Received September 8, 2019, accepted September 22, 2019, date of publication September 25, 2019, date of current version October 8, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2943632

Towards an Energy Efficient Computing With Coordinated Performance-Aware Scheduling in Large Scale Data Clusters

PRINCE HAMANDAWANA[1], RONNIE MATIVENGA[1], SE JIN KWON[2], (Member, IEEE), AND TAE-SUN CHUNG[1]

[1]Department of Computer Engineering, Ajou University, Suwon 16499, South Korea
[2]Department of Computer Engineering, Kangwon National University, Chuncheon 24341, South Korea

Corresponding author: Tae-Sun Chung (tschung@ajou.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2019R1F1A1058548 and Grant NRF-2017R1D1A3B04031440.

ABSTRACT Many prior works have investigated on how to increase the job processing performance and energy efficient computing in large scale clusters. However, they employ serialized scheduling approaches encompassed with task straggler "hunting" techniques which launches speculative tasks after detecting slow tasks. These slow tasks are detected through node instrumentation which collects system level information whilst tracking the task execution progress. Such approaches are however detrimental towards achieving maximum processing performance and preserving cluster energy as they increase communication overheads. In this paper, we observe that node instrumentation and serialized scheduling in existing works does not only degrade the job processing performance, but also increase cluster energy consumption. To alleviate this, we propose EPPADS, a light-weight scheduler which eradicates the need for instrumentation modules for job scheduling purposes. EPPADS schedules tasks in two stages, the slow-start phase (SSP) and accelerate phase (AccP). The SSP schedules initial tasks in the queue using baseline FIFO scheduling and records the initial execution times of the processing nodes, whilst tagging the effective and straggling nodes. The AccP uses the initial execution times to compute the processing nodes task distribution ratio of remaining tasks and schedules them in parallel using a single scheduling I/O, boosting up the processing performance. To amortize the computing energy costs, EPPADS implements a power management module that coordinates with the scheduling module and leverage on node tagging information, to place nodes in two different power transition pools, i.e., high and low state power pools. A single power transition signal per pool is then broadcasted to lower or raise the energy state in the low-power state pool and high-power state pool. Our evaluation using a Hadoop cluster shows that EPPADS achieves 30% and 22% performance improvement and 15% to 20% energy savings as compared to the FIFO and DynMon schedulers, respectively.

INDEX TERMS Distributed processing, energy efficiency, scheduling.

I. INTRODUCTION

For the past two decades, there has been a continuous trend in big data proliferation, with the amount of stored data doubling every 2 years [1]. To process this huge amount of stored data, a new generational approach is needed which offers high performance parallel data processing with minimal power consumption in large scale data clusters. Prior research [2]–[4] proved that the job completion time is a key performance indicator that needs serious attention to achieve high job processing throughput in big data processing clusters. To achieve this, jobs are divided into smaller manageable tasks which are then distributed to several nodes for processing. However, the job processing time is frequently slowed down by one or more tasks which may be executing on a slower node. These slower nodes are referred to as *straggler nodes* while the slower tasks executing on the straggler nodes are referred to as *stragglers*. The straggler nodes often occur as a result of

The associate editor coordinating the review of this manuscript and approving it for publication was Kuan Chee.

congestion/overloading or system crash at processing nodes. In order to achieve high job processing performance, it is vital to amortize the performance degradation caused by the stragglers.

Because of its popularity and robustness, MapReduce [5] has been increasingly adopted in the cloud community to provide a high performance, parallel framework in large scale job execution clusters. One significant benefit of MapReduce is that it hides the problems of managing stragglers from the developers, through the automatic re-assignment of redundant tasks of stragglers onto free slots of effective job execution nodes. This automated process is known as speculative execution (SE) and it often results in decreased job response time. Prior research [5], proved that SE decrease the job processing time by almost 44%.

However, in the MapReduce system, the default scheduler divides the tasks evenly across all available processing nodes called task trackers (TT), for parallel task processing in a FIFO way. Such task scheduling presume that all TTs have a homogeneous computing capacity. In cases where heterogeneous cluster set-ups are provisioned, the default MapReduce schedulers will not produce optimal performance because some TTs will have higher computing capacity leading to better performance as compared to others. Previous research work, LATE [6], demonstrated that when the default FIFO scheduler is subjected to heterogeneous clusters set-ups, multiple unnecessary speculative invocations of tasks are initiated which increase the job response time. To avoid such unwarranted speculative execution invocations, the LATE scheduler thoroughly chooses only tasks that lead in reduced job execution time, for speculative execution. But previous research [7], proved that LATE only improve the execution time of the first job in the queue.

Several self adaptive schedulers [7]–[10], which dynamically adapts to heterogeneous clusters have been presented. These proposed schedulers operates by frequently gathering up to date system level information and use it to dynamically distribute job tasks to the heterogeneous task trackers, based on their computing capabilities. This is accomplished through implementation of additional software modules in the Job tracker node (also known as master node) which instruments all task tracker nodes computing capabilities. By so doing, it therefore impose the use of system level information that is always up to date and consequently, only speculative tasks that result in lower job response times are invoked. However, such instrumentation add more overheads as the number of task trackers to be monitored exceeds a certain threshold.

In a nutshell, existing MapReduce schedulers suffer from several limitations that hinders the achievement of maximum job processing performance in large scale data clusters; i) sequential scheduling of tasks, where the succeeding task in a job/task queue can only be scheduled only if the preceding task has completed its execution. This results in longer job execution time as all tasks has to wait until the scheduled task finishes. ii) Current implementations adopt the straggler ''hunting'' technique. In this approach, speculative execution is only initiated after the detection of a straggler. This means that speculative execution is invoked in the middle of a task execution which introduces some lag time between the beginning of task execution and the launch of its corresponding speculative task. Consequently, the job response time is often degraded as the speculative task is often invoked late. iii) Increased instrumentation overheads which are caused by the frequent collection of system level information of the task tracker nodes to forecast the task completion time in order to make scheduling decisions. However, such communication overheads result in significant performance degradation and also increased cluster power consumption.

These communication and power consumption costs increase dramatically as the cluster scales out. Nowadays, the power consumption in large scale clusters, is an important factor which needs to be taken into account when designing high performance and energy efficient clusters. Therefore, effective energy saving approaches needs to be designed to solve the power consumption problems faced in large scale data processing clusters [11]–[16]. Previous works [17], tried to resolve the performance penalty caused by node instrumentation, but it did not resolve the power consumption problem faced in large scale clusters. To address the discussed limitations of the existing approaches, we propose an energy efficient **E**nhanced **P**hase-based **P**erformance **A**ware **D**ynamic **S**cheduler (EPPADS). Rather than using heavy instrumentation modules to make scheduling decisions, EPPADS uses an opportunistic rolling window implementation where inbound tasks are divided into smaller partitions, with each partition representing a scheduling window instance.

To predict the compute capability of each task tracker and also pre-determine straggling nodes during the first phase called the slow-start phase (SSP), EPPADS launches the first tasks in the current scheduling window instance in the usual baseline FIFO scheduling approach. At this stage it predicts the effectiveness of each task tracker and flag them as either effective or straggler. This exterminate the purpose of node instrumentation used in prior works. During the second phase of EPPADS, called the accelerate phase (AccP), EPPADS addresses the problem caused by serialized task scheduling through scheduling the outstanding tasks in the current scheduling window at one go, using a single scheduling I/O. As the straggling nodes are determined during the SSP, all tasks scheduled to task trackers flagged as stragglers, are invoked simultaneously with their respective redundant speculative tasks. This helps to eradicate the problem caused by delayed speculative execution in previous works. The pre-scheduling of speculative tasks at the same time with their pre-determined straggling tasks helps to amortize the heavy instrumentation overheads by removing the need for the additional instrumentation modules that frequently monitors the task trackers' system level information.

Finally, our approach achieves effective energy computing through the use of a straggler tagging technique that transit all slow nodes into a low power state, whilst transiting all effective processing nodes into high power state. This technique is called *power transition tagging.* This is motivated by the idea that, for all the tasks which are scheduled to straggling nodes, we fuse the speculative task launching together with the original task, then it is more likely that the speculative tasks have an increased chance to finish first. Hence, we prioritize power to non-straggling nodes. To maximize the power conservation in large clusters, we build our power conservation module which works on top of the existing OS built power saving configurations such as system idle time, sleep modes etc.

We summarize the contributions of our work as follows;

- **Problem Identification:** We expose the current limitations of existing implementations which inhibit the attainment of high job processing performance in large scale data processing clusters.
- **Lightweight dual-phased opportunistic scheduler:** We propose EPPADS, a lightweight dual-phase opportunistic job scheduler, which drastically diminish the performance penalty caused by processing nodes instrumentation and sequential scheduling of tasks.
- **Highly accelerated job processing performance:** We design a rolling scheduling window strategy that enables the fusion of job scheduling and task pre-speculation based on the computational effectiveness of each processing node. The dual-phased nature of EPPADS enables an accelerated scheduling performance on the second phase (AccP), where a single I/O is used to schedule all outstanding tasks at once.
- **Effective and simple cluster power management:** EPPADS provision an efficient energy computational model by providing a simple yet effective power management scheme that tightly integrate the job scheduling together with the power management functions. We achieve this through the implementation of a pool based software defined dynamic frequency scaling that transit the power usage of nodes between high and low power states.
- **EPPADS prototyping:** We implemented a real prototype of EPPADS in Hadoop, achieving 30% and 22% performance improvement as compared to baseline FIFO scheduler and existing dynamic self adaptive schedulers, respectively. In terms of energy efficiency, EPPADS achieves between 15% to 20% as compared to FIFO and the dynamic scheduling approaches.

The rest of the paper is organized as follows: Section II outlines a detailed description of the background, limitations and motivation. Section III discusses our approach in addressing the limitations in existing works. We discuss our proposed technique towards achieving effective energy efficient computing in Section IV. Section V shows the details of our implementation in Hadoop. In Section VI, we present our evaluation results followed by related work in Section VII. Finally we share our concluding remarks in Section VIII.

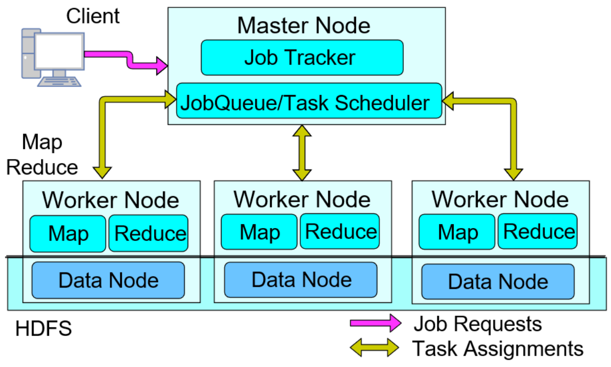


FIGURE 1. **MapReduce framework.**

II. BACKGROUND AND MOTIVATION

A. MAPREDUCE SCHEDULING MECHANISM

In-order to fully understand the drawbacks that are associated with current MapReduce job processing techniques, we will first look at how jobs are processed. Figure 1 depicts the general architecture of a MapReduce job processing cluster. Firstly, client nodes issue job requests to the job tracker node, and the job tracker node divides the incoming job requests into smaller manageable tasks which are equally distributed to the task trackers for parallel processing. The job tracker has the responsibility to track the progress of the scheduled tasks in each task tracker node (also known as worker nodes). All issued jobs pass through a map and a reduce phase and again the job tracker is responsible for coordinating between the map and reduce phases of every MapReduce job. During the Map stage, all the incoming data is partitioned and the output of the map stage is some intermediate key-value pairs. These intermediate key-value pairs are then sorted and combined in a way that same key-value pairs will be processed by the same reducer on the reduce stage. The reducers will then finally process the sorted key-value pairs and outputs the final key-value pairs.

The way in which these map reduce jobs are scheduled is vital to attain maximum job processing performance. In the MapReduce parallel framework, jobs are split and distributed equally for processing across task tracker nodes using the default FIFO scheduler. During the job execution time, the job tracker continuously monitors the progress of each job task. If there is any task that is below the average progress rate of all tasks (straggler), a redundant/speculative task is launched on an effective task tracker with an available processing slot. The result of whichever task finishes first is sent as output and the corresponding lagging task is immediately terminated. This speculative execution of tasks helps to significantly boost the job processing performance [5]. However, this performance gain is significantly degraded when the default FIFO scheduler runs on a cluster provisioned with heterogeneous nodes [6]. The reason is that the default FIFO scheduler was designed to work effectively in homogeneous clusters where all nodes have uniform compute capabilities. To avert this problem, it is important to design efficient

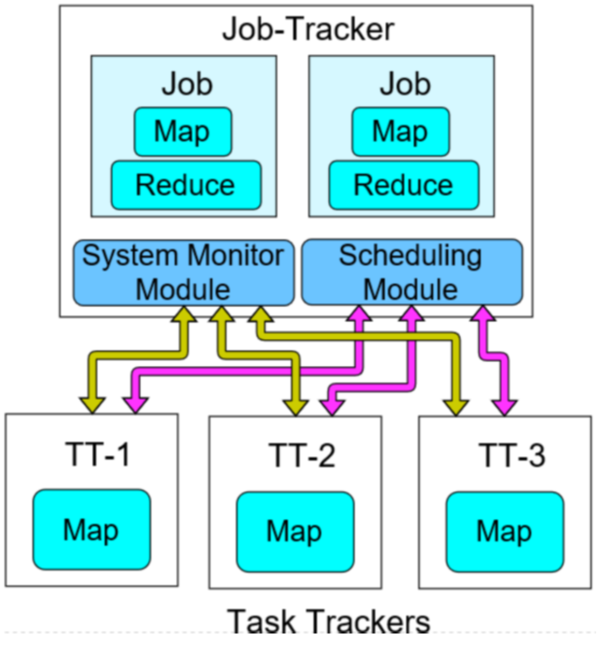


FIGURE 2. Node instrumentation.

performance-aware dynamic schedulers that helps to improve job processing performance in clusters equipped with heterogeneous processing nodes.

B. PERFORMANCE-AWARE DYNAMIC SCHEDULING

A lot of prior research works have already put a lot of effort in trying to improve the job processing performance in clusters provisioned with heterogeneous nodes [6]–[10], [17]–[20]. This heterogeneity, coupled with fluctuating workload conditions, leads to degraded job processing performance and this can become worse in large scale data processing clusters which handles exabytes of data per day. Consequently, there is a need to provision an effective speculative execution strategy that can consistently respond to the fluctuations in task tracker node system level loading [7], [9], [21]. Based on the system level information of task trackers, the speculative execution of tasks can be invoked judiciously only if their invocation can result in an improved job runtime.

The widely adopted node instrumentation technique which was implemented in prior works [2], [3], [7]–[9], [18], [21], [22] is depicted in Figure 2. In this approach, extra instrumentation modules are added which are responsible to track and record the system level information of all task tracker nodes. Scheduling decisions are then invoked based on the collected task tracker system level information. By so doing, this avoids blindly scheduling tasks to processing nodes which are already overloaded and hence the job processing time is significantly decreased as the jobs are scheduled in a load-aware fashion. However, the current scheduling schemes suffer from the following limitations that inhibits the achievement of maximum job processing performance;

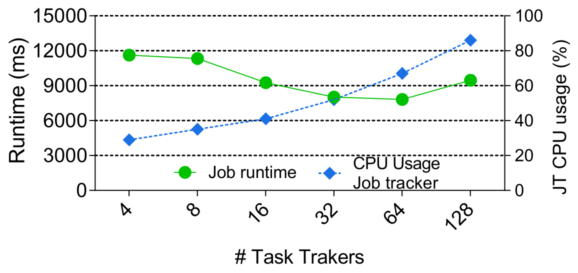


FIGURE 3. An analysis of the default FIFO scheduler. JT in Y-axis means job tracker.

1) INSTRUMENTATION OVERHEADS

Node instrumentation helps to improve job processing performance through enforcing precise speculative task execution by using up to date system level information. However, as the number of nodes scales to hundreds of nodes the instrumentation overhead can cause serious performance degradation. This performance penalty can emanate from the communication and network overhead that often leads to degraded cluster performance. To analyze the effect on performance from instrumentation/monitoring overheads in large scale data processing clusters, we carried out an experiment using the setup described in Section V - Table 1.

We used the Scan traces from the HiBench suite as workloads. We ran this experiment by starting with a 4 node cluster and doubling the cluster nodes at each experimental run, whilst capturing the effect of increased node instrumentation on the cluster. Figure 3 shows the results of our experiment. Our observation shows that an initial increase in the number of cluster nodes, will have a corresponding increase in the job processing performance. Nevertheless, this performance increase reaches a ceiling point (in our case 64 task tracker nodes) and a continuous increase in the number nodes starts to degrade the job processing performance. We also observed that, at above 64 nodes the CPU usage on the job tracker node start to shoot up to above 90% creating a central bottleneck in cluster job processing performance. This observation demonstrates that the performance penalty which is caused by the instrumentation/monitoring overheads is non-negligible in large scale data processing clusters.

2) SEQUENTIAL SCHEDULING OF TASKS

The existing job scheduling techniques, adopts the traditional sequential way of scheduling tasks. This means that all the consecutive tasks in the job queue can be scheduled only when any of the currently scheduled tasks has finished execution. This presents a major constraint towards achieving maximum job execution performance. Mostly, it is because the serialization of task scheduling results in increased scheduling time considering the large number of jobs that are scheduled in large scale data processing clusters. Also, this can increase the task transfer time if the tasks are scheduled in neighboring nodes in which the data is not stored in. Furthermore, the average waiting time of successive tasks in the task queue consequently increases which add more performance degradation.

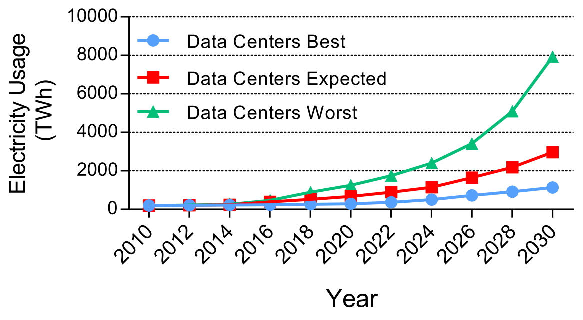


FIGURE 4. Global data center electricity usage demands in Tera-Watts per hour (TWh) from 2010-2030 [23].

3) DELAYED SPECULATIVE TASK EXECUTION

In order to detect straggling tasks during job execution, there should be a mechanism that tracks down the progress of all the tasks running in all task tracker nodes. Existing approaches adopts a straggler "hunting" technique that tracks all tasks that are running below the average execution time. This entails that a straggling task can only be detected in the middle of execution time and only at that time that is when a redundant task can be speculatively invoked on another slot of a faster task tracker node. However, this creates a lag time before the speculative task is scheduled. Consequently, invoking a speculative task at this time maybe too late to limit the slowdown effect on job performance caused by straggling tasks. A more effective approach is needed to prevent the effects on performance caused by this delayed scheduling of speculative tasks.

4) ENERGY EFFICIENT DATA PROCESSING

The deployment of large distributed data clusters in public and private clouds has lead to the escalating operating power costs ($/Watt) [11]. This is mainly due to the amount of data movement inbound, outbound and within the cloud platforms. Also, the highly compute intensive nature related to workloads processed in these huge parallel processing clusters increase the computational energy costs [11], [23], [24]. This Big Data phenomenon raises new challenges that associate with how data is scheduled and processed in large scale data clusters. Works such as [25]–[27] outlines the need to effectively cut the data processing costs whilst improving the way the data is communicated, stored and processed within and across clusters.

Figure 4 shows that the global electricity energy demands across data centers will grow significantly towards 2030 [23]. This trend shows that if data centers adopt effective energy computing approaches, they will be able to save about 61% in energy costs (depicted by Data center best trend Figure 4). On the contrary, data center energy usage would sky-rocket by more than 160% if effective energy computing is not applied (depicted by Data center worst trend in Figure 4). In this regard, energy efficient data processing has become a priority in large scale distributed data clusters.

Many prior works [24], [28]–[36] tried to cut the energy savings by utilizing idle, inactive (sleep) and scaling the power for transition servers from high to low voltage. However in large scale data processing clusters, it is difficult to get sufficient idle and inactive windows to guarantee a significant energy savings. Worse off, it is difficult to get this idleness because of the heavy node instrumentation which guarantees continuous communication (heartbeat, task-tracking, resource weighting, etc) between job trackers and task tracker nodes. This frequent communication also leads to increased power consumption and needs to be optimized. From this perspective there is need to come up with a way of transiting power states in such a way that we can guarantee significant energy savings.

To amortize the above mentioned drawbacks we propose EPPADS, a lightweight data processing scheduler, which achieves high performance job execution without the need for heavy node instrumentation, schedules multiple tasks in the same scheduling window instance in a parallelized manner and fuse job scheduling and task pre-speculation at the same time. EPPADS aims to achieve effective energy computing through the use of pool based node power transitions. Section-III and section-IV explains in detail our proposed design and the efficient energy computing approach in EPPADS.

III. EPPADS DESIGN

Due to the existing limitations in existing works, we aim to achieve the following goals in EPPADS;

- *Elimination of monitoring overheads:* Frequent communication due to monitoring of system level information of all task tracker nodes results in communication overheads. This overhead cost is significant as the cluster scales out. The objective of our proposed design is to reduce significantly this monitoring overhead.
- *Amortize the slow down caused by sequential scheduling:* Scheduling tasks in a serialized manner limits the job execution performance. Consequently, the average waiting time of jobs in the queue is increased due to the sequential scheduling of tasks. We aim to amortize this drawback in our proposed approach.
- *Reduce the time delay in launching speculative tasks:* In order to initiate a speculative task, existing works often wait until a job task starts to straggle. This degrades the job processing performance as it is often late to limit the performance degradation caused by the straggling tasks. We also aim to limit this overhead caused by the delay in invoking speculative tasks.
- *Effective energy efficient data processing:* As the amount of data to be processed and the number of nodes increase, it is important to take care of the power consumption factor in large scale data processing clusters. Our Approach aims to amortize the power consumption costs involved with large scale data processing.

A. EPPADS DESIGN OVERVIEW

In this section we discuss a brief overview of the EPPADS architecture. EPPADS is built on top of the existing Hadoop's

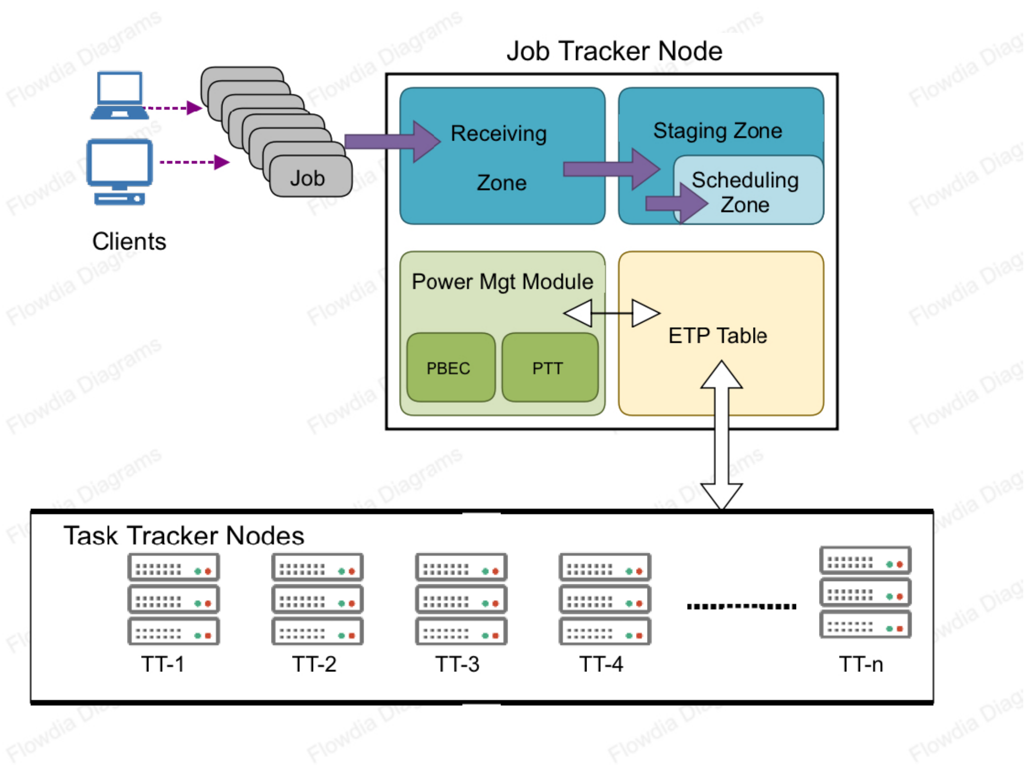


FIGURE 5. An overview of EPPADS architecture.

MapReduce architecture and is composed of clients, a job tracker node and several task tracker nodes which executes the task processing. The architecture of EPPADS is depicted in Figure 5. Firstly, EPPADS divides the processing of inbound jobs which are issued by client applications into three distinct buffer regions, *receiving zone, staging zone and scheduling zone*. The function of the receiving zone is to chunk the incoming jobs submitted by clients into smaller manageable tasks that can be processed quickly by the task tracker nodes. The receiving zone also forwards the smaller tasks into the second buffer area called the staging zone. This staging zone is a pinned/fixed buffer region which is responsible for preparing the tasks for scheduling. The staging zone is pinned so that tasks which are staged for scheduling cannot be swapped during periods of memory stress. Inside the staging zone there is a small sliding window buffer area called the scheduling zone and is where the actual task scheduling occurs.

Recap that when the job tracker node receives job requests from clients, these jobs are chunked into smaller equal sized tasks. By default, these smaller tasks are then shared equally among existing task tracker nodes for task execution. This task distribution is done in a sequential manner following the FIFO scheduling approach. On the Contrary, in order for EPPADS to achieve high performance job processing, it pipelines the incoming job requests in the following flow; i). In the receiving zone, the client job requests are first chunked into smaller equal tasks just as in the baseline. EPPADS then forwards these chunks into the staging zone which stages the tasks in preparation for the actual scheduling. These staged tasks are then absorbed onto the smaller scheduling zone buffer area as the rolling scheduling window slides across the staging zone buffer area as depicted by Figure 6. The details of this rolling scheduling window are discussed in detail in Section III-B.

EPPADS utilizes a dual pipeline based scheduling strategy to amortize the performance penalty caused by tracking system level information of task trackers. In the first pipeline stage called the Slow Start Phase (SSP), the first tasks within the current scheduling window instance, are scheduled to the available task trackers in the usual FIFO way. An *Execution Time Prediction Table* (ETP) then records and store the initial execution time for the first FIFO scheduled tasks. The ETP is discussed in more detail in Section III-C. At the second pipeline stage, called the Accelerate Phase (AccP), EPPADS then utilizes the computed execution time ratios of the first tasks scheduled in the SSP and calculates the task distribution ratio to be used in distributing the rest of the remaining tasks in the scheduling window. The task distribution ratio is then used to schedule all the remaining tasks in the scheduling window using a single I/O operation and the scheduling window rolls by an offset which is equal to the size of the scheduling zone.

Most importantly, EPPADS uses the SSP stage to detect the slow task tracker nodes and these nodes are tagged with a straggler flag. The details of straggler node detection are explained in detail in Section III-C. This is done so that EPPADS can simultaneously invoke a redundant speculative task to the closest non-straggler node when a task is scheduled to a task tracker flagged as a straggler. This helps to reduce the performance slowdown caused by the problem of delayed speculative invocation in current approaches.

Lastly, in order to improve power consumption, EPPADS is provisioned with a power management module that controls the power consumption of processing nodes. The power management module consist of two sub-modules i) Power Transition Tagging (PTT) module and the Pool-Based Energy Conversation (PBEC) module. The main goal of PTT module is to flag nodes with power transition tags (PTT = 0 for low power and PTT = 1 for high power.) The PBEC module then groups the nodes into two power transition pools based on the information provided by the PTT module. Power transition decisions are then executed on a per pool basis rather than individual nodes using a single power transition broadcast signal rather than per individual node. We discuss in detail our approach to achieve effective energy computing in Section IV. Next, we discuss in detail the scheduling mechanism in EPPADS.

B. TASK STAGING AND SCHEDULING MECHANISM

To schedule job tasks in EPPADS, the tasks must first be forwarded onto the pinned staging zone area. The scheduling zone, which is made up of a small rolling scheduling window takes the responsibility of initializing the scheduling of all the tasks that are forwarded into the staging zone. To understand the scheduling mechanism inside EPPADS, we first describe the details of the rolling scheduling window.

Rolling Scheduling window: To predict the task response times without the need of additional instrumentation modules, we design our scheduling decision window based on the number of tasks to schedule rather than using instrumented periodic time windows. Instead of making scheduling decisions based on the entire number of job tasks in the job queue, we split the job tasks into several smaller decision windows called rolling scheduling windows inside the staging zone area, as depicted by Figure 6. The function of the rolling window is to initiate the actual scheduling of the tasks that are staged in the staging zone. This scheduling happens at offsets equal to the size of the configured scheduling window at a time, until the scheduling window rolls across all the tasks in the entire staging zone. After the rolling scheduling window reaches the offset position at the end of the staging zone, it then resets back to the starting offset position inside the staging zone.

It is also important to determine the optimal size of the staging zone and the scheduling zone (scheduling window) buffers as they also influence the job scheduling performance. A small staging zone size increases the task transfer frequency between the receiving zone and the staging zone. This can impact the overall job processing performance. On the other hand a smaller scheduling zone, will result in less parallelism as it reduces the number of tasks to be scheduled in

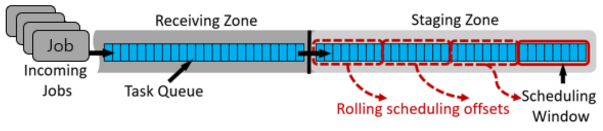


FIGURE 6. Design of EPPADS rolling scheduling window.

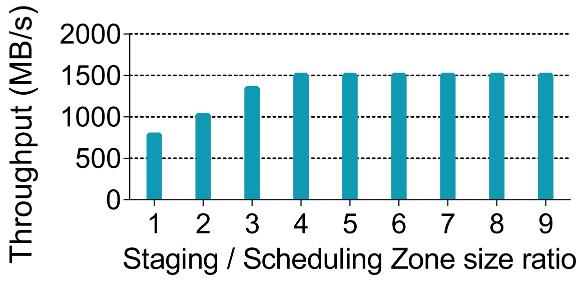


FIGURE 7. Buffer size analysis.

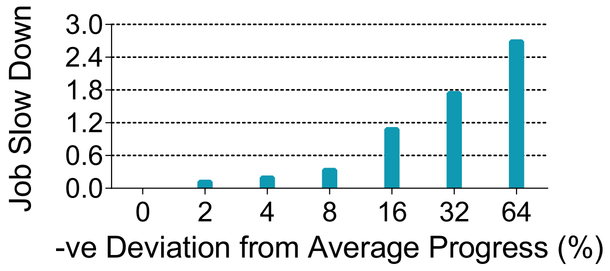


FIGURE 8. Job slow down effect by stragglers.

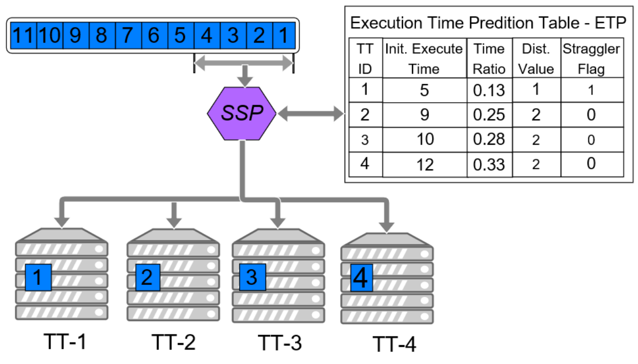
parallel. Therefore, an optimal size ratio between the staging zone and the scheduling zone must be provisioned to attain optimal job processing performance.

In our case, we configured the staging zone to the scheduling zone buffer ratio to 4:1. This ratio determines the size of the scheduling window in our algorithm. The idea is to amortize the memory copy operations when tasks are copied from the receiving zone buffer to the staging zone buffer. The other reason is also to reduce the waiting time to forward tasks from the staging zone buffer onto the scheduling zone buffer. Figure 7 shows that setting the staging zone size greater than 4 times that of the scheduling zone did not have any performance benefit. During the actual scheduling, each scheduling instance is made up of a two-staged pipelined scheduling approach (SSP & AccP). Before we detail the two phased scheduling approach, we will explain the execution time prediction table which is used in both the SSP and the AccP pipeline stages.

C. EXECUTION TIME PREDICTION TABLE (ETP)

In the ETP, each task tracker ID (TT-ID) has a corresponding initial execution time, time ratio, distribution ratio and a straggler flag. We define these terms as follows;

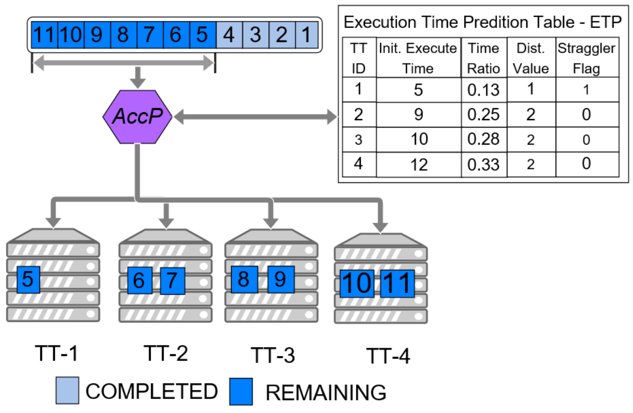
- **Initial execution Time:** This is the total time to execute the initial task sent to that task tracker during the SSP stage.
- **Time ratio:** This is defined as the ratio of time taken to execute the initial task to the total execution time of all



Execution Time Predition Table - ETP

TT ID	Init. Execute Time	Time Ratio	Dist. Value	Straggler Flag
1	5	0.13	1	1
2	9	0.25	2	0
3	10	0.28	2	0
4	12	0.33	2	0

FIGURE 9. Slow-Start Phase(SSP).



Execution Time Predition Table - ETP

TT ID	Init. Execute Time	Time Ratio	Dist. Value	Straggler Flag
1	5	0.13	1	1
2	9	0.25	2	0
3	10	0.28	2	0
4	12	0.33	2	0

FIGURE 10. Accelerate Phase (AccP).

the initially scheduled tasks in the SSP phase. Example of the first scheduled tasks is task-1 to task-4 as depicted in Figure 9.

- **Dist. Value:** This value is used in the second pipeline stage of EPPADS and it defines the number of tasks to be distributed to a task tracker node during the AccP stage. It is computed by multiplying the number of remaining tasks (e.g. task-5 to task-11 in Figure 10) in the current scheduling window by the time ratio of the task tracker node. The Dist. Value is rounded to the nearest integer.
- **Straggler Flag:** This Flag is used to pre-determine the possibility of the node being a straggler on not. The straggler flag is set to 0 when the processing node is not straggling or 1 when the node is straggling. We set all nodes with the initial execution time which is $\leqslant \theta$ from the average execution time of all tasks during the SSP stage as stragglers. The value θ is a threshold value of the tolerable negative deviation from the average execution time during the SSP stage. This value of θ is provisioned as a tunable parameter which is workload depended. In our case we set it to 8% negative deviation from average execution time. To determine the 8% value, we measured the job slow down caused by the increase in negative deviation from the average task on the straggler node. We increased the load in the straggler node at each experimental run as we measured the job slow down. Figure 8 shows that a negative deviation of less than 8% from the average progress will not cause significant slowdown. With the Dist. Value and straggler flag result we can schedule all the remaining tasks in the scheduling window in parallel rather than in a serialized manner.

In the next subsection we discuss in detail our proposed dual phased scheduling approach which is comprised of the slow start phase and the accelerate phase.

D. DUAL PHASE PIPELINED SCHEDULING

1) KEY IDEA

For high performance job execution, EPPADS consist of two pipelined scheduling phases (SSP & AccP). The key idea of the dual-phase pipelined scheduling is to first forecast the compute capabilities of the task tracker nodes in the first pipeline stage (SSP) then distribute the rest of the remaining tasks in parallel according to the predicted compute capability of each task tracker in the second pipeline stage. By so doing, straggling nodes can be detected in advance, thereby eliminating the need for tracking task progress during the task execution period. The important idea of this two stage pipelined scheduling in EPPADS is that task scheduling can be fused together with task pre-speculation hence, increasing the job processing performance. Next, we explain the the two stages of EPPADS scheduling in detail.

2) STAGE-1: SLOW-START PHASE

The scheduling of tasks inside the current scheduling window instance starts with the slow start phase. Figure 9 depicts an example of scheduling in the slow start phase. In this phase, the first tasks, e.g. tasks 1, 2, 3 and 4, at the front of the task queue in the scheduling window are scheduled to the available task tracker nodes sequentially just like in the default FIFO scheduler. The execution time of these first scheduled tasks are recorded in the ETP table. At this stage EPPADS also computes for each node, the time ratio, distribution value and sets the straggler flag to either 0 or 1. The reason to do this is to get an insight of the execution time of a single task on each task tracker node. Since the scheduling window is small we assume; (i) It is fast enough to execute all tasks in the current scheduling window before any significant change in task tracker node system load. (ii) Since it is fast enough to compute all tasks in a scheduling window instance, we assume that the time to compute each remaining task on the task-tracker is equal to the recorded time taken to compute the first issued task. By so doing, our key idea is to eliminate the overhead caused by task tracker system level instrumentation. Although the slow start phase of the EPPADS scheduling limits performance, the worst case scenario is equal to the performance of the default FIFO scheduler. This is because at this stage the scheduling is done in a sequential manner similar to the default FIFO scheduler.

3) STAGE-2: ACCELERATION PHASE

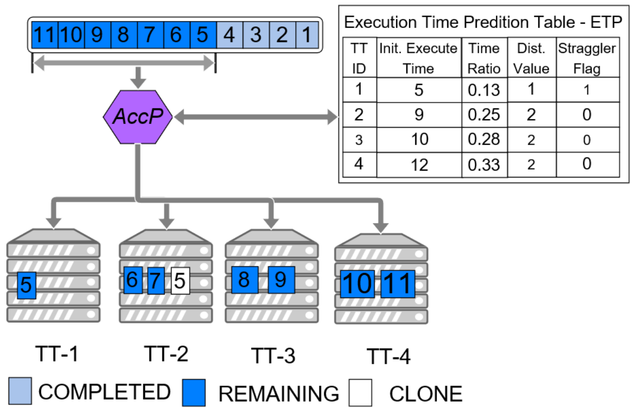
With the knowledge of the time to compute a single task at each task tracker node (stored in the ETP table), EPPADS

Algorithm 1 EPPADS Scheduler.

```
Input: $J_i$, ith Requested job to cluster
$\Omega$, Rolling scheduling size;
$\alpha$, Initial sequentially scheduled tasks
T, Total tasks for scheduling window
$t_i$, ith task in scheduling window
M, Number of task-trackers in the cluster
$W_i$, ith window of $J_i$
$P_{rate}$, Average Progress Rate
$\theta_i$, Tolerable negative deviation of $task_i$
$p_{rate}$, Task Progress Rate
$Ct_i$, Clone of Task $t_i$
$TS_{Ci}$, , Completion time of Clone $Ct_i$
$TS_{ti}$, Completion time of task $Ct_i$
Output: RESULT, key-value pair of task $t_i$;
for $W_i \neq 0$ do
    for $0 < t_i < T$ /* begin Slow-Start Phase (SSP) */
    do
        Insert $t_i$ into scheduling window $W_i$
        Query progress for $\alpha$ tasks
        for all $t_i$ with $p_{rate} < (P_{rate} - \theta_i)$ do
            Mark task-tracker as straggler
        end
        for Remaining tasks in $W_i$ /* Initialize
        Accelerate Phase */
        do
            Preschedule all $t_i$ using the time from $\alpha$
            tasks
            Clone tasks scheduled on stragglers /* using
            alg_2*/
        end
        for All pre-scheduled $Ct_i$ do
            if slot_exists=true then
                Launch $Ct_i$ together with corresponding
                $t_i$
                if $TS_{Ci} < TS_{Ti}$ then
                    RESULT = result of $Ct_i$
                end
                else
                    RESULT = result of $t_i$
                end
            end
            else
                Execute $Ct_i$ on next available slot
            end
        end
    end
    Slide by offset $\Omega$ to next scheduling window
    return RESULT
end
```

then schedules the remaining tasks in parallel using a single I/O. For example, in Figure 10, EPPADS uses the task tracker distribution value, which is computed from the initial time ratio, to schedule the remaining tasks 5, 6, 7, 8, 9, 10, and 11 at once. The key idea here is to achieve the best overall execution time by making sure that scheduled tasks in the AccP phase finish at almost same time on all nodes as they are scheduled based on the nodes execution capabilities.

Therefore, for any scheduling window instance, we assume time to compute scheduled tasks on all task-trackers is almost equal. Consequently, the overall scheduling time is speed up as the remaining tasks are scheduled at once in the accelerate phase, rather than in a serialized way. The main benefit of the acceleration phase is to amortize the drawback caused by sequential scheduling of tasks. Algorithm 1 shows the scheduling flow of EPPADS using the SSP and the AccP dual phase scheduling.



Execution Time Predition Table - ETP

TT ID	Init. Execute Time	Time Ratio	Dist. Value	Straggler Flag
1	5	0.13	1	1
2	9	0.25	2	0
3	10	0.28	2	0
4	12	0.33	2	0

FIGURE 11. Clone invocation in AccP.

E. SPECULATIVE EXECUTION OPTIMIZATION

In order to increase the job execution performance the manner in which straggler tasks/nodes within the cluster are detected is important. Two important questions to come up with a suitable solution arises; (i) At which stage during task execution is a straggling node detected? (ii) At the time of straggler node detection, does the launch of a speculative task avoid the job execution delay or it actually causes more overheads? To answer these two questions, the dual-phase design of EPPADS assist in detecting straggler nodes early before serious degradation in job processing performance. At the slow start phase a straggling node is flagged.

Later at the accelerate phase, if a task is scheduled to a straggler node a redundant task called a clone is concurrently scheduled to the nearest non-straggler node with greatest distribution ratio. We use a topology-aware mechanism that prioritizes a faster node within the same rack. The nearest node here refers to the shortest hop count, location in cabinet, etc, depending with cluster set-up. Figure 11 shows the invocation of a clone of task-5 which was scheduled to straggler task tracker-1. The benefit of the concurrent scheduling of a straggler task and its clone, is that we can fuse the task scheduling together with the task pre-speculation. This helps to amortize the delay in speculative execution caused in pre-

Algorithm 2 Minimizing Delay in Clone Execution

```
Input: $t_i$, $_i$th task in scheduling window
  $DR_i$, Distribution ratio of task-tracker$_i$
  $SF_i$, Straggler Flag of task tracker node$_i$
  $Ct_i$, Clone of Task $t_i$
  $Rt_i$, Remaining time to complete task $t_i$
  $RCt_i$, Remaining time to complete cloned task $Ct_i$
Output: $Ct_i$_RESULT, key-value pair of task $Ct_i$;
for $Ct_i \neq 0$ do
    while $t_i.begin = true$ do
        if $SF = 1$ then
            $Ct_i$_Launch on node with max $DR_i$
        end
        else
            $Ct_i$_Launch = false
        end
    end
    return $Ct_i$_RESULT
end
```

vious works where a straggler is detected at the middle of task execution. In the event that two or more tasks are scheduled to a straggler node, then the tasks are shared equally among the nearest non-straggler nodes. By so doing we improve the speculative task execution and consequently improve job processing time. Algorithm 2 depicts the cloning process in EPPADS.

IV. ENERGY EFFICIENT DATA PROCESSING

In-order to achieve effective energy efficient computation, EPPADS has to find a way of maximizing energy conservation within the cluster. Recall that one of the idea behind EPPADS is to remove the heavy instrumentation or communication overheads between the job tracker and task tracker nodes. By so doing, this reduces, to some degree, the amount of power used by the cluster nodes. As depicted in our evaluation Section VI-B, removing the instrumentation overheads alone is not sufficient enough to achieve maximum power savings. To further reduce the energy consumption within the cluster, the PTT module exploits the information from the ETP module to provide power transition flags between effective and straggling nodes. Based on the power transition tags the PBEC module groups the nodes into two different power pools and then provide an easy and effective power transition management to a large number of nodes belonging to the same pool at the same time. We discuss in detail about the PTT and PBEC modules in Section IV-A and IV-B. Figure 12 depicts the proposed pool based power management design in our approach.

A. POWER TRANSITION TAGGING (PTT) MODULE

Key Idea (Flag Straggling Nodes With Low Power Tag): As discussed in Section III-D2, EPPADS leverages on the straggler flag in the ETP module to distinguish between the effective and straggling processing nodes. Since, for any task scheduled to a processing node marked as a straggler, we fuse task scheduling in parallel with its corresponding speculative task, then it is beneficial to prioritize node power to the effective nodes as there is high probability of the speculative task to finish earlier than the original straggling task. In this case, the PTT module queries the ETP module for the straggler flag, and based on the straggler flag, it tags all processing nodes marked as stragglers to a low power transition state, that is PTT = 0. We call this phenomenon *power transition tagging*. The power transition module then transit all the processing nodes with tag PTT = 0 to a low power state and all the processing nodes with tag PTT = 1 to a higher power state.

B. POOL BASED ENERGY CONSERVATION (PBEC) MODULE

Key Idea: Aggregate Power Savings With Pool Based Power Management: One challenge that is associated with cluster power management is how to easily and effectively transit the power for a large number of nodes. Power transition decisions per individual node may not give the best power saving results, and also the number of power transition message broadcast may complicate the whole power management process. In-order to ease the power transition management, EPPADS uses a dual based node pooling approach depicted in Figure 12.

We take advantage of the power transition tags to group all the nodes into two pools, the high power state pool (HPSP) and the low state power pool (LPSP). Nodes tagged with transition tag PTT = 1 are pooled in the high power state pool and vice versa for the nodes with transition PTT = 0. The power transition module then broadcast power transition signals to the 2 pools using a single message per pool, that is, the high power signal (P1) to the HPSP and the low power signal (P0) to the LPSP. With this approach EPPADS is able to give a simple power management design for large scale data clusters.

V. IMPLEMENTATION

A. MODIFICATION IN HADOOP

In-order to implement EPPADS in the existing Hadoop setup, we modified the Jobtracker and the task tracker implementations and replaced the TaskReport module with the ETP module in Hadoop version 2.7.2. We modified the JobTracker module to use 2 classes, the ssp class, and the acc_p class, for the slow-start and accelerate phase respectively. The ssp class polls each task tracker node computing time using FIFO based scheduling and communicates with the etp module which records and stores the compute capabilities of each task tracker in an array as described in Section III-C.

After that, the acc_p class uses the information stored in the etp array and schedules all the remaining tasks in a scheduling window instance in parallel. Due to the tagging information stored in the etp array, that indicates effective and straggling nodes in the cluster, acc_p class schedules a redundant task

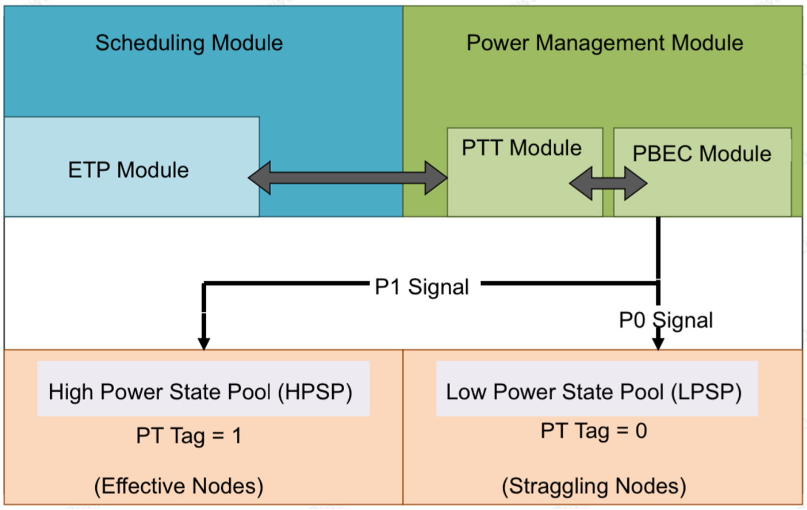


FIGURE 12. Proposed cluster power management design.

for tasks scheduled to nodes with straggler flag, SF = 1. This redundant task is scheduled to the node with highest distribution ratio.

Since we heavily depend on the ETP module for determining straggling nodes, we disabled most reporting and tracking functionalities on the JobTracker module to limit the communication overheads. We modified the TaskTracker module so that it only sends a few light-weight reporting messages that are polled by the ETP module.

B. POWER TRANSITION MODULE IMPLEMENTATION

Nowadays, modern processors are provisioned with a number of different operating frequencies which can be varied dynamically using dynamic voltage frequency scaling (DVFS) [37]–[40]. To this end, we implemented a software defined power management scheme that leverages the use of DVFS to transit the processing nodes between high and low frequencies.

To simplify our implementation, we designed our proposed power transition module to operate in dual mode frequency. We set the higher state frequency to the maximum 2100 MHz of our nodes and the lower state frequency to 250 MHz. The power transition module makes use of the ETP tagging information to group the nodes into two pools, HPSP and LPSP. A single frequency transition broadcast signal is then send to each of the pools to switch the operating frequency of the nodes to either the high state frequency or the low state frequency.

C. TESTBED SETUP

To evaluate the behavior and performance of our proposed EPPADS scheduler on a large scale data cluster, we configured the modified version of Hadoop [41] that fuses with our proposed EPPADS scheduler. We configured the modified hadoop (version 2.7.2) on a real cluster composed of 128 nodes running on Linux CentOS v7.3 and equipped with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz. The number of cores and memory allocated per each node is depicted in Table 1, which depicts heterogeneity among task tracker nodes. Each storage server is equipped with a total of 2TB ATA HDDs. On the Hadoop configuration, we set the replication factor of 3 and set the number of Map and reduce tasks per each node to 3.

Workloads: For all experiments irrespective of difference in workload type, we used a total of 10 TB input data. In cases where the input data was less than 10 TB we simply amplified the input data by a factor which makes it equal or close to 10 Terabytes. For experiments in Figure 13, Figure 14 and Figure 16 we used the Scan workload traces with the number of processing nodes equaling to 128. In Figure 15 experiment, we used a variety of workload traces from the HiBench suite and also with a fixed number of processing nodes equal 128. For the energy conversation experiments (Section VI-B) we specify under each subsection the type of workloads used.

To evaluate the effectiveness of the proposed EPPADS scheduler, we compared the following approaches:

- ***FIFO:*** This is the default scheduler in MapReduce framework. We used this as the baseline in our evaluation. FIFO schedules tasks equally between task tracker nodes irrespective of any difference in processing capabilities that might exist between the nodes.
- ***DynMon:*** To show the impact of node instrumentation in large scale processing clusters, we implemented a Dynamic Scheduling Algorithm which uses some instrumentation modules to monitor task trackers. The instrumentation modules regularly collect task tracker system level information and compute some utility value which is used to make scheduling decisions. Like all the existing Dynamic approaches it follows a First-In First-Out approach of scheduling job tasks. We called this Approach DynMon.

TABLE 1. Testbed setup with 128 heterogeneous task trackers (TTs) and a single powerful job tracker (JT) node.

	JT-Node	TT 1-16	TT 17-32	TT 33-48	TT 49-64	TT 65-80	TT 81-96	TT 97-112	TT 113-128
RAM (GB)	32	8	16	32	8	32	16	16	8
# Cores	32	16	16	32	16	16	8	16	16

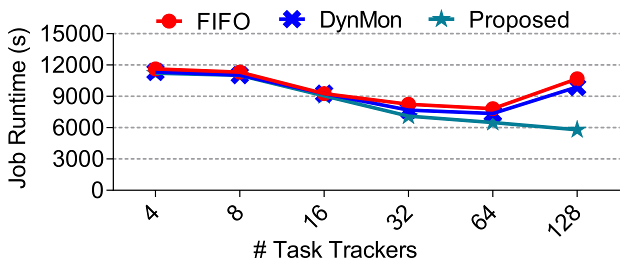


FIGURE 13. Scalabilty analysis.

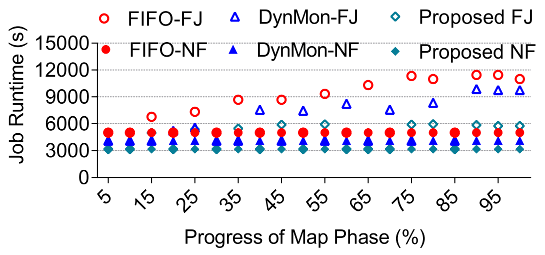


FIGURE 14. Effects of job slow down.

- ***Proposed:*** This refers to our proposed EPPADS implementation approach referred to in Section III.
- ***Proposed + PBPT:*** This refers to the pool-based power transition module built on top of the proposed EPPADS implementation.

In the next section, we evaluate our proposed EPPADS scheduling scheme. We first compare the performance with existing scheduling techniques and then secondly, we evaluate the power saving effectiveness of our pool-based power transition approach with existing approaches.

VI. EVALUATION

A. PERFORMANCE AND SCALABILITY EVALUATION

1) SCALABILITY ANALYSIS

To show the scalability effectiveness of our proposed EPPADS scheduler, we first measured the performance effect caused by increasing the number of the nodes in the cluster. We started with 4 task tracker nodes and doubled the nodes at each experimental run until the size depicted a large sized data processing cluster-128 task trackers. Figure 13 shows the results of our experiment. The results shows that as the number of task-tracker nodes increases, the job execution performance increases for all algorithms. This is because the contention in the cluster decreases with increasing number of task-trackers. However as we kept on adding the cluster processing nodes, we observed that the performance start to degrade at a certain point (after 64 nodes in our setup) in FIFO and the DynMon scheduling approaches. However, EPPADS performance does not suffer from the performance degradation due to continuous increase in cluster nodes.

This is because of the fact that, though the node instrumentation forms an integral part of dynamic and self adaptiveness of scheduling algorithm, it incurs monitoring overheads as the number of nodes to be monitored increases above a certain limit. This is mostly due to the frequent communication overheads between the centralized job tracker node and the processing task tracker nodes. However, due to the approach in which EPPADS eliminates the use of instrumentation modules, this communication overhead is greatly amortized, thus no degradation as the cluster size becomes huge.

2) EXECUTION TIME PERFOMANCE

Figure 14 shows the job execution time comparison between FIFO, DynMon and proposed EPPADS, at different stages in their Map phases. **NF** stands for No-Fault in which all the tasks associated with that job executed in normal time without any delay. **FJ** stands for Faulty Job in which one or more tasks contributed to the slow down of that specific job. Each plotted point on the graph depicts a single job and its execution time at different stages of the map phase. The solid plotted points depict jobs without failures and the non-solid plotted points represent jobs which incurred some failure or slow down during their processing.

We observed a big job execution performance degradation of 2× or more when there is a job failure in the baseline FIFO scheduling approach and the DynMon scheduling approach. We attribute this to the delayed speculative task invocation caused by the straggler hunting approach. Mostly a speculative task is invoked at a later stage when the primary task has been running for quite a number of seconds. The Job tracker later observes that the task is straggling and decides to launch a redundant task. This speculative task invocation is often too late to minimize the slowdown to job processing performance.

However, the job slowdown is significantly less in our proposed EPPADS scheduling approach. This is due to the dual-phase approach used in our approach. The straggler detection is determined in the Slow Start Phase and then any task which are scheduled to nodes marked as stragglers has a corresponding cloned task which is scheduled at the same time with the straggler task. This drastically reduces the delay in speculative task invocation caused in current scheduling approaches. EPPADS reduces the delayed speculative task invocation by almost 2×, thereby drastically improving the job processing performance.

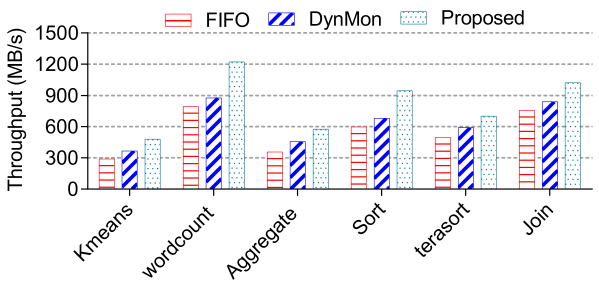


FIGURE 15. Performance with HiBennch suite.

3) JOB THROUGHPUT ANALYSIS

Figure 15 shows the job throughput comparison between the baseline FIFO scheduler, Adaptive Dynamic Approach with monitoring - DynMon, and the proposed EPPADS scheduler. To do an intensive evaluation of our proposed approach, we employed the HiBench benchmark suite which consists of a number of benchmark workloads. We observed that our proposed EPPADS scheduler have an average throughput performance improvement of 30% and 22%, compared to the baseline and DynMon scheduling approaches, respectively.

The reason behind this is the concurrent scheduling of job tasks in the accelerate phase (AccP) of EPPADS using a single scheduling I/O. One might argue that the scheduling in the slow-start phase degrades the job processing throughput, but the worst case scenario is equal with the best case of baseline FIFO and DynMon cases which all schedules the task in a sequential First-In First-Out Approach. However the AccP of EPPADS increases the job throughput significantly as multiple jobs are scheduled at once. Furthermore, the concurrent scheduling of straggling tasks with their clones increases the performance as this helps to amortize the speculative execution delay.

4) JOB QUEUING TIME

Another important factor in determining job processing performance is the amount of time a job spends in the queue before it can be scheduled for processing. For each approach we profiled the job waiting time from the time the job tracker node receives a job request until the first task of that job is scheduled for processing in any of the task tracker nodes. Figure 16 shows the results of this experiments. We observed that when there is no fault in the processing nodes, the job waiting time in EPPADS is less as compared to baseline FIFO and DynMon approaches. However this gap increases when there are job faults within the processing nodes. We observe that in the existence of job faults, the job waiting time in FIFO and DynMon approaches increases dramatically.

This is mostly due to the following two factors; (i) The serialized scheduling of job tasks in FIFO and DynMon approaches increases the job waiting time. Furthermore, when multiple job failure occurs, other jobs has to wait whilst the failed jobs execute extra speculative tasks. (ii) To make it worse there is a delay in launching these extra speculative tasks in the FIFO and DynMon scheduling

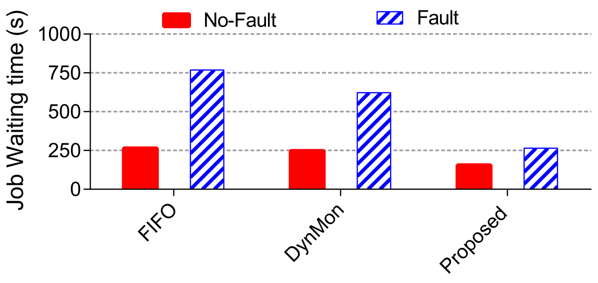


FIGURE 16. Comparison of job waiting time.

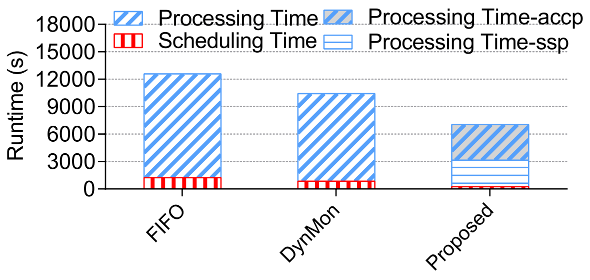


FIGURE 17. Job process time breakdown.

approaches. On the other hand, the job waiting time in the proposed EPPADS scheduler does not increase significantly. This is because of the early detection of straggler nodes in EPPADS. Also, the job waiting time is reduced significantly due to the scheduling of multiple job tasks in parallel during the accelerate phase. The reduction of job waiting time in the queue assist in boosting the job processing performance.

5) EFFECTS OF THE DUAL-PHASE SCHEDULING

In this subsection we evaluate the effect of the dual phase pipelining in terms of the aggregate time spend during the scheduling stage and task execution phase. Figure 17 shows the aggregated time that was taken during scheduling and task processing phases using the Scan workload from the HiBench benchmark with 10 TB of data as input. i) The results shows that our proposed EPPADS has a high scheduling performance as compared to the baseline and Dynamic Monitoring approaches. ii) Even if EPPADS adopts a sequential scheduling approach to schedule the first initial tasks in the slow-start phase (SSP), the results shows that during the accelerate phase, our proposed scheduler speed-up the task processing by scheduling multiple tasks in parallel using a single scheduling I/O. By using the dual phased scheduling in the proposed design we archive a job performance increase of 43% and 32% as compared to the baseline and DynMon approaches, respectively.

B. ENERGY EFFICIENCY ANALYSIS

In this section we evaluate the effectiveness of EPPADS towards attaining energy efficient computing. To achieve this we first measure the power consumption from the computation of a similar task on all approaches. We compare the power consumption of the baseline, DynMon approach, proposed approach (EPPADS without the pool-based power

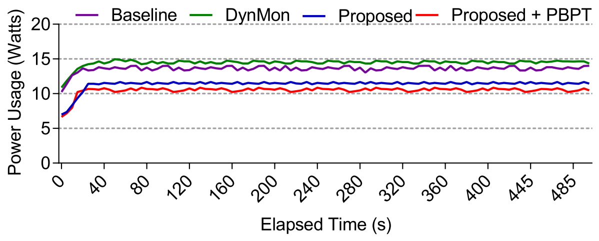


FIGURE 18. Snapshot of cluster power usage over 500 seconds.

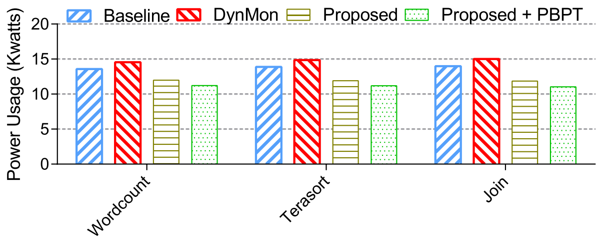


FIGURE 19. Power consumption with different workloads.

transitioning modules) and proposed + PBPT (EPPADS provisioned with the pool-based power transition module).

1) ENERGY CONSUMPTION TREND

To roughly understand the energy consumption effectiveness of the different approaches, we first took a snapshot of the first 485 seconds of running a job using the Sort workload from the Hi-Bench suite. From Figure 19 (a), we observed that the energy consumption trend from our proposed approach outperforms the baseline and the DynMon approaches. However, provisioning the pool-based power transitioning module in the cluster will further increase the energy saving effectiveness as shown by the trend of the proposed + PBPT in Figure 19.

This is because the proposed EPPADS is free from the heavy instrumentation modules that is associated with the baseline and DynMon Approaches. Moreover, when provisioned with the pool-based power transitioning module, EPPADS lowers the power usage of none performing nodes in the cluster, which helps to reduce power consumption. This benefit becomes so significant when the cluster scales to hundreds of nodes.

2) ENERGY EFFECTIVENESS WITH VARIOUS WORKLOADS

To do a rigorous analysis of the energy efficient computing of EPPADS, we evaluate the energy consumption with different workload characteristics. In this experiment we used the Wordcount, Terasort and Join workloads from the Hi-Bench suite to analyze the energy consumption between the different approaches. Figure 19 shows the average power consumption of the different approaches with various workloads.

The results shows that our proposed approach provides consistent energy efficient computing in all the various workload characteristics. In all cases, the proposed EPPADS approach shows superior energy savings as compared to the baseline and DynMon approaches. Also, in all the different workload characteristics, the proposed + PBPT approach

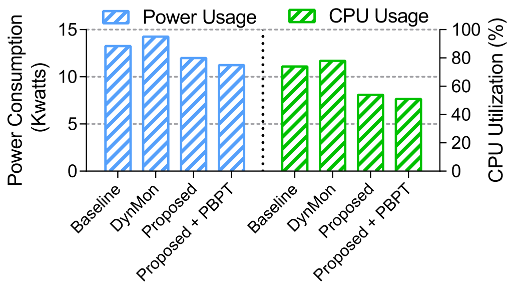


FIGURE 20. Energy consumption and CPU utilization.

further decreases the cluster power consumption as compared to all other approaches. This result shows that our proposed design can provide for the much needed energy saving computing in large scale cluster environments.

3) POWER CONSUMPTION AND CPU UTILIZATION

In this section we analyze the relationships between the power consumption and the CPU utilization for the different approaches using the Sort workload from the Hi-Bench suite. Figure 20 shows the correlation between the energy consumption and CPU utilization of the different scheduling approaches. The results depict a direct proportional relationship between the power consumption usage and the CPU utilization. The more the CPU utilization, the more the power consumption in the cluster.

This is evidenced by the results of the baseline and DynMon approaches which shows high power consumption and high power CPU utilization. On the contrary, our proposed approach shows both low power consumption and low CPU resource utilization. The reason behind this is that the power consumption of the CPUs in the servers contributes to the larger portion of energy consumption in the cluster. Therefore, the more the CPU utilization of cluster nodes, the more the power consumption in the server.

Generally, task schedulers over provision resources for job processing. However, over provisioning of CPU resources to non performing nodes can lead to unnecessary over-utilization of power. EPPADS uses the pool-based power transitioning (PBPT) scheme to switch all nodes marked as stragglers ($PT = 0$) to a low power state, thereby amortizing the over-utilization of CPU resources. By so doing, the overall power consumption of our proposed approach provisioned with PBPT is reduced significantly as compared to the proposed without PBPT, the baseline and the DynMon approaches altogether. In overall, EPPADS with PBPT shows an average power savings between 15% to 20% as compared to the other approaches.

VII. RELATED WORKS

The way in which jobs are scheduled and processed play a vital role in boosting up the job execution performance in large scale data processing clusters. The baseline FIFO scheduling algorithm schedules the tasks equally among all the available task tracker nodes [5]. On the flip side, when the baseline FIFO scheduler is subjected to clusters provisioned with heterogeneous nodes, it results in significant performance degradation. This is because of the imbalances in computational power that exists between the nodes.

Quite a number of previous works on dynamic and self adaptive schedulers [6]–[10], [18], [20], [42] has been investigated in order to limit the job execution performance degradation in heterogeneous cluster environments. However, they adopted node instrumentation and monitoring to frequently collect the system level information of task tracker nodes which is consequently used as a basis to make scheduling decisions and speculative task invocations. In spite of this improvement, additional instrumentation can cause system performance degradation as the number of task tracker nodes increases. Furthermore, all these prior studies follow a sequential task scheduling approach in which a successive task can only be scheduled only if one of the already scheduled tasks is completed, which is a major constraint in achieving maximum job processing performance.

The authors of [2], [4], [18], [21], [43]–[46], proposed proactive straggler mitigation techniques to minimize job processing time in production clusters. The technique of straggler "hunting" was used in these works which often wait until tasks are straggling so as to invoke a speculative task. However, these solutions suffer from enough information needed to separate between slow nodes and faster nodes. These approaches are likely to cause some unnecessary over-utilization of resources without significant improvement in job completion performance. This is likely due to scheduling of speculative tasks onto already slower nodes. To avoid such scenarios our proposed EPPADS scheduler detects the stragglers at an early stage of the slow-start phase and speculative tasks are launched on the nearest effective processing node using a topology aware policy (preference given to node in the same rack).

On the contrary, energy efficient computing is an important issue in amortizing the costs associated with data computation, especially in large scale cluster environments. A lot of previous works, [24], [28]–[34], [47]–[49], tried to provision some techniques that amortize the energy costs associated with intensive computation in large data centers. Works such as [47]–[49] achieved the benefits of combining scheduling and power management, but the drawback is that they assumed this job scheduling and power management in a uni-processor environment. Most of the other works [24], [28]–[34], tried to leverage on idle time slots, which might be challenging to obtain in situations where there is scarcity of idle times. GreenHDFS [29], amplifies this idle time by separating hot and cold data and then migrate the cold data into servers reserved for data that is not frequently accessed. Consequently, they will transition these servers allocated for cold data into low power states.

However, it presents a challenge that, in-order for the migration of cold data to cold region, it requires periods of inactivity. This might not result in the anticipated maximum energy savings. To resolve this challenge, our proposed solution uses straggler flagging, not idleness to transit the servers flagged as stragglers into low power state. In periods of idleness EPPADS also leverages on the idle states of nodes to further reduce energy consumption in the cluster. Collectively, none of the previous works implemented a multi-task scheduling approach which is employed in our proposed approach. The phase based implementation in EPPADS greatly improves the job execution performance as evidenced by our evaluation results.

VIII. CONCLUSION

This paper presents EPPADS, a lightweight and high performance job scheduler for improving job processing performance in large scale data processing clusters. With its dual phased scheduling approach, EPPADS can drastically mitigate the job straggler problem in production clusters. This is achieved through the early detection of stragglers in the slow-start phase (SSP) and the speed-up in scheduling performance that occurs at the acceleration phase(AccP). Furthermore, EPPADS achieves efficient energy computing by coordinating the scheduling together with its power saving management scheme. Our experimental evaluation shows that EPPADS achieves both significant performance increase and energy savings and can guarantee the much needed energy savings in large scale clusters.

REFERENCES

[1] P. Swabey. *The Data Deluge: Five Years On*. Accessed: May 2, 2019. [Online]. Available: https://www.slideshare.net/economistintelligenceunit/the-data-deluge-five-years-on

[2] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using Mantri," in *Proc. 9th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Vancouver, BC, Canada, Oct. 2010, pp. 265–278.

[3] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Lombard, IL, USA, 2013, pp. 185–198.

[4] H. Xu and W. C. Lau, "Task-cloning algorithms in a mapreduce cluster with competitive performance bounds," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, Jun./Jul. 2015, pp. 339–348.

[5] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[6] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proc. 8th USENIX Conf. Operating Syst. Design Implement.*, Berkeley, CA, USA, 2008, pp. 29–42.

[7] H.-H. You, C.-C. Yang, and J.-L. Huang, "A load-aware scheduler for mapreduce framework in heterogeneous cloud environments," in *Proc. ACM Symp. Appl. Comput.*, New York, NY, USA, Mar. 2011, pp. 127–132.

[8] S.-J. Yang, Y.-R. Chen, and Y.-M. Hsieh, "Design dynamic data allocation scheduler to improve mapreduce performance in heterogeneous clouds," in *Proc. IEEE 9th Int. Conf. e-Business Eng. (ICEBE)*, Sep. 2012, pp. 265–270.

[9] X. Sun, C. He, and Y. Lu, "ESAMR: An enhanced self-adaptive mapreduce scheduling algorithm," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2012, pp. 148–155.

[10] A. Rasooli and D. G. Down, "COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems," *Future Gener. Comput. Syst.*, vol. 36, pp. 1–15, Jul. 2014.

[11] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale mapreduce workloads with significant interactive analysis," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, New York, NY, USA, Apr. 2012, pp. 43–56. doi: 10.1145/2168836.2168842.

[12] J. A. Aroca, A. Chatzipapas, A. F. Anta, and V. Mancuso, "A measurement-based analysis of the energy consumption of data center servers," in *Proc. 5th Int. Conf. Future Energy Syst.*, New York, NY, USA, Jun. 2014, pp. 63–74. doi: 10.1145/2602044.2602061.

[13] L. Alsbatin, G. Öz, and A. H. Ulusoy, "An overview of energy-efficient cloud data centres," in *Proc. Int. Conf. Comput. Appl. (ICCA)*, Sep. 2017, pp. 211–214.

[14] H. Yuan, C.-C. J. Kuo, and I. Ahmad, "Energy efficiency in data centers and cloud-based multimedia services: An overview and future directions," in *Proc. Int. Conf. Green Comput.*, Aug. 2010, pp. 375–382.

[15] J. K. Verma and C. P. Katti, "A comparative study into energy efficient techniques for cloud computing," in *Proc. IEEE 2nd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2015, pp. 2062–2067.

[16] Y. Qiu, C. Jiang, Y. Wang, D. Ou, Y. Li, and J. Wan, "Energy aware virtual machine scheduling in data centers," *Energies*, vol. 12, no. 4, p. 646, 2019. [Online]. Available: https://www.mdpi.com/1996-1073/12/4/646

[17] P. Hamandawana, R. Mativenga, S. J. Kwon, and T.-S. Chung, "EPPADS: An enhanced phase-based performance-aware dynamic scheduler for high job execution performance in large scale clusters," in *Proc. 24th Int. Conf. Database Syst. Adv. Appl.*, G. Li, J. Yang, J. Gama, J. Natwichai, and Y. Tong, Eds. Cham, Switzerland: Springer, 2019, pp. 140–156.

[18] J. H. Hsiao and S. J. Kao, "A usage-aware scheduler for improving mapreduce performance in heterogeneous environments," in *Proc. Int. Conf. Inf. Sci., Electron. Elect. Eng. (ISEEE)*, vol. 3, Apr. 2014, pp. 1648–1652.

[19] H. Fu, H. Chen, Y. Zhu, and W. Yu, "FARMS: Efficient mapreduce speculation for failure recovery in short jobs FARMS: Efficient mapreduce speculation for failure recovery in short jobs," *Parallel Comput.*, vol. 61, pp. 68–82, Jan. 2017.

[20] A. Rasooli and D. G. Down, "A hybrid scheduling approach for scalable heterogeneous Hadoop systems," in *Proc. SC Companion, High Perform. Comput., Netw. Storage Anal.*, Nov. 2012, pp. 1284–1291.

[21] Q. Chen, C. Liu, and Z. Xiao, "Improving mapreduce performance using smart speculative execution strategy," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 954–967, Apr. 2014.

[22] H. Xu, W. C. Lau, Z. Yang, G. de Veciana, and H. Hou, "Mitigating service variability in mapreduce clusters via task cloning: A competitive analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2866–2880, Oct. 2017.

[23] A. S. G. Andrae and T. Edler, "On global electricity usage of communication technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[24] M. Anan and N. Nasser, "SLA-based optimization of energy efficiency for green cloud computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[25] J. Wu, S. Rangan, and H. Zhang, *Green Communications: Theoretical Fundamentals, Algorithms and Applications*. Boca Raton, FL, USA: CRS Press, 2012.

[26] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Greening big data," *IEEE Syst. J.*, vol. 10, no. 3, pp. 873–887, Sep. 2016.

[27] C. Ge, Z. Sun, N. Wang, K. Xu, and J. Wu, "Energy management in cross-domain content delivery networks: A theoretical perspective," *IEEE Trans. Netw. Service Manag.*, vol. 11, no. 3, pp. 264–277, Sep. 2014.

[28] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan, "Robust and flexible power-proportional storage," in *Proc. 1st ACM Symp. Cloud Computing*, New York, NY, USA, Jun. 2010, pp. 217–228. doi: 10.1145/1807128.1807164.

[29] R. T. Kaushik and M. Bhandarkar, "GreenHDFS: Towards an energy-conserving, storage-efficient, hybrid Hadoop compute cluster," in *Proc. Int. Conf. Power Aware Comput. Syst.*, Berkeley, CA, USA, 2010, pp. 1–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924927

[30] D. Cheng, X. Zhou, P. Lama, M. Ji, and C. Jiang, "Energy efficiency aware task assignment with DVFS in heterogeneous Hadoop clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 70–82, Jan. 2018.

[31] V. Getov, A. Hoisie, and P. Bose, "New frontiers in energy-efficient computing [Guest editors' introduction]," *Computer*, vol. 49, no. 10, pp. 14–18, Oct. 2016.

[32] D. Cheng, P. Lama, C. Jiang, and X. Zhou, "Towards energy efficiency in heterogeneous Hadoop clusters by adaptive task assignment," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, Jul. 2015, pp. 359–368.

[33] W. Liu, H. Li, and F. Shi, "Energy-efficient task clustering scheduling on homogeneous clusters," in *Proc. Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, Dec. 2010, pp. 381–385.

[34] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No 'power' struggles: Coordinated multi-level power management for the data center," *SIGARCH Comput. Archit. News*, vol. 36, no. 1, pp. 48–59, Mar. 2008. doi: 10.1145/1353534.1346289.

[35] L. Mashayekhy, M. M. Nejad, D. Grosu, Q. Zhang, and W. Shi, "Energy-aware scheduling of mapreduce jobs for big data applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 10, pp. 2720–2733, Oct. 2015.

[36] Y. Shao, C. Li, W. Dong, and Y. Liu, "Energy-aware dynamic resource allocation on Hadoop YARN cluster," in *Proc. IEEE 18th Int. Conf. High Perform. Comput. Commun., IEEE 14th Int. Conf. Smart City, IEEE 2nd Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Dec. 2016, pp. 364–371.

[37] J. Lee, B.-G. Nam, and H.-J. Yoo, "Dynamic voltage and frequency scaling (DVFS) scheme for multi-domains power management," in *Proc. IEEE Asian Solid-State Circuits Conf. (ASSCC)*, Nov. 2007, pp. 360–363.

[38] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proc. Int. Conf. Power Aware Comput. Syst.*, Berkeley, CA, USA, 2010, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924921

[39] S. Wang, Z. Qian, J. Yuan, and I. You, "A DVFS based energy-efficient tasks scheduling in a data center," *IEEE Access*, vol. 5, pp. 13090–13102, 2017.

[40] D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Examples of Voltage and Frequency Scaling Design*. Boston, MA, USA: Springer, 2007, pp. 139–154. doi: 10.1007/978-0-387-71819-4_10.

[41] *Hadoop*. Accessed: May 8, 2019. [Online]. Available: http://hadoop.apache.org/

[42] *Fair Scheduler*. Accessed: May 8, 2019. [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html

[43] H. Chang, M. Kodialam, R. R. Kompella, T. V. Lakshman, M. Lee, and S. Mukherjee, "Scheduling in mapreduce-like systems for fast completion time," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 3074–3082.

[44] F. Chen, J. Liu, and Y. Zhu, "A real-time scheduling strategy based on processing framework of Hadoop," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Jun. 2017, pp. 321–328.

[45] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous Hadoop clusters," in *Proc. IPDPSW*, Apr. 2010, pp. 1–9.

[46] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 61–74, Mar. 2012. doi: 10.1145/2189750.2150984.

[47] P. Singh and N. Hailu, "Energy-aware online non-clairvoyant multiprocessor scheduling: Multiprocessor priority round robin," *IET Comput. Digit. Techn.*, vol. 11, no. 1, pp. 16–23, Jan. 2017.

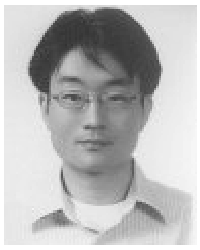
[48] P. Singh and B. Wolde-Gabriel, "Executed-time round Robin: EtRR an online non-clairvoyant scheduling on speed bounded processor with energy," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 29, no. 1, pp. 74–84, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S131915781630009X

[49] P. Singh, "Energy efficient non-clairvoyant scheduling for unbounded-speed multi-core machines," *Comput. Elect. Eng.*, vol. 67, pp. 441–453, Apr. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0045790617305475

PRINCE HAMANDAWANA received the B.Sc. degree (Hons.) in computer science from the National University of Science and Technology (NUST), Bulawayo, Zimbabwe, in 2010. He is currently pursuing the Ph.D. degree in computer engineering with Ajou University, Suwon, South Korea, where he is a member of the Database and Dependable Computing (DBDC) Lab. He worked as a Network Engineer in two of the Zimbabwean leading service providers, Econet Wireless, from 2008 to 2011, and Liquid Telecom, from 2011 to 2016. His research interests include distributed and parallel storage systems, and GPU assisted cluster-wide data deduplication.

RONNIE MATIVENGA received the B.Sc. degree (Hons.) in computer science from the National University of Science and Technology (NUST), Bulawayo, Zimbabwe, in 2010. He is currently pursuing the Ph.D. degree in computer engineering with Ajou University, Suwon, South Korea. His current research interests include emerging nonvolatile storage arrays, building a high-performance, reliable SSD-based storage systems, large database systems, distributed systems, and simulation tools.

SE JIN KWON (M'16) received the M.S. and Ph.D. degrees in computer engineering from Ajou University, South Korea, in 2008 and 2012, respectively, where he was a Research Professor with the Department of Information and Computer Engineering, from 2013 to 2016. He was a Postdoctoral Researcher with the University of California, Santa Cruz, in 2016. He is currently an Assistant Professor with the Department of Computer Engineering, Kangwon National University, Kangwon-do, South Korea. His current interests include nonvolatile memory systems, reliable storage systems, and large database systems.



TAE-SUN CHUNG received the B.S. degree from KAIST, Daejeon, South Korea, in 1995, and the M.S. and Ph.D. degrees from Seoul National University, Seoul, South Korea, in 1997 and 2002, respectively, all in computer science. He is currently a Professor with the Department of Computer Engineering, Ajou University, Suwon, South Korea. His current research interests include flash memory storages, XML databases, and database systems.