Parallelizing Video Transcoding With Load Balancing On Cloud Computing

Song Lin[1,2] and Xinfeng Zhang[2,3]
[1]School of Computer & Information Engineering
Peking University Shenzhen Graduate School
Shenzhen, China
{slin,xfzhang}@jdl.ac.cn

Qin Yu[2],Honggang Qi[3]and Siwei Ma[2]
[2]Institute of Digital Media, Peking University, Beijing, China
[3]Universityof Chinese Academy of Sciences, Beijing, China
{qyu, swma}@pku.edu.cn

***Abstract*—Cloud computing is emerging as a very promising technology for computing and storage services. However, the multi-resources load balancing over heterogeneous cluster or cloud is a NP-hard problem. To obtain an optimized solution, in this paper, we propose a heuristic algorithm named Minimum Longest Queue Finish Time(MLFT). In the proposed scheme, we first divide the high computation task into multiple sub-tasks, and then- re-organize all the tasks into multiple task queues to shorten the entire finish time of all the tasks submitted to the cluster and launched in parallel according to load balancing. In the task division process, an adaptive segmentation algorithm is proposed according to the complexity and maximum segmentation granularity of the input task. Based on the proposed algorithm, an efficient parallel video transcoding framework with cloud computing is presented. Experimental results show that the proposed algorithm outperforms the existing algorithms significantly on the entire finish time of the tasks and approaches to the optimal solution closely.**

I. INTRODUCTION

Along with the popularity of digital sets such as digital cameras, digital video recorders and mobile phones, the amount of video data with different formats available on the internet has been increasing explosively. At the same time, cloud computing is emerging as a very promising technology for video computing and storage services over the internet. Therefore, cloud based transcoding and streaming come to be an efficient solution for internet video services due to its highly parallel computation capability.

Cloud computing consists of a cluster of loosely connected computers working together. Each computer can run its own instance independently and all the computers together can provide powerful parallel computing capability. Since computers can be heterogeneous, the cloud computing is extendable and relatively inexpensive.

In order to improve the efficiency of cloud computing, the authors in [1] proposed the classical distributed cloud computing programming model named Map/Reduce. In this model, the computing efficiency can be improved via load balancing on all the computers, which need split the input data into many splits and handle them in distributed computing system. But load balancing is a great challenge in distributed computing system. A well known batch scheduling algorithm on computing cluster is First Come First Serve (FCFS) where tasks are processed in the order of arrival. Each task specifies the number of processors it requires and is placed in a FIFO queue upon arrival [2]. The FCFS scheduling algorithm is easy to implement, but ignores the load balancing problem. Generally, the load balancing problem has been proved to be NP-hard [3]. In [4], eleven heuristic algorithms for mapping a class of independent tasks onto heterogeneous distributed computing system had been compared and the performance of Min-min is the best one. For video transcoding, Minimal Complete Time (MCT) algorithm is equal to Min-min algorithm when the execution time is in proportion to segment complexity [6].

There are also several other efforts devoted to parallel transcoding on multi-core processor or cloud computing, such as [5] [6]. In [5], Huang *et. al.* proposed a cloud-based proxy that can transcodes video in real time. They formulated the transcoding process as an on-line scheduling problem and provided two mapping options to reduce the transcoding jitters and optimize transcoding speed. Unfortunately, they did not consider the sub-task launching overhead. A parallel transcoding system using Map/Reduce based on cloud computing was proposed in [6]. However，only the case of single-tasking is considered and the segmentation scheme is not adaptive to the input task.

In this paper, we focus on optimizing the entire finish time for batch of transcoding tasks. Considering the characteristics of video transcoding and load balancing, we propose a novel parallelizing video transcoding framework. This framework includes three modules: task pre-analysis, adaptive threshold segmentation and minimal finish time (MFT) scheduling. Based on the complexity and the maximum segmentation granularity of the input video, these tasks are divided into sub-tasks, and then all the tasks are launched to the cluster based on load balancing strategy. The adaptive threshold segmentation and the MFT scheduling constitute a novel heuristic algorithm, named minimum longest queue finish time algorithm (MLFT).

The rest of this paper is organized as follows. In Section II, a brief introduction to the system architecture and problem formulation is provided. In Section III, the proposed minimum longest queue finish time algorithm is detailed, including adaptive task splitting and the minimal finish time scheduling. Finally, experimental results are given in Section IV.

II. PROBLEM FORMULATION AND SYSTEM ARCHITECTURE

A. *Problem Formulation*

In general, the entire running time of one task consists of the task-launching overhead and execution time. For video transcoding, the task-launching overhead is less compared to the execution time. So the task-launching overhead of each task can be defined as a constant T_{delay}. We assume that one batch of tasks J consists of n independent tasks. Each task j in J has the different complexity, denoted as C_j ($j=1, 2,\dots, n$).The cluster has m computing cores with different computing capacity P_k ($k = 1,2,\dots,m$). The execution time of computing core is proportional to the task complexity and inversely proportional to the computing capacity. So the transcoding time spent for task j on core k can be calculated as:

$$t_{jk} = C_j \Big/ P_k + T_{delay} \qquad (1)$$

In the computing period, each computing core will handle several tasks. We denote the set of tasks on computing core k as S_k . Then the overall completion time of set S_k is

$$^{1}T_{S_k} = \sum_{j\in S_k} t_{jk} = \sum_{j\in S_k} \frac{C_j}{P_k} + |S_k| \times T_{delay} \qquad (2)$$

So the problem can be formulated as an optimal scheduling problem. Our scheduling goal is to minimize the overall completion time as follows,

$$\min_{L} \max_{S_k \in L} T_{S_k} \qquad (3)$$

Where $L=\{S_1, S_2,\dots, S_m\}$is the scheduling strategy.

If the capacities of computing cores are identical and the task launching overhead is zero, this problem is known to be the load balancing problem. As mention above [6], the load balancing problem has been proved to be NP-hard, so our scheduling problem is NP-hard. To find the optimal solution by traversing all possible solutions, the complexity of solution will be $O(m^n)$.

The optimal solution of the problem is to make the running time of each computing core exactly identical. So the theoretical optimal solution of entire finish time $T_{ld\text{-}ideal}$ is

$$T_{ld-ideal} = \sum_{j=1}^{n} C_j \Big/ \sum_{k=1}^{m} P_k + n \times \frac{T_{delay}}{m} \qquad (4)$$

For the initial task set, due to the dispersion of C_j and P_k, even the optimal solution by traversing all possible solutions is usually far from the theoretical optimal solution. For heterogeneous cluster and non-zero task-launching overhead, our motivation is to balance the load of each computing core. Since some tasks of video transcoding can be divided into lots of sub-tasks, we can achieve better load balancing by task segmentation. However, due to the additional overhead from task segmentation, the segmentation scheme of each task is a great challenge.

B. *System Architecture*

The flow chart of our proposed parallelizing video transcoding method on cloud is shown in Fig. 1. The framework mainly includes three modules: task pre-analysis, adaptive task segmentation and load balancing scheduling.

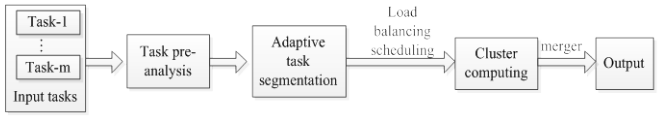

Figure 1. System flowchart

For video coding, if the input sequence has instantaneous decoder refresh (IDR) frame, this video coding task can be divided into sub-tasks [7]. For complexity estimation of video transcoding tasks, the existing algorithms [8] [9] can be utilized, and we won't talk more here and start with task segmentation. The complexity and maximum segmentation granularity which equal to the number of IDR frames of each task are obtained in the pre-analysis procedure. Based on the number of IDR frames and complexity of tasks, the splitter would divide each of them into several video segments at the limit of the segmentation threshold, and then each segment is regarded as a new task. The complexity of each new task can also be obtained using above estimating algorithm. After that, all segments and undivided tasks are submitted to the cluster.

The cluster consists of many computing nodes, and each node has several computing cores with different computing power. The computing capacity of each core can be estimated and normalized by the historical video transcoding record. All the tasks are restructured into m scheduling queues using MFT algorithm based on load balancing strategy. Each task queue is mapped to a specific computing core. The tasks in the queue are processed sequentially, without preemption. When all the computing cores finish their tasks, the batch of tasks are entirely finished. To launch a task, computing nodes have to obtain the input files from the distributed file system and prepare the input data. We denote the sum of them and other overhead as a constant task launching overhead T_{delay}.

After all the computing cores finish their tasks, for all divided tasks, all the streams segments are available on the cluster. The merger downloads these segments and concatenates them together to provide the output.

III. MINIMUM LONGEST QUEUE FINISH TIMEALGORITHM

In this section, we propose a novel heuristic algorithm, named minimum longest queue finish time. The algorithm is based on the load balancing strategy, and mainly consists of two procedures: adaptive multi-threshold segmentation and minimal finish time algorithm. Before discussing the algorithm, the complexity and the maximum segmentation granularity of all the input tasks are obtained in the pre-analysis stage. In the adaptive task segmentation procedure, the segmentation schemes of highly complexity tasks which can be segmented are decided. The number of divided segments depends on the maximum segmentation granularity of original task and the comprehensive influence of the segmentation scheme on the entire finish time. After that, all the tasks are arranged in descending order based the complexity of each task. Then, we employ MFT algorithm for the sorted tasks to average the entire finish time of the cores, so that we can minimize the entire finish time.

A. *Adaptivethreshold Segmentation*

After the pre-analysis of each input task, the input task set is divided into two task sets: one is divisible, the other is indivisible. Based on the complexity of task, the threshold of segmentation C_{thr} is defined as:

$$C_{thr} = \sum_{j=1}^{n} C_j \Big/ (m \times k_{th}) \tag{5}$$

Where k_{th}= (*1, 2, …,* k_{thmax}), and the k_{thmax} is a user data, the value of which is defined by the user. C_j is the complexity of task *j*. *n* is the number of task. *m* is the number of computing cores in the cluster. To all initial input tasks, if the complexity of segmentable task is larger than C_{thr}, the task will be divided into sub-tasks sequentially. For video transcoding, the picture data between each two IDR frames are defined as a basic unit. If the complexity of basic unit is greater than C_{thr}, the basic unit would be regarded as a new sub-task. Otherwise, each sub-task contains one or multiple basic units and the complexity of each sub-task should be less than and as close as possible to C_{thr}. Fig.2 shows an example of task segmentation.

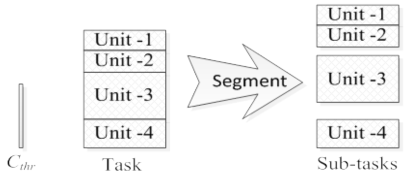

Figure 2. The schematic diagram of task segmentation, the height of graphics indicates complexity, the task is divided into 3 sub-tasks.

As the data dependence doesn't exist between the basic units for video transcoding, these segments can be handled independently. When the splitting procedure is finished, the overall completion time of new task set can be obtained using the MFT algorithm in the simulator. The optimal scheduling scheme could be selected by traversing all values of k_{th} in the simulator. Then, the initial task set *J* is handled based on the optimal scheduling scheme. The adaptive multi-threshold segmentation is shown in Fig. 3.

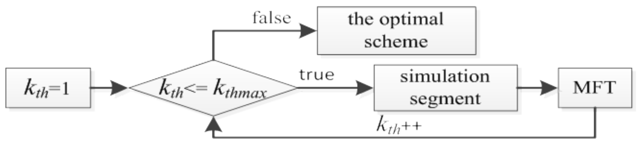

Figure 3. Adaptive multi-threshold segmentation

B. *Minimal Finish Time(MFT)*

For a specific segmentation threshold C_{thr} (k_{th} = *ns*), there exists a generated task set, which is defined to be Z_{ns}. Then the theoretically optimal solution of the entire completion time of Z_{ns} is:

$$T_{ns-ideal} = \sum_{j=1}^{N} C_j \Big/ \sum_{k=1}^{m} P_k + N \times \frac{T_{delay}}{m} \tag{6}$$

N is the number of tasks. *m* is the number of cores and P_k is the computing capacity of each computing core k,(*k=1, 2,…,m*).

First, we select out *s* tasks which have the highest complexity from the set Z_{ns}, and *s* also is user data. The traversal algorithm with pruning is adopted to obtain the optimal solution of this sub-set. As the optimal solution $T_{opt\text{-}s}$ of this sub-set will not exceed the optimal solution of the whole set. T_{opt} is set to be max{ $T_{ld\text{-}ideal}$, $T_{opt\text{-}s}$ }, and T_{opt} will be less than or equal to the optimal solution of set *J*.

Before the tasks of set Z_{ns} are assigned, they are sorted in descending order according to their complexity by Quicksort algorithm. According to (2), the finish time T_κ of core *k* is

$$T_k = \sum_{j \in S_k} \frac{C_j}{P_k} + |S_k| \times T_{delay} \tag{7}$$

In order to minimize the overall completion time, the load on each core should be balanced, which means that all the computing cores should finish their tasks as simultaneously as possible. For each computing core, we establish a scheduling queue. After that, the tasks of set Z_{ns} which have been sorted sequentially are assigned to these cores using a greedy like algorithm. T_{thr} is defined as the threshold of overall completion time and its initial value is set to be max {$T_{ns\text{-}ideal}$, $T_{opt\text{-}s}$ }. The assigning principle is as follows:

1. Assign all the tasks sequentially in descending complexity order. (For each unassigned task, the core with higher computing capacity has priority over other cores.)

2. For each unassigned task *j*, the cores are judged in their descending computing capacity order according to the following criterion: assuming the task *j* is assigned to core *k*, if $T_\kappa \leqslant T_{thr}$, the assignment is verified. Otherwise, we will judge the next core.

3. If all the cores are traversed and all the computing time are beyond T_{thr}, the task *j* will be assigned by MCT algorithm. and T_{thr} is updated to be the new finish time of the received core T_k.

After all the tasks have been assigned, we select out the longest and shortest queues. All the tasks of the two queues are redistributed using the MCT algorithm. If the difference between the two new queues is less than the original difference, repeat the above redistribution step. Otherwise, the current scheduling scheme is regarded as the final scheme. Fig. 4 shows the flowchart of MFT algorithm.

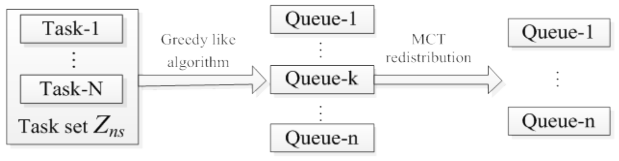

Figure 4. The flowchart of MFT algorithm

IV. Experiment Results

In order to evaluate the efficiency of the proposed algorithm, simulations are conducted using C/C++ program. A heterogeneous cluster with 50 computing cores is designed. The computing capacity of each core ranging from 1.0 to 3.0 is randomly generated. The complexity and the maximum segmentation granularity of each task are also randomly generated. The complexity ranges from 15 to 3600 and the maximum segmentation granularity ranges from 1 to 150. The maximum of k_{th} and s are set as 20 and 8 respectively. For each situation, 500 experiments are carried out and the average of entire finish times is employed as the average completion time T_f. The task-launching overhead T_{delay} is set to be 20 seconds.

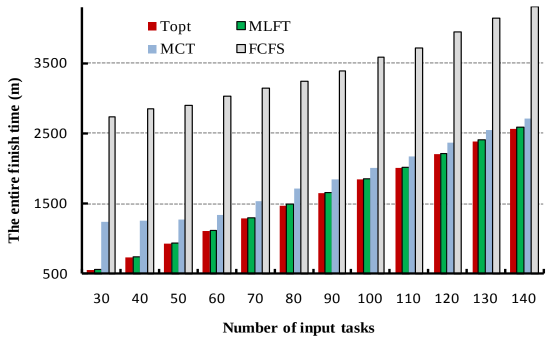

Figure 5. The average completion time T_f of different algorithms

The factor E is employed to evaluate the efficiency of different algorithms. E is denoted as follows.

$$E = \frac{T_f - T_{opt}}{T_{opt}} \times 100\% \quad (8)$$

TABLE I. The Performance Comparison Between MLFT, FCFS and MCT

Number of Tasks	FCFS	MCT	MLFT
30	403.05	124.14	4.18
40	295.31	71.48	2.16
50	222.94	39.41	1.68
60	177.10	20.75	1.61
70	145.82	19.36	1.22
80	121.27	16.48	1.06
90	105.21	12.00	0.97
100	94.14	9.09	0.89
110	85.33	8.64	0.83
120	77.55	7.93	0.76
130	72.48	6.51	0.70
140	68.12	5.61	0.67

Fig. 5 and Table I show the performance comparisons between the proposed algorithm and other two algorithms while the number of input tasks ranges from 30 to 140. It is observed that our scheme outperforms the others in all cases. Moreover, the MLFT still maintains excellent performance when the number of the tasks is fewer than that of cores while the MCT algorithm performs badly.

To further examine the influences of the task-launching overhead time, we conduct an experiment to evaluate the entire finish time under different task-launching overhead. The experimental results are shown in Fig. 6 with T_{delay} varying from 20seconds to 200 seconds. The number of input tasks is set as 40, 60 and 80. The other parameters are set to be the same as mentioned above. It shows that the task-launching overhead has little influence on the entire finish time since the number of divided segments adaptively decreases as the task-launching overhead increasing.

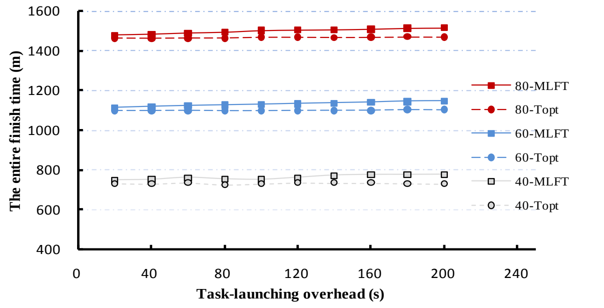

Figure 6. The performance with different task-launching overhead time

V. Conclusion

In this paper, cloud computing based multi-tasking video transcoding is investigated and a parallelizing video transcoding framework is proposed based on load balancing strategy. First, we formulate the multi-resources transcoding process in the cloud as a NP-hard problem. Then a heuristic algorithm called MLFT is proposed to minimize the entire finish time. Experimental results demonstrate that the proposed algorithm is more effective than the existing algorithms, and approaches to the optimal solution closely. In addition, it can be observed that the proposed algorithm has strong robustness to the task launching delay.

Acknowledgment

This research is supported by the 973 program (2009CB320903), the National Science Foundation of China (61121002, 61103088), the 863 program (2012AA011505) and National Key Technology R&D Program (2011BAH08B01), which are gratefully acknowledged.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters". Proc. of the 6th Symp. On OSDI, pp. 137-150, 2004, USENIX Association, 2004.

[2] D. G. Feitelson, L. Rudolph and U. Schwiegelshohn, "Parallel Job Scheduling - A Status Report." in 10th Workshop on Job Scheduling Strategies for Parallel Processing, 2004, pp. 1-16.

[3] R. L. Graham, L.E. Lawler, J. K. Lenstra and A. H. Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling:A Survey," Annals of Discrete Math, pp. 287-326, 1979.

[4] T. Braun, H. J. Siegel and N. Beck, "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.

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

[6] F. Lao, X. G. Zhang, and Z. M. Guo, "Parallelizing Video Transcoding Using Map-Reduce Based Cloud Computing," In ISCAS, Seoul, May2012, pp.2905-2908.

[7] T. Wiegand, G. Sullivan, G. Bjøntegaard and A. Luthra, "Overview of the H.264/AVCvideo coding standard," in IEEE Trans. Circuits Syst.Video Technol., vol. 13, no. 7, pp. 560–576, Jul. 2003.

[8] L. Su, Y. Lu, F. Wu, S. P. Li, and W. Gao, "Complexity-constrained H.264 video encoding," IEEE Trans. Circuits Syst. Video Technol, vol.19, no. 4, pp. 1–15, Apr. 2009.

[9] S. M. Sadjadi, S. Shimizua and J. Figueroa,etc, "A modeling approach for estimating execution time of long-running scientific applications," in IEEE Int'lSymposium on Parallel and Distributed Processing, Apr 2008, pp. 1–8.