Job Scheduling for Acceleration Systems in Cloud Computing

Yangming Zhao, Xin Liu and Chunming Qiao
Department of Computer Science and Engineering
State University of New York at Buffalo

***Abstract*—With the increase of various of applications, CPU is no longer adequate for the computation tasks. Accordingly, some providers deploy accelerators in their cloud. Since not all the servers in the cloud can carry accelerators, how to schedule jobs onto accelerators and improve the system performance is important. Due to the distributed computing frameworks in cloud computing systems, the jobs usually arrive in batches, and hence we try to minimize the make-span of a batch of jobs in this paper. To this end, we first formulate this problem as a mathematic programming model, and prove the NP-hardness of this problem. To solve this problem efficiently, we propose a 4-approximation algorithm. Through extensive simulations, we find that our algorithm can reduce the make-span of a batch of jobs by about 32%, and enhance the system throughput by up to 29% compared with our comparison baseline.**

I. Introduction

With the development of big data and Virtualized Network Function (VNF), it is difficult to deal with the computation tasks on local servers. Accordingly, more and more people outsource their applications to the public cloud. This increases the computation workload in the public cloud and the universal CPU is no longer adequate for many applications. For example, Deep Neural Networks (DNN) [1] needs many iterations and a CPU needs a long time to execute such application. Furthermore, for the applications like AES encryption and Deep Packet Inspection (DPI), even though universal CPU can deal with them in a reasonable time, it may lead to a large latency and bring performance degradation to the customer experience.

To solve this problem, some leading cloud providers, such as Baidu and Microsoft, start to integrate accelerators (implemented by FPGA, GPU, ASIC chips etc.) into their platforms [2, 3], since the accelerators have the properties of high speed and high energy efficiency [3]. However, due to the cost issue and deployment constraints, the cloud providers cannot install accelerators on all of their servers in the cloud. Accordingly, how to manage the accelerator resources and schedule the jobs in the cloud in order to improve the system performance and provide better customer experience is a critical problem. In the field of accelerator resource management, [4] and [5] are two representatives. Both of them mapping resources to different accelerators based on their cost model. In [4], the cost model is based on resource, while it is based on the expected reconfiguration time and throughput in [5]. As far as we know, there are no existing works on the job scheduling problem in the accelerator system.

In the public clouds, the application requests usually arrive in batches. This is mainly due to two reasons. First, many of the parallel computation platforms, such as Hadoop [6], Spark [7], involve multi-tier split-aggregation workflow. In these platforms, a single application request may be divided into a large number of computation tasks. In addition, there are some applications itself containing many small jobs, since a single job is so tiny that the communication overhead may dominate the outsourcing procedure. For example, to deal with a DNN application for one photo, it is not necessary to outsource it to a public cloud. Usually, a DNN application request contains thousands of photos to offset the communication overhead.

When the jobs arrive in batches, a reasonable objective is to minimize the make-span of such a batch of jobs. For these applications contains many small jobs, minimize the make-span can speed up the responding to the tenant and hence improve the customer experience. There are lots of existing works to minimize the make-span of a batch of jobs. [8] may be the first work that studies the job scheduling on unrelated machines and proposed an efficient algorithm with approximation ratio 2. However, it did not consider the time to transmit a job to the machine which it is assigned to. A case in classic scheduling problems similar to ours is that each job has a release time [9], and each job can start processing after the release time. This setting is also not suitable to our problem as the time to transmit a job is not fixed, but determined by the available network capacity. As far as we know, there is not existing work that focuses on both of the job transmission and computation.

The main contribution of this work can be summarized as follows:

- Formulate the problem to minimize the make-span of a batch of jobs for acceleration and analyze the problem complexity (Section III)
- Propose a 4-approximation algorithm to solve the problem (Section IV)
- Extensive simulations to show that our algorithm can not only reduce the make-span of a batch of jobs, but also enhance the system throughput if there are batches of jobs arriving continuously (Section V)

II. System Model

In our system, some of the hosts are carrying accelerators to process the jobs for acceleration purpose; we call such hosts

978-1-5386-3180-5/18/$31.00 ©2018 IEEE

as *responders*. Jobs are issued by the customers and should be assigned to accelerators for processing. Since there are different types of jobs needing accelerating, each responder may carry heterogeneous accelerators, i.e. each job may need different processing time on different accelerators. Formally, say t_{ij} is the time required by responder j to process the job i, there would be $t_{ij_1} \neq t_{ij_2}$ if $j_1 \neq j_2$.

All the hosts are connected through a communication network. As most of the data center topologies, such as Fattree [10] and VL2 [11], are nonblocking, we assume the communication bottleneck exists only at the NIC of each host. Furthermore, since the accelerators are more expensive than the conventional CPU, and there may be deployment issues, only part of hosts in the system can carry accelerators and perform as responders. Therefore, we further assume the communication bottleneck only exists at the responders' NIC.

When a batch of jobs arrive, a centralized scheduler aggregates all the job information, such as the job type and size, and determines the job assignment and the order of job processing on each responder. It should be noted that a job should be first delivered to a responder, and then processed by the accelerator carried by the responder. Accordingly, the job completion time contains two parts: transmission and computation. The transmission phase to assign jobs to different responders can be overlapped, and so is the transmission phase and computation phase of different jobs. However, the computation phase of different jobs assigned to the same accelerator cannot be overlapped. Hereafter, we refer *transmission time* and *computation time* to the time for these two phases, respectively. In addition, *execution time* refers to both phases, i.e. the time starts from the job transmission to the end of computation.

III. Formulation and Analysis

When a batch of jobs arrive, the centralized scheduler minimizes the make-span of all the jobs in this batch by assigning jobs to different responders and determining the order to process these jobs. Assume x_{ij}^k is the binary variable to denote if job i is the k^{th} job to be executed on responder j, t_{ij} is the time required by responder j to process the job i, s_i is the size of job i, and r_j is the incoming rate of responder host j, the completion time of the k^{th} job on responder j (denoted as $c_{j,k}$) should satisfy

$$c_{j,0} \geq \frac{\sum_i s_i x_{ij}^0}{r_j} + \sum_i t_{ij} x_{ij}^0 \quad \forall j \tag{1}$$

$$c_{jk} \geq \max\{c_{j,k-1}, \frac{\sum_i \sum_{l<k} s_i x_{ij}^l}{r_j}\} + \sum_i t_{ij} x_{ij}^k \quad \forall j, k > 0 \tag{2}$$

The first constraint is used to calculate the completion time of the first job on each responder. $\frac{\sum_i s_i x_{ij}^0}{r_j}$ is the time to transmit the job input data, and the job cannot be executed utill the job transmission is completed. The second constraint is for the completion time of jobs but not for the first one. For the k^{th} job on responder j, it can start processing when 1) the $(k-1)^{th}$ job on responder j completes, and 2) all the data for the first k jobs are transmitted to responder j. Accordingly, it can start at time $\max\{c_{j,k-1}, \frac{\sum_i \sum_{l<k} s_i x_{ij}^l}{r_j}\}$.

In addition, there is an inherent constraint that each job can be assigned to a unique responder and placed on a certain order for processing. Therefore, we have

$$\sum_j \sum_k x_{ij}^k = 1 \quad \forall i \tag{3}$$

If T is the make-span of all the jobs, then

$$c_{jk} \leq T \quad \forall j, k \tag{4}$$

In summary, the problem to minimize the make-span of a batch of jobs can be formulated as

$$\text{minimize} \qquad T$$

subject to

$$(1) - (4)$$

$$x_{ij}^k \in \{0, 1\}$$

This formulation is difficult to solve due to two reasons: 1) the number of jobs assigned to each responder is not previously known, i.e. the number of constraints contained in (2) is varying with the solution; 2) the x_{ij}^k is binary variable, which makes the problem to be non-convex optimization. Actually, this problem is NP-hard.

Theorem 1. *The problem should be solved in our work is NP-hard.*

Proof: As in [12, 13], we prove this theorem by constructing a special case of this problem, which is NP-hard. By assuming all the jobs are very small, i.e. the data transmission phase can be ignored, the problem becomes how to assign jobs to parallel unrelated machines in order to minimize the make-span. This is the well-known unrelated parallel machine scheduling problem [8], which is NP-hard. Accordingly, our problem is also NP-hard. ■

IV. Algorithm Design

As we discussed above, the problem that should solve in our work is NP-hard. Therefore, we propose an efficient heuristic to solve it in this section. The key idea of our heuristic is to first assign the jobs without considering the transmission phase and computation phase of different jobs can be overlapped on a host (which is to determine the allocation of each job), and then determine the processing order of jobs assigned to the same responder by pursuing the maximum overlap (and hence minimize the completion time of the last job on each responder host).

A. Job Assignment

Under the assumption that the transmission phase and computation phase of different jobs cannot be overlapped, the execution time of job i on responder j can be calculated as

$$p_{ij} = \frac{s_i}{r_j} + t_{ij} \tag{5}$$

The first term is the time used to transmit job i to responder j, while the later term is the computation time for responder j to process job i. In this case, the problem to minimize the job make-span can be formulated as following Make-span Minimization Problem without Overlap (M2PO):

$$\text{minimize} \qquad T \tag{6}$$

subject to

$$\sum_i p_{ij} y_{ij} \leq T, \quad \forall j \tag{6a}$$

$$\sum_j y_{ij} = 1, \quad \forall i \tag{6b}$$

$$y_{ij} \in \{0, 1\}, \quad \forall i, j \tag{6c}$$

In M2PO, y_{ij} is a binary variable and used to indicate if job i is assigned to responder j. Since the transmission phase and the computation phase cannot be overlapped, it is not necessary to care the execution order of jobs on a specific host. The objective is to minimize the make-span of all the jobs, denoted by T. Constraint (6a) says that any responder j should complete all the jobs assigned to it before the make-span T. Constraint (6b) and (6c) indicate that any job i must and can only be assigned to one responder.

M2PO is the classic unrelated parallel machine scheduling problem [8]. Though it is still NP-hard, it can be solved with relaxation and round method. First, we relax M2PO to be

$$\text{minimize} \qquad T \tag{7}$$

Subject to:

$$\sum_{i \in E_j(T)} p_{ij} y_{ij} \leq T, \quad \forall j \tag{7a}$$

$$\sum_{j \in H_i(T)} y_{ij} = 1, \quad \forall i \tag{7b}$$

$$y_{ij} \geq 0, \quad \forall i, j \in H_i(T) \tag{7c}$$

where $E_j(T)$ is the set of jobs that can be completed on responder j in time T, and $H_i(T)$ is the set of responder hosts that can complete job i in time T. It should be noted that *above relaxation forbids a job i to be assigned to a responder that cannot complete it before T even if only job i is assigned to this responder.*

When T is a variable, problem (7) is still difficult to solve since the number of terms in each constraint is varying with the value of T. However, by fixing T, all the constraints become linear and there are efficient algorithms to check the problem feasibility. Accordingly, we can use the binary search to find the minimum T that can make (7) feasible, which is the objective value of (7). However, according to the solution of (7), a job may be split to multiple responders. In practice, we need to assign each job to a unique responder.

Fortunately, it has been shown that if there are J jobs and R responders in the system, there are at most $(J+R)$ variables (i.e. y_{ij}) strictly larger than 0 in the solution of problem (7) [8]. Accordingly, we can construct a forest of pseudo trees (a tree or a tree plus one more edge) as follows: we first construct a bigraph $BG = \{U, V, E\}$ based on the solution of (7), y; $U = \{u_1, u_2, \ldots, u_R\}$ is the set of nodes denoting responder, called responder nodes, while $V = \{v_1, v_2, \ldots, v_J\}$ is the set of nodes denoting jobs, called job nodes; there is an edge between v_i and u_j, if $y_{ij} > 0$. If BG is connected, it must be a pseudo tree as there are $J+R$ nodes and at most $J+R$ edges in BG. Otherwise, say P is a connected part of BG, we can use the jobs and responders associated with P to solve (7) again and modify the edges based on the new solution. In this case, P will become a pseudo tree or multiple disconnected parts. If it becomes multiple disconnected parts, we repeat above procedure until all the connected parts become pseudo tree. Apparently, such iteration will not increase the make-span of all the jobs.

On BG, we can first deal with the job nodes with only one node degree. These jobs can be assigned to a unique responder according to the solution of (7). After that, every remaining job is split to multiple responders. For every connected part in BG, if it is a tree, we can root the tree at arbitrary job node, and assign each job to arbitrary one of its child responders as shown in Fig. 1(a). If it is a tree plus one more edge, we can round the job assignment as in Fig. 1(b). First, we find the circle in that part with Depth First Searching (DFS), and assign the jobs on that cycle to ensure every responder on that cycle gets only one job. Then, by removing the job nodes and responder nodes on the cycle, a forest of trees is left. We can again root these trees at any job node and assign every job to its arbitrary one child responders. With above job assignment, we can ensure that *every responder only gets one split job according to the solution of* (7). Based on above discussions, we can summarize the algorithm to derive the job assignment as Algorithm 1.

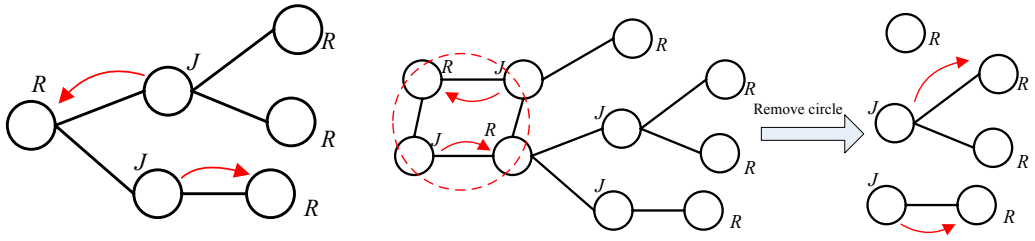
Theorem 2. *The approximation ratio of Algorithm 1 is 2.*

Proof: Say T' is the optimal objective value of (7), $\hat{T}$ is the make-span derived by Algorithm 1 and T^* is the objective value of M2PO. Apparently, there is $T' \leq T^* \leq \hat{T}$. Without considering the jobs that should be split onto multiple responders according to the solution of (7), the make-span should be less than T'. In addition, Algorithm 1 ensures that every responder j only gets at most one split job and it can complete this job in T'. Therefore, the make-span after rounding is at most $2T'$, i.e. $\hat{T} \leq 2T'$. Then, we conclude that $T' \leq T^* \leq \hat{T} \leq 2T'$, which also indicates $\hat{T} \leq 2T^*$. ■

B. Job Scheduling on Specific Responder

In last subsection, we assign the jobs to responders by assuming the transmission phase and computation phase cannot be overlapped. To overlap the transmission phase and computation phase, and minimize the time to complete all the jobs assigned to a responder, we propose following theorem.

Theorem 3. *Suppose $A_1, A_2, \cdots, A_k$ are the jobs assigned to a responder, scheduling all the jobs with the increasing order of $\{\frac{s_i}{r}\}$ can minimize the time to deal with all the jobs on this responder, where s_i is the size of job A_i and r is the incoming*



(a) Rounding for a tree (rooted at the upper job node).

(b) Rounding for pseudo tree.

Fig. 1. Rounding scheme (R for responder nodes and J for job nodes).

Algorithm 1: Job Assignment

Input: The job execution time updated with (5) $\{p_{ij}\}$

Formulate M2PO and relax it as (7) and solve it with binary search
Construct a forest of pseudo trees based on the solution of (7)
Assing jobs with only one node degree to the connecting responder
for all connected components $P \in BG$ **do**
  if P is pseudo tree **then**
    Find the cycle C in P with DFS
    Assign jobs on C to ensure every responder on C gets only one job
    for all the remaining trees **do**
      Rooting at the unique job leaf node (if there is), or arbitrary job node, and assign each job to one of its child responders
    end for
  else
    Treat arbitrary job node as the root to form a tree and assign each job to one of its child responders
  end if
end for

rate of this responder.

Proof: We prove this theorem by contradiction. Without loss of generality, we assume the optimal job scheduling order is $A_1, A_2, \cdots, A_k$, and it completes all the jobs on the responder in time T. However, it does not follow Theorem 3. In this case, there must be a l, such that $\frac{s_l}{r} > \frac{s_{l+1}}{r}$.

Now, we exchange the scheduling order of A_l and A_{l+1}, and the responder can complete all the jobs in time T'. If $\sum_{i=2}^{l} \frac{s_i}{r} \leq \sum_{i=1}^{l-1} t_i$ (t_i is the computation time of job A_i on this responder), which means the computation time is larger than the transmission time. In this case, the completion time of all the jobs are determined by the computation phase, and hence we have $T = T'$.

If $\sum_{i=2}^{l} \frac{s_i}{r} > \sum_{i=1}^{l-1}$, the gap between the start time of A_l and the completion time of A_{l-1} is $\sum_{i=2}^{l} \frac{s_i}{r} - \sum_{i=1}^{l-1} t_i$ before we modify the scheduling order. By changing the order of A_l and A_{l+1}, the gap between the start time of A_{l+1} and the completion time of A_{l-1} becomes $\max\{\sum_{i=2}^{l} \frac{s_i}{r} + \frac{s_{l+1}}{r} - \sum_{i=1}^{l-1} t_i, 0\}$. As $\frac{s_l}{r} > \frac{s_{l+1}}{r}$, the time to complete all the jobs would reduce at least $\frac{1}{r}(s_l - s_{l+1})$. Accordingly, $T' \leq T$. In other words, we find a scheduling order to reduce the time to complete all the jobs. ■

It is obvious that by overlapping the transmission phase and computation phase, we can reduce the make-span. However, at most, it can reduce the make-span by half compared with the case that we do not consider the overlap. This can be shown by following theorem:

Theorem 4. *By overlapping the transmission phase and computation phase of different jobs assigned to the same responder, we need at least half of the time to complete all the jobs where the transmission phase and computation phase cannot be overlapped.*

Proof: This theorem is intuitive by noting that the completion time should be larger than both of the time to compute all the jobs and the time to transmit all the jobs. At least the time to compute all the jobs or the time to transmit all the jobs should be larger than half of the time to complete all the jobs without overlapping the transmission phase and computation phase. ■

C. Approximation Ratio Analysis

Based on above discussions, we can summarize our algorithm to schedule a batch of jobs in order to minimize their make-span:

- Assign jobs to different responders with Algorithm 1 without considering the overlap of transmission phase and computation phase of different jobs
- Schedule all the jobs assigned to each responder following the Minimum Transmission Time First (MTTF) principle as shown in Theorem 3

Theorem 5. *The algorithm presented above to minimize the make-span of a batch of jobs is 4-approximation.*

Proof: Say T^{opt} is the optimal make-span of our problem, and $T_{non-lap}$ is the make-span derived by following the job assignment and scheduling of the optimial solution without overlapping the transmission phase and computation phase of

different jobs. According to Theorem 3, we have

$$T_{non-lap} \leq 2T^{opt}$$

Assume $T^{opt}_{non-lap}$ is the optimal make-span for the case that the transmission phase and computation phase of different jobs cannot be overlapped, there is

$$T^{opt}_{non-lap} \leq T_{non-lap}$$

According to Theorem 2, if $T^{alg}_{non-alp}$ is the make-span derived by Algorithm 1, we know

$$T^{alg}_{non-alp} \leq 2T^{opt}_{non-lap}$$

After overlapping according to Theorem 3, the make-span derived by our algorithm, T^{alg}_{lap}, should be smaller than $T^{alg}_{non-lap}$. Accordingly, we have

$$T^{alg}_{lap} \leq T^{alg}_{non-lap} \leq 2T^{opt}_{non-lap} \leq 2T_{non-lap} \leq 4T^{opt}$$

■

V. Performance Evaluation

In this section, we are to evaluate the performance of our scheduling system. To this end, we compare the performance of our algorithm to two baselines: 1) Smallest Completion Time (SCT) heuristic. With SCT, we assign each job to the responder that can execute this job quickest (i.e. with minimal execution time), and schedule them on each responder according to Theorem 3; 2) Minimum Make-span Increase (MMI) heuristic, where we myopically select a job that can minimize the make-span increase and put it into the system.

In the simulations, we generate job size following the exponential distribution with the parameter 0.005, i.e. the average job size is 200 Mb. Correspondingly, We set the ingress bandwidth of responder hosts to be 1 Gbps. The computation time of each job is set to be approximately proportional to its size with the ratio of 0.001. In this way, the data transmission and computation of each job spend almost the same time. Since the jobs are randomly generated, each of the point in the simulation results is averaged by 20 tries.

A. Impact of Workload

In this subsection, we study how the make-span changes with the number of jobs in a batch. To this end, we assume there are 50 responders and inject different number of jobs per batch into the system. The simulation results are shown in Fig. 2. From this figure, we can make following observations.

First, our scheduling algorithm outperforms both baselines, while MMI scheme derives a smaller make-span than that derived by SCT. This is because that our algorithm first optimizes the job allocations in a global view, and optimally determines the order to execute jobs that have been assigned to each responder. In SCT, it only considers to reduce the execution time of each job, rather than how one job placement impacts the completion of others. Accordingly, it yields a bad make-span performance. MMI scheme sequentially selects jobs that can minimize the make-span increase and assigns it to the best responder, which optimizes the make-span in a myopic manner but takes more factor into account. Therefore, it outperforms SCT, but performs worse than our scheme.

Second, our scheme and MMI have similar performance when the system is lightly loaded (e.g. only 200 jobs in a batch) and heavily loaded (e.g. 1200 jobs in a batch). When the system is lightly loaded, the make-span is dominated by the few large jobs, and hence both schemes achieve similar make-span. When the system is heavily loaded, the overall make-span is large, if the system is fully utilized, a greedy algorithm can also achieve an asymptotically optimal solution. When there are 600–800 jobs in the system, our algorithm outperforms MMI by about 10%, and outperforms SCT by about 32%.

B. Impact of Responder Number

In this subsection, we fix the number of jobs in a batch as 1000 and vary the number of responders to study how the responder number impacts the make-span. The simulation results are shown in Fig. 3.

Intuitively, more responders can reduce the make-span of a batch of jobs. This is because there will be less jobs assigned to each responder. In addition, we can also observe that the make-span decreases slower with the increase of responder number when our scheduling scheme or MMI scheme is adopted. Again, this is because the make-span is dominated by the few large jobs; more responders cannot help reduce the make-span since a job cannot be split to multiple responders. However, with SCT scheme, the make-span reduces approximately linearly with the increase of responder number. This is due to the fact that SCT does not find a good job assignment scheme and the make-span has not been bottlenecked at the large jobs yet.

C. Benefit to System Throughput

By minimizing the make-span of each batch of jobs, we can corresponding improve the system throughput. This is shown in Fig. 4. In this figure, we change the batch arrival interval and observe how the system throughput changes. The arrival rate is 0 means all the batches arrive simultaneously. We set the maximum arrival interval to the 5s as we need about 4.2s to complete each bath of jobs.

From this figure, we can see that our scheduling scheme can improve the system throughput by about 11% compared with MMI scheme, while it improves the system throughput by 20%–29% compared with SCT scheme, since we enhance the system utilization when we minimize the make-span of each batch of jobs.

When the job batch arrival interval decreases, i.e. more jobs arrive in a time interval, the system throughput increases. This is because the later batch of jobs can utilize the bandwidth/accelerator resources that are released by previous batch of jobs. If a suitable scheduling scheme is adopted (e.g. our scheme or MMI scheme), the system throughput cannot be improved too much when the job batch arrival interval is small (0s or 1s), as the system is fully utilized. This phenomenon

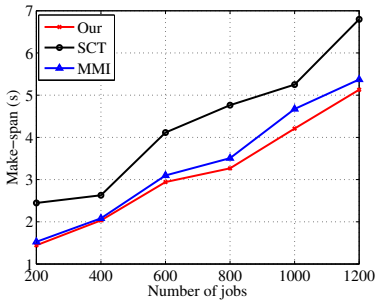


Fig. 2. Impact of workload

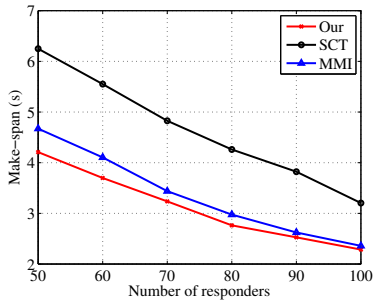


Fig. 3. Impact of responder number

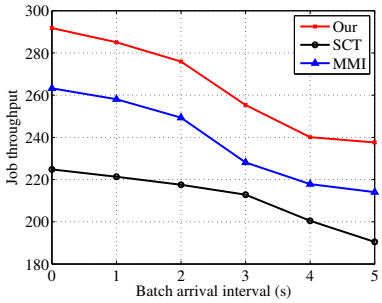


Fig. 4. Benefit to system throughput

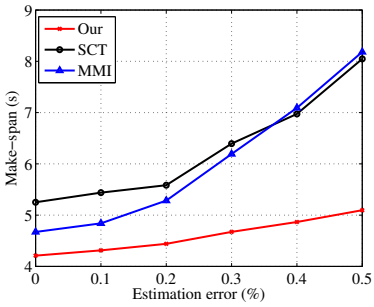


Fig. 5. Impact of estimation error

also appears when the system is lightly loaded, i.e. the batch arrival interval is 4s or 5s, since the later batch arrives when most of the jobs in previous batches are completed, and not many jobs in later batch can be processed before the completion of previous batch of jobs.

D. Impact of Estimation Error

Our scheduling algorithm is based on the estimation of job computation time. However, we cannot obtain an accurate computation time estimation. In this subsection, we study how the estimation error impacts the algorithm performance. To this end, we add some estimation error to the computation time of each job on different responders, and test the make-span achieved based on the inaccurate computation time. There is 10% error means the input computation time is on $[0.9t, 1.1t]$ where t is the real computation time. The simulation results are shown in Fig. 5.

As shown in this figure, the make-span will increase with the estimation error regardless of which scheduling scheme is adopted. However, our scheme is more robust to the estimation error. Even the estimation error is 30%, it results in about 5% of make-span increase. By digging into the results, we found that this is because that estimation error of different jobs assigned to the same host can compensate each other's error impact.

The estimation error has larger impact on SCT than MMI. Though MMI outperforms SCT when we can accurately estimate the job computation time, SCT achieves smaller make-span when the estimation error exceeds 40%. The reason to this phenomenon is that SCT only simply assigns each job to the responder that can execute it fastest, while MMI takes the impact to entire make-span into consideration. The estimation error leads to a wrong estimation to the make-span and hence results in larger impact to the performance of MMI.

VI. Conclusions

This paper studied the job scheduling problem in the acceleration systems. Different from classic scheduling problem, our work considered both of the job transmssion and computation. We first formulated this problem as a mathematic programming problem and proved that the problem studied in this paper is NP-hard. Then, we proposed an efficient algorithm to solve the scheduling problem. In addition, we proved that the proposed algorithm is 4-approximation. Through extensive simulations, we found that our algorithm cannot only reduce the make-span of each batch of jobs, but also enhance the system job throughput.

References

[1] J. Gu, M. Zhu, Z. Zhou, F. Zhang, Z. Lin, Q. Zhang, and M. Breternitz, "Implementation and evaluation of deep neural networks (dnn) on mainstream heterogeneous systems," in *Proceedings of APSys*, 2014.

[2] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *ACM/IEEE ISCA 2014*, pp. 13–24.

[3] J. Ouyang, S. Lin, W. Qi, Y. Wang, B. Yu, and S. Jiang, "Sda: Software-defined accelerator for largescale dnn systems," in *Hot Chips*, 2014.

[4] C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1393–1407, 2004.

[5] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in fpga systems: A survey and a cost model," *ACM TRETS*, vol. 4, no. 4, pp. 36:1–36:24, Dec. 2011.

[6] "Apache hadoop." [Online]. Available: http://hadoop.apache.org/

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the USENIX HotCloud 2010*.

[8] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," in *IEEE FOCS*, Oct 1987, pp. 217–224.

[9] K. Fox and B. Moseley, "Online scheduling on identical machines using srpt," in *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 2011, pp. 120–128.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[11] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," *ACM SIGCOMM CCR*, 2009.

[12] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the ACM SIGCOMM*, 2014, pp. 443–454.

[13] X. Fang, D. Yang, and G. Xue, "Map: Multiconstrained anypath routing in wireless mesh networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 10, pp. 1893–1906, Oct 2013.