m\}$$
$$\bigcup_{S_j \in L} S_j = J$$
$$\forall S_i, S_j \in L, S_i \cap S_j = \emptyset$$

We show that this problem is NP-hard. Consider the sub problem that computer capacities are identical and the task-launching overhead is zero. This sub problem here is known to be the load balance problem, which had been proved to be NP-hard. So our original problem is NP-hard. To find the optimal solution we have to traverse all the possible solutions, which leads to a complexity of $\mathrm{O}(n^m)$. Therefore, heuristic algorithms are proposed.

Our motivation is, in addition to the load balance problem, our problem has heterogeneous computing power and non-zero task-launching overhead. Since the task-launching overhead is constant to different computers, we should map complex segments to powerful computers so that they would have relatively fewer sub-tasks. Therefore they would not waste too much capacity on overhead.

IV. Max-MCT algorithm

Here, we proposed a novel heuristic algorithm, named Maximizing Minimal Complete Time (Max-MCT), which consists of two procedures: virtual knapsack and MCT procedure. In the virtual knapsack procedure, by estimating the low-bound and converting our problem to a virtual knapsack problem, we assign complex segments to powerful computers until they may overflow. Then, we employ MCT algorithm for the residual segments to average the finish times, so that we can minimize the entire finish time.

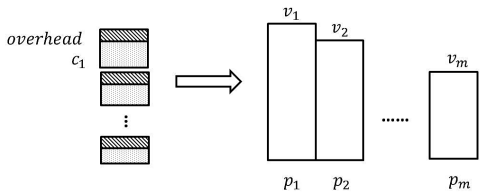


Fig. 3. Virtual Knapsack

A. Virtual knapsack procedure

At first, we are given m computers and n segments. We can estimate the low-bound as the average finish time:

$$f^* = \sum c_i / \sum p_j + t_{overhead} \times n/m$$

Obviously, this gives the optimal situation. If the finish times of some computers are lower than the optimal finish time, there must exist some computers whose finish times exceed the low-bound. Therefore, we can treat the computers as virtual knapsacks with a volume $v_i = p_i \times f^*$. And there are n items weighting $c_i + overhead$. Here, $overhead$ denote the task-launching overhead. As showed in figure 3. Therefore, our problem is converted to a virtual knapsack problem, which limits that the knapsacks can not overflow. In this procedure we schedule complex segments to powerful computers.

We use f_j to record the finish time of computer j. Then we fill the computers in descending capacity order. For each time, it attempts to place the most complex unassigned segments to the current handling computer. Before assigning, we estimate if the computer may overflow. That is for segment i to computer j, if $f_j + t_{ij} > f^*$, the computer would overflow. Then we should begin to handling the next computer, and attempt to place the segment to it.

B. MCT procedure

After the virtual knapsack procedure, there might be some residual segments and some gaps between f_j and f^*, which are different from each others. Therefore we should assign these segments to minimize the maximal finish time, since the entire finish time is bound by the computer with the maximal finish time.

Here, we employ MCT algorithm to handle the residual pieces. It traverses the segments in descending complexity order. For each segment, it estimates the complete time on each computer, such as segment i on computer j has $t_{ij}^c = f_j + t_{ij}$. Then the computer with the minimal complete time is chosen. It continues until all the residual segments have been assigned. Obviously, MCT algorithm tends to average the finish time of the computers.

C. Algorithm Analysis

Now, we analyze the complexity of our Max-MCT algorithm. Sorting the segments according has complexity $\mathrm{O}(n \log n)$. The virtual knapsack procedure has complexity $\mathrm{O}(n)$ and the complexity of MCT algorithm is $\mathrm{O}(nm)$. So our algorithm has a low complexity $\mathrm{O}(n \log n)$.

It can be proved that the entire finish time generate by our algorithm is at most twice the average finish time. To simplify, we assume that $n > m$. If $n \leq m$, it is easy to prove. And we denote $f_{entire} = \max_{S_j \in L} f_{S_j}$.

Theorem 1: the entire finish time by Max-MCT algorithm is not larger than twice the average finish time, that is

$$f_{entire} \leq 2 \times f^* = 2\left(\sum c_i / \sum p_j + n \times t_{overhead}/m\right)$$

Proof: Assume that the entire finish time by Max-MCT algorithm is larger than twice the average finish time. Suppose there are many cases satisfying such assumption. Then it must exist a case having the smallest n. In this case, the least complex segment is not only the last to be scheduled but also the last to be completed. Otherwise, removing this segment makes no difference to f_{entire} but reduces f^* and the assumption is still satisfied.

In the case with the smallest n, f_{entire} is the finish time of the computer that handles the last segment. It satisfies that

$$f_{entire} \leq f_i + c_n/p_i + \mathrm{t_{overhead}}$$

Then, with our assumption, we can get

$$f_{entire} p_i \leq f_i p_i + c_n + p_i t_{overhead}$$

$$f_{entire} \sum p_i \leq \sum f_i p_i + m c_n + t_{overhead} \sum p_i$$

$$f_{entire} \leq \sum f_i p_i / \sum p_i + m c_n / \sum p_i + t_{overhead}$$

$$f_{entire} \leq \frac{\sum c_i}{\sum p_j} + n \times \frac{t_{overhead}}{m} + \frac{m c_n}{\sum p_i} + t_{overhead}$$

$$\sum c_i / \sum p_j + n \times t_{overhead}/m < m c_n / \sum p_i + t_{overhead}$$

It is known that c_n is the least complex segment, and $n < m$, so the above expression is never set up. The assumption is incorrect and the theorem is proved. ■

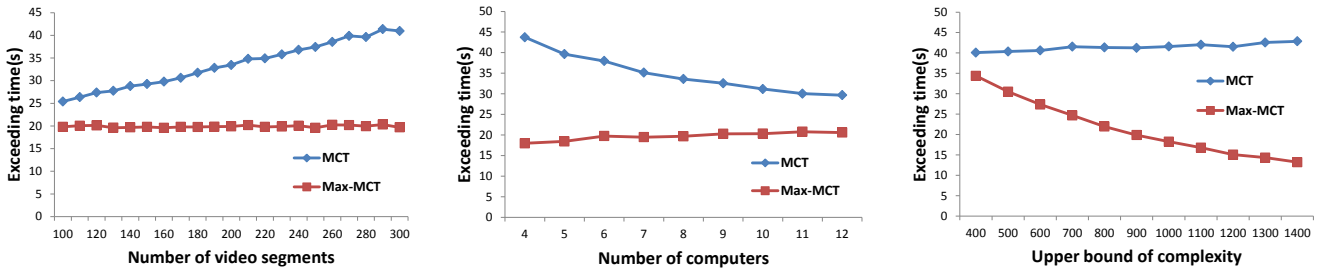
V. Experiment

As mentioned above, we focus on the task scheduling to mainly consider segment length, computer capacity and task-launching overhead. Compared with these operations, other overheads, such as network or system overhead, is too small to be considered in our simulations. Thus we employ Matlab to conduct simulation experiments to evaluate different scheduling strategies.

Generally, we create 8 computers, with capacity ranging from 5 to 15, and 300 video segments whose complexity ranges from 300 to 900. And the task-launching overhead is 10. For each situation, we conduct 1000 experiments and pick the average as an output. Here we mainly evaluate the Max-MCT algorithm against MCT algorithm.

We mainly compares the entire finish time of algorithms. As we mention above, f^* is the optimal finish time and we want to approximate it. Hence, we compare the gaps between the entire finish time and f^*, which is denoted as exceeding time, $t_e = f_{entire} - f^*$.

In figure 4(a), we vary the number of segments from 100 to 300. Obviously, the exceeding time of MCT algorithm increases as the number of segments increase. While the



(a) different numbers of segments (b) different numbers of computers (c) different range of complexity

Fig. 4. Exceeding time with different environments

exceeding time of our proposal is not only lower but also insensitive to the change of segments number. And figure 4(b) shows the result of changing computer number from 4 to 12. The MCT algorithm gets decreasing exceeding time as the number of computers increase. While our algorithm gets almost constant and lower exceeding time. Both of the results show that Max-MCT algorithm performs better.

We also examine the performance with different complexity feature. As showed in figure 4(c). We set the lower bound of the complexity to 300 and vary the upper bound from 400 to 1400. The performance of MCT is not sensitive to the change of complexity range. However, the exceeding time of our algorithm is decrease when the upper bound is increasing. This is because, as the differentiation of the video segment increase, putting the complex segments into the powerful computer becoming more significant and greatly reduce the capacity waste of powerful computer.

The main difference between our algorithm and MCT is the virtual knapsack procedure that tends to map complex segments to powerful computers to reduce the sub-tasks in these computers. Figure 5 shows this feature. We compare the number of the video segments in different computers. The computer here is sorted in descending capacity order. As we can see, MCT algorithm tends to assign more segments to the powerful computers. The first computer gets almost 50 segments, while the last one has only 25 pieces. While the distribution of Max-MCT is quite average. All the computers have nearly 37 clips. Our algorithm makes the powerful computers have relatively fewer sub-tasks and fewer task-launching overheads. As the computers are heterogeneous, this means powerful computer must handling those complex segments.

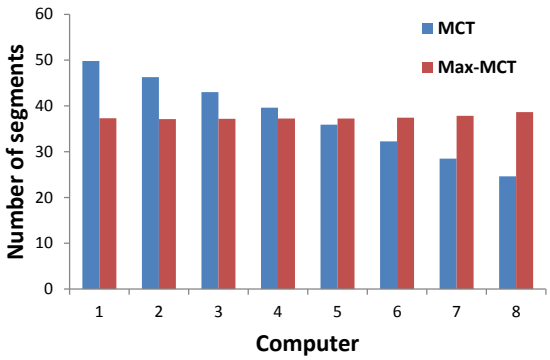


Fig. 5. The number of segments in different computers

VI. Conclusion

In this paper, we investigate the fast transcoding problem and present a Map-Reduce-based cloud transcoding system. We further formulate the transcoding process in the cloud as an NP-hard problem. To reduce complexity, we propose a heuristic algorithm named Max-MCT with two procedures. The virtual knapsack procedure assigns complex segments into powerful computers to reduce their capacity waste on overheads. And the MCT procedure tends to average the finish time of the computers. We also conduct various simulation experiments to verify that our algorithm outperforms the exiting algorithms.

Acknowledgment

This work was supported by National Basic Research Program (973) of China under contract No.2009CB320907, National Development and Reform Commission High-tech Program of China under Grant No. [2010]3044, National Science Foundation of China and Microsoft Research Asia under 60833013.

References

[1] D. Barbosa, J. Kitajima, and J. Weira, W., "Parallelizing mpeg video encoding using multiprocessors," in *Computer Graphics and Image Processing*, Campinas, Brazil, Oct 1999, pp. 215–222.

[2] Y. Chen, E. Li, X. Zhou, and S. Ge, "Implementation of h.264 encoder and decoder on personal computers," *J. Visual Commun. Image Represent.*, vol. 17, no. 2, pp. 509–532, 2006.

[3] B. Jung and B. Jeon, "Adaptive slice-level parallelism for h.264/avc encoding using pre macroblock mode selection," *J. Visual Commun. Image Represent.*, vol. 19, no. 8, pp. 558–572, 2008.

[4] Y. Sambe, S. Watanabe, D. Yu, T. Nakamura, and N. Wakamiya, "High-speed distributed video transcoding for multiple rates and formats," *IEICE Trans.*, vol. E88-D, no. 8, pp. 1923–1931, 2005.

[5] Z. Tian, J. Xue, W. Hu, T. Xu, and N. Zheng, "High performance cluster-based transcoder," in *ICCASM*, Taiyuan, Oct 2010, pp. V2 48–52.

[6] Z. Huang, C. Mei, and L. L. W. T., "Cloudstream: Delivering high-quality streaming videos through a cloud-based svc proxy," in *INFOCOM*, April 2011, pp. 201–205.

[7] T. Braun, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[8] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time cpu scheduling for mobile multimedia systems," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, p. 149, 2003.

[9] S. Sadjadi, S.M.and Shimizu, J. Figueroa, R. Rangaswami, J. Delgado, H. Duran, Collazo-Mojica, and X.J., "A modeling approach for estimating execution time of long-running scientific applications," in *IEEE Int'l Symposium on Parallel and Distributed Processing*, Apr 2008, pp. 1–8.