In-Switch Approximate Processing: Delayed Tasks Management for MapReduce Applications

Koya Mitsuzuka*, Ami Hayashi*, Michihiro Koibuchi†, Hideharu Amano*, Hiroki Matsutani*
*Dept. of ICS, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan
Email: {koya@arc,hayashi@arc,hunga@am,matutani@arc}.ics.keio.ac.jp
†National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
Email: koibuchi@nii.ac.jp

***Abstract*—In MapReduce, the parallel processing performance is often limited by only a few compute nodes that delay to complete given tasks. Although various techniques have been invented to handle such stragglers, these techniques mostly impose a burden on master node to monitor the progress of all the compute nodes, resulting in a new bottleneck as the number of compute nodes increases. As an alternative approach, in this paper, we propose to move such straggler management burden from master node to network switch that connects the master and compute nodes, because all the information goes through the switch. More specifically, the proposed network switch monitors output packets from Map tasks to detect stragglers. When detected, the proposed switch generates a response instead of the straggler based on the outputs of the other normal Map tasks, so that Reduce tasks can be started without delay. We introduce some approximate techniques for the proxy computation and response at the switch; thus our switch is called "ApproxSW." We implement ApproxSW on NetFPGA-SUME board that has four 10Gbit Ethernet (10GbE) interfaces and a Virtex-7 FPGA. An experiment shows that the ApproxSW functions do not degrade the original 10GbE switch performance. We also analyze the accuracy of the proxy computation and response for stragglers and show that the proposed approximation based on task similarity achieves the best accuracy.**

I. Introduction

As the data sets grow rapidly in size, parallel processing frameworks such as MapReduce [1] are becoming more important. MapReduce consists of two types of nodes: worker nodes (i.e., compute nodes) that perform parallelized tasks and a master node that manages the entire system including worker nodes. MapReduce job is divided into finer tasks and assigned to workers for parallel processing. Although the granularity of tasks depends on applications, in this paper we focus on a fine-grain parallel processing that processes thousands or tens of thousands of Map tasks, each of which finishes in subseconds.

The parallel processing performance is often limited by only a few worker nodes that process given tasks with low performance due to machine troubles and/or excessive workloads. These workers are called stragglers. Although various techniques have been invented to handle the stragglers, they mostly impose a burden on master node to monitor the progress of all the worker nodes. In Backup Task [1], for example, when a straggler is detected, the delayed task is assigned to worker node that has been completing its task faster than the others. Backup Task that reruns delayed tasks on fast worker nodes can always return correct results even with stragglers. The master node is in charge of monitoring all the worker nodes for Backup Task. However, as the number of worker nodes increases, the management overhead of the master node increases, resulting in a new performance bottleneck in massively parallel processing. Because such management tasks by the master node do not contribute to application performance, spending a lot of CPU times for the management is a waste of CPU resources; thus the CPU resources should be devoted for the application performance. Please note that the ratio of delayed tasks over all the tasks is quite small in the case of a large degree of parallelism [2]. Depending on applications, especially for those that do not require exact results, additional costs for Backup Task cannot be justified. Although replication of master nodes can distribute the management overhead, more efficient approach is required.

As an alternative approach, in this paper, we propose to move such straggler management burden from master node to network switch that connects the master and worker nodes, because all the information goes through the switch. More specifically, the proposed network switch monitors output packets from Map tasks to detect stragglers. When detected, the proposed switch generates a response instead of the straggler based on the outputs of the other Map tasks, so that Reduce tasks can be started without delay. We introduce some approximate techniques for the proxy computation and response at the switch; thus our switch is called "ApproxSW." We implement ApproxSW on NetFPGA-SUME board that has four 10Gbit Ethernet (10GbE) interfaces and a Virtex-7 FPGA and demonstrate that the ApproxSW functions do not degrade the original 10GbE switch performance.

The rest of this paper is organized as follows. Section II overviews related work. Section III proposes ApproxSW architecture and Section IV illustrates its implementation on NetFPGA-SUME board. Section V shows experimental results and Section VI concludes this paper.

II. Related work

Backup Task [1] is the conventional solution for the straggler problem in MapReduce framework. Sophisticated scheduling algorithms for Backup Task have been proposed in [3][4]. They focus on detecting stragglers as early and correctly as possible and thus they impose more burden on the master node to collect more detailed progress report from worker nodes. A distributed scheduling algorithm for large-scale clusters is also proposed in [5]. However, it does not discuss how to monitor the tasks in parallel in detail. Although Backup Task can obtain accurate results while mitigating the effect of stragglers, it imposes more burden on the master node and the network to monitor progresses of a number of worker nodes. In addition, speculative rerunning consumes extra power and compute resources. Our ApproxSW is completely different from these prior works but is a natural approach because all the information goes through the switch. A prototype of ApproxSW is demonstrated on NetFPGA-SUME board.

Approximate computing improves the compute performance and the energy efficiency in exchange for acceptable degradation in computation accuracy. ApproxHadoop [6] adopts an approximation technique that consists of input data sampling and task dropping in MapReduce. The input data sampling computes a partial result based on a part of input data and estimates the entire result based on the partial result. Task dropping reduces the computational cost and execution time by dropping some tasks. Especially, tasks that take longer time to complete compared to the others and those that have not been started are dropped, in order to reduce the execution time. Our ApproxSW also employs the dropping technique in the network switch and implements it on NetFPGA-SUME board.

III. Design

This paper proposes ApproxSW, a network switch based solution for the straggler problem to eliminate a burden of master/worker nodes to handle stragglers. In general, a straggler solution consists of two stages: detection and proxy response. A simple network switch based solution is to just detect delayed tasks and request the master node to reschedule them as well as Backup Task. On the other hand, ApproxSW creates proxy responses instead of detected stragglers; thus no computation and communication overheads are imposed in any master/worker nodes to handle stragglers. Although there are Map and Reduce tasks, in this paper we focus on stragglers of Map phase for the proxy response by a network switch.

A. Straggler Detection at Network Switch

ApproxSW monitors Map outputs to check the progress of each task and detect stragglers. More specifically, ApproxSW counts the number of key-value pairs from each Map task, and the task whose counter value is less than θ (Slow Task Threshold) from the average is detected as a straggler. This assumes that each Map task processes almost the same number of keys. The other cases can be also handled by adjusting θ appropriately. When Combiner function that aggregates the Map task results within the Map phase is applied, ApproxSW is modified to monitor the Combiner function to detect stragglers.

B. Proxy Response at Network Switch

ApproxSW does not notify the master node about the detected stragglers but creates proxy responses instead of the stragglers. That is, ApproxSW is in charge of the proxy computations and completion notifications instead of delayed tasks. Completion notification is to inform a completion of a task to master node for starting the next phase. Generating it by proxy omits the waiting time due to stragglers. However, it introduces some uncompleted tasks and degrades the accuracy of the final results. Here, we propose proxy computation for the delayed tasks to compensate the negative impact on accuracy.

Algorithm 1 Similarity function for proxy computation

```
$a \Leftarrow task\ sending\ new\ data$
$key \Leftarrow key\ included\ in\ new\ data$
**for** all the other tasks $i$ **do**
    **if** $i$ has sent $key$ and $a$ sent $key$ for the first time **then**
        $S[a][i] \Leftarrow S[a][i] + 1$
        $S[i][a] \Leftarrow S[i][a] + 1$
    **end if**
**end for**
```

1) Proxy Computation: The proxy computation for stragglers by a network switch is not a trivial job. It is difficult for network switches to access input files and process them accordingly to complete the computation. Therefore, we adopt an approximate computing for the proxy computation to strike a balance between a burden for handling stragglers and the accuracy of the final results. A unique characteristic of network switch based solutions is that the outputs from the other tasks are available at a network switch since the outputs go through the switch. We thus propose to use the outputs from the other tasks for proxy computation. A simple implementation of this approach is to duplicate the outputs of the other tasks. In this paper we introduce the following the three proxy computation methods.

- Drop: No proxy computation.
- Random: Copy the normal task outputs randomly.
- Similarity: Copy the normal task outputs based on the similarity calculated by Algorithm 1.

In the following, the similarity method, the most sophisticated one among them, is introduced.

ApproxSW utilizes Map outputs sent by the other workers connected to the switch in order to generate similar outputs of delayed tasks. Since it is not feasible to store all the Map outputs in the limited memory capacity of the network switch, we propose to use only the recently-arrived Map outputs (called “new data”) and statistical information based on all the past Map outputs. Every time ApproxSW receives new data, it updates the statistical information and then generates outputs of delayed tasks by proxy. A similarity between tasks is used as the statistical information. We employ a modified version of cosine similarity [7] as shown in Algorithm 1. Similarity counters of two tasks are incremented when the two tasks output the same key during a certain time window. If some specific keys are sent many times by almost all the tasks (e.g., “the” and “and” for word counting), the similarities between these tasks become uniformly high and the accuracy of the proxy computation becomes worse. To avoid this, the increment of similarity is done only once for the same key. After updating the similarity, ApproxSW performs a proxy computation for each delayed task based on the output of the normal task which has a high similarity to the delayed task. The similar data are selected from the outputs of normal tasks based on a probability determined by the degree of their similarity. That is, ApproxSW copies outputs of a similar task with the probability P in order to perform a proxy computation for the delayed task. Equation (1) shows how to calculate P

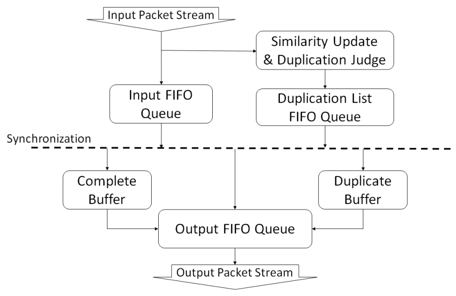

Fig. 1. Packet processing flow in ApproxSW

based on the similarity S.

$$P = \frac{{S_{ab}}^4}{\sum_{i=0}^{n-1} {S_{ai}}^4}, \quad (1)$$

where n is the number of tasks, a is the delayed task, b is the other task, and S_{ab} is their similarity. P is a value obtained by normalizing the similarity. We empirically use the fourth power of similarity as the evaluation function.

2) Completion Notification: If ApproxSW has not received completion notifications from all the tasks yet, it sends the completion notifications instead of delayed tasks when a predetermined time has passed since it received the first completion notification. After sending completion notifications by proxy, ApproxSW communicates with delayed worker nodes to terminate their delayed Map tasks.

IV. Implementation

We employ NetFPGA-SUME board as an FPGA-based switch that has four 10GbE interfaces. NetFPGA-SUME Reference Switch Lite design [8] is used as a baseline 10GbE switch, and we implement our ApproxSW to handle stragglers by modifying the baseline switch.

A packet stream goes through the switch in 256-bit per cycle based on the implementation of AXI (Advanced eXtensible Interface) on the Reference Switch Lite. When the network switch receives a packet, the packet is classified into a Map output, a completion notification, or the other application packet. In ApproxSW, UDP packets destined to specific port numbers are identified as Map outputs or completion notifications [1]. A hash table manages whether each key has been sent by each Map task. That is, a hashed value of a key is used as an index of the table where the flag (sent or not) of the key is stored. Please note that when ApproxSW receives the other packets, such as ARP (Address Resolution Protocol) requests and replies, they are simply passed to the original L2 processing module in the network switch as regular packets.

Figure 1 illustrates a packet processing flow of ApproxSW. A received packet is first buffered in the FIFO queue. If it is an output from a Map task, corresponding task similarities are updated based on its key field. Also, delayed tasks are detected based on their Map output amounts as illustrated in Section III-A. The division for the average calculation is implemented with a right shift operation based on an assumption that the number of Map tasks is a power of two. The received packet is duplicated for a delayed task with a probability P.

[1]Although we assumed UDP as a transport layer protocol for simplicity, our concept can be extended to TCP by combining with commercially or freely available FPGA-based 10Gbps TCP cores, such as [9].

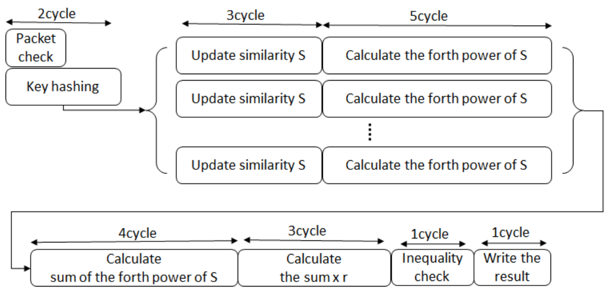

Fig. 2. Pipeline for similarity updating and duplication judgement

TABLE I
Three Datasets

	File configuration	Total number of words	Words per task
Dataset1	1 file × 16	25,803	1,612
Dataset2	8 files × 2	24,859	1,554
Dataset3	16 files × 1	24,850	1,553

Figure 2 illustrates a pipelined processing for updating the similarities and detecting the delayed tasks for which the proxy computation is required. The pipeline processing takes 19 cycles for each packet. After the duplication judgement in the pipeline, the packet is removed from the FIFO queue and passed to the original L2 switch module. If the packet is a Map output which will be duplicated for one or more delayed tasks, it is duplicated and stored in a duplication buffer. A proxy Map output is generated by coping from the duplication buffer and then it is modified so that its id field is changed to one of the delayed tasks and passed to the L2 switch module. These steps are performed for each of the delayed tasks to which the Map output is duplicated. If the packet is a completion notification, it is also duplicated and stored in a completion buffer. When a predetermined time has passed since the first completion notification was received, proxy completion notifications are generated in the same way as the proxy Map outputs.

V. Evaluations

A. Accuracy of Proxy Computation

We evaluate ApproxSW in terms of the accuracy of proxy computation on the final result. Word count is employed as a target application in the experiments. In the word count workload, a Map output is a key-value pair where the key is a word and the value is always 1 (e.g., “apple, 1”). We used three input datasets listed in Table I and evaluate the three proxy computation methods: drop, random, and similarity. Each dataset consists of 16 files. Dataset1 has 16 identical files. Dataset2 has eight pairs of two identical files. Dataset3 has 16 different files. These files included in the datasets are obtained by randomly-selected Wikipedia articles so that their file sizes are approximately 10,000 Bytes. Each Map task is in charge of a single file. The number of Map tasks is set to 16 and the number of stragglers is set to two. The processing times of delayed tasks are set to 20 times those of the normal tasks.

Figure 3 shows the results of proxy computations in ApproxSW with two proxy methods: random and similarity. In these figures, “error” and “correct” represent the wrong and correct results generated by the proxy computations, respectively. Both the random and similarity methods work

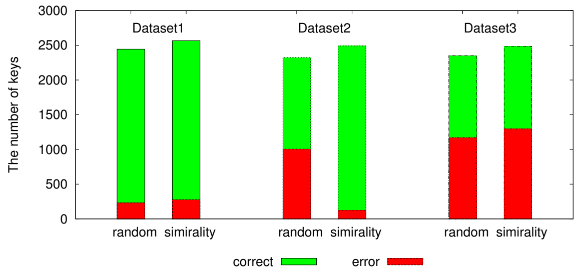

Fig. 3. Accuracy of proxy computation results only

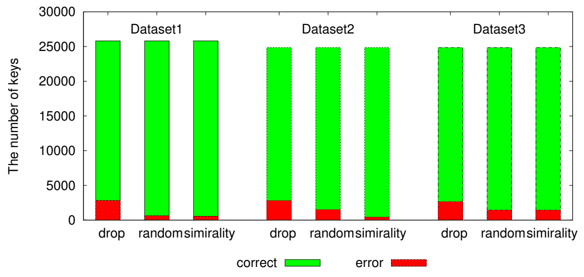

Fig. 4. Accuracy of total results including proxy computation results

well for Dataset1, in which all the tasks have the largest similarity. Furthermore, the similarity method achieves a high accuracy for Dataset2, in which at least a single pair of tasks has a high similarity. Such situations would be more likely in real workloads where at least a single pair of tasks has a high similarity. For example, tasks that process one of the chunks sourced from the same file may have a high similarity to each other. On the other hand, when all the tasks have uniform similarity at all, the random method is better because of simplicity. Such situations occur when the input data have no locality.

Figure 4 shows the results with the three proxy response methods compared to the ideal results with no stragglers. Please note that these figures include entire results of Map tasks while Figure 3 accounts only for the results generated by proxy computation. In these figures, “correct” represents the results which are matched to the correct results. “error” represents the results which are not matched to the correct results. As shown, negative impacts on the proxy computation in terms of accuracy is small because the proportion of stragglers is originally small.

B. FPGA Utilization

The resource utilization of ApproxSW that implements the similarity method is 18% of LUTs, 32% of BRAMs, and 37% of DSPs on the target FPGA device (Xilinx Virtex-7 XC7VX690T). The bit width of a hashed value is 16-bit and thus the number of hash table entries is 32,768. A similarity between two tasks is represented as a 16-bit value. The number of managed tasks is 16. The operating frequency is 200MHz. As shown, the resource utilization of ApproxSW is still low and can be extended to manage more tasks with more sophisticated methods.

C. Throughput

We evaluate the throughput of ApproxSW by using Open Source Network Tester (OSNT) on NetFPGA-10G board [8]. The number of incoming packets is counted in ApproxSW and the counter value is read by a host application in every 500msec to measure the real throughput. We measured the throughputs of the original Reference Switch Lite and ApproxSW for ten times and calculate the average values. When the packet size is set to 512-bit, the original Reference Switch Lite processes packets at 17.86Gbps and ApproxSW achieves 17.87Gbps. When the packet size is 1,024-bit, the original Reference Switch Lite achieves 21.44Gbps and ApproxSW achieves 21.40Gbps. As shown, there are no significant differences between them and performance overhead of ApproxSW is negligible.

VI. Conclusions

To eliminate the burden to handle stragglers by the master node in MapReduce applications, in this paper we proposed ApproxSW which is a network switch based straggler detection and proxy computation mechanism, because all the information goes through the switch. Thus, by introducing ApproxSW, the master node no longer has to monitor the progress of each task and detect stragglers. Also, fast workers no longer have to rerun delayed tasks speculatively as in Backup Task. However, the proxy computation for delayed tasks by a network switch is not a trivial job due to its limited resource; thus, we adopted an approximate proxy computation that replicates Map outputs of normal tasks which have a high similarity to the delayed tasks. The proxy computations of delayed tasks are performed in parallel with Map tasks so that it does not increase the total execution time. ApproxSW was implemented on NetFPGA-SUME board that has Xilinx Virtex-7 FPGA and four 10GbE interfaces. It achieved the same performance as the original Reference Switch Lite. As a future work, we are planning to adopt our ApproxSW in large-scale MapReduce applications that handle hundreds of worker nodes.

Acknowledgements This work was supported by JSPS KAKENHI Grant Number JP16H02816 and SECOM Science and Technology Foundation.

References

[1] J. Dean and S. Ghemawat, “MapReduce: Simplied Data Processing on Large Clusters,” in *Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI’04)*, Dec. 2004, pp. 137–149.

[2] K. Ousterhout *et al.*, “The Case for Tiny Tasks in Compute Clusters,” in *Proceedings of the USENIX Workshop on Hot Topics in Operating Systems (HotOS’14)*, May 2013.

[3] M. Zaharia *et al.*, “Improving MapReduce Performance in Heterogeneous Environment,” in *Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI’08)*, Dec. 2008, pp. 29–42.

[4] G. Ananthanarayanan *et al.*, “Reining in the outliers in mapreduce clusters using Mantri,” in *Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI’10)*, Oct. 2010, pp. 1–16.

[5] K. Ousterhout *et al.*, “Sparrow: Distributed, Low Latency Scheduling,” in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP’13)*, Nov. 2013, pp. 69–84.

[6] I. Goiri *et al.*, “ApproxHadoop: Bringing Approximations to MapReduce Frameworks,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’15)*, Mar. 2015, pp. 383–397.

[7] Gerard Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[8] “The NetFPGA Project,” http://netfpga.org/.

[9] D. Sidler *et al.*, “Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware,” in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM’15)*, May 2015, pp. 36–43.