An Approach to Optimise Resource Provision with Energy-awareness in Datacentres by Combating Task Heterogeneity

John Panneerselvam, Lu Liu and Nick Antonopoulos

Abstract—Cloud workloads are increasingly heterogeneous such that a single Cloud job may encompass one to several tasks, and tasks belonging to the same job may behave distinctively during their actual execution. This inherent task heterogeneity imposes increased complexities in achieving an energy efficient management of the Cloud jobs. The phenomenon of a few proportions of tasks characterising increased resource intensity within a given job usually lead the providers to over-provision all the encompassed tasks, resulting in majority of the tasks incurring an increased proportions of resource idleness. To this end, this paper proposes a novel analytics framework which integrates a resource estimation module to estimate the resource requirements of tasks a priori, a straggler classification module to classify tasks based on their resource intensity, and a resource optimisation module to optimise the level of resource provision depending on the task nature and various runtime factors. Performance evaluations conducted both theoretically and through practical experiments prove that the proposed methodology performs better than the compared statistical resource estimation methods and existing models of straggler mitigation, and further demonstrate the effectiveness of the proposed methodology in achieving energy conservation by postulating appropriate level of resource provisioning for task execution.

Index Terms—Energy-aware systems, Interactive data exploration and discovery, Power Management, Servers

1 Introduction

IN the Cloud Computing concept, user request arrives in the form of jobs encompassing one to several number of tasks. Such tasks are usually processed in various servers across the datacentres. Schedulers usually allocate the incoming tasks onto isolated containers with a predefined level of CPU, memory and disk space resources for the LXCs (Linux Containers) or VMs (Virtual Machines) to consume of the physical resources. This predefined level of resource provision is usually the maximum level of allowed resources for the LXCs or VMs to consume. It is common that the resource requirements of the tasks are often over-estimated in an attempt to avail resources at a level which does not compromise the task execution resulting from resource scarcity. There are two immediate implications - over-estimation of resource requirements usually incurs a significant proportion of resource idleness and under-estimation usually leads to task terminations directly affecting the Quality of Service (QoS). While it is being argued that the level of provisioned resources usually far exceed [1] the actual requirements of the tasks, resource idleness resulting from over-estimation of resource requirements can be minimised when the resource requirements of the jobs are estimated before execution. But accurately predicting the resource requirements of the jobs include various challenges, one of them is the inner heterogeneity of tasks within jobs in terms of their resource requirements, task duration, resource intensity etc. Completion of a given job can only be assured when all of its encompassing tasks are successfully executed. Thus, a task-level optimisation is of the interests of energy efficiency to achieve an overall energy optimisation of the entire job.

Giving a special emphasis to this inherent task heterogeneity, this paper postulates the phenomenon of a few tasks within a single job exhibiting a resource intensiveness as an increased multiple of majority of the remaining co-located tasks as energy-aware straggling behaviour of tasks. Since these straggling behaviours of tasks are not previously known, all the encompassed tasks within a given job over-provisioned at a level that can guarantee termination-less execution of the energy-aware stragglers. The worse energy implication can occur when the energy-aware stragglers are only a marginal proportion of the total tasks, thus leaving a larger proportion of resources allocated to the non-straggling tasks unutilised.

Existing works to date addressing the inner task heterogeneity [2-6] focused only on long tails, whereby a few tasks usually exhibit a duration as an increased multiple than those of the majority of the remaining tasks within the same job. Stragglers are of two types based on their locality, such as the node-level stragglers and the task level stragglers. Whilst the former is usually identified among the running physical servers the latter results depending on the nature, characteristics and requirements of the tasks. Server nodes exhibiting poor performance and declining process capability can cause node-level stragglers, naturally tasks scheduled on to such node-level stragglers suffer performance constraints, possible terminations and prolonged execution duration than anticipated. Tasks-level stragglers naturally exhibit a varied execution behaviour than the other co-located tasks with-

- *The authors are with the Department of Engineering and Technology, University of Derby, Derby, DE22 1GB, United Kingdom,*
- *Email: {j.panneerselvam, l.liu, n.antonopoulos}@derby.ac.uk*

in the same job in terms of their resource consumption, execution duration etc. Whilst the node-level stragglers can be mitigated by avoiding scheduling jobs onto such poor nodes, task-level stragglers pose an increased management complexity since they can hardly be identified before they occur. Further the existence of the relationship between node-level and stragglers is trivial, such that a common task-level straggler in two different execution instances of the same job can behave distinctly on alternative server nodes. Node-level long tail stragglers usually affect the overall completion time of the entire job but energy-aware stragglers, posing a significant energy implication, have not gained suffice importance so far.

To this end, this paper proposes a novel analytics framework for mitigating the energy-aware stragglers within jobs to optimise the level of resource provisioning to reduce energy expenditures incurred in the form of idle resource proportions during task execution. The proposed framework encompasses three cascaded modules. The novelty beyond the state-of-the-art models of straggler mitigation lies in the fact that the inherent task heterogeneity has been given primary importance and every individual task within a given job has been uniquely treated to optimise their resource provision. Important contribution of this paper includes the following:

- Firstly, a resource estimation module has been developed to estimate the resource intensiveness of every individual task within jobs. This resource estimation module exploits the inherent periodical effects among the Cloud workloads and predicts the resource requirements of every individual tasks within jobs with reliable level of accuracy.
- Secondly, a straggler classification framework has been integrated to classify tasks within a given job as potential energy-aware stragglers and non-stragglers. This classification framework exploits the historical behaviours of the tasks from suitable execution instances and classifies tasks based on their anticipated resource intensiveness and execution trend.
- Finally, a resource optimisation module has been incorporated to optimise the level of resources availed to tasks within jobs to achieve termination-less execution with reduced proportion of resource idleness. This resource optimisation module considers various run-time execution factors and the nature of the tasks within jobs to recommend the most appropriate level of resource provision to conserve energy.

The remainder of this paper is organised as follows: Section 2 reviews the related works of straggler mitigation techniques for energy efficiency and Section 3 details our proposed analytics framework. The resource estimation module, the straggler classification module and the resource optimisation module are described in Section 4, 5 and 6 respectively. Section 7 is covered with the performance evaluations and discussions, and Section 8 concludes this paper along with outlining our future works.

2 Related Works

Energy management has been approached from various perspectives [7-15] such as energy-aware resource scheduling, server switching, task allocation, live migration, VM consolidation etc. This paper focuses on approaches those aim to mitigate task heterogeneity, particularly the straggling behaviours of tasks within jobs. Straggler identification is growing importance as an integral component in the context of energy management in datacentres, since identifying the straggling tasks and treating them accordingly benefits not only to ensure completion of the straggling tasks but also to restrain the stragglers from consuming more server resources resulting from termination driven resubmissions and prolonged execution.

Stragglers are usually identified based on a defined threshold to locate tasks exhibiting an abnormal execution behaviour than those of the other co-located tasks within the same job. One of the most commonly considered execution metrics is the task duration for characterising long tail stragglers [3, 16]. Most of the existing works adopts this duration threshold as a typical value of tasks exhibiting 50% longer [5] than the average duration for straggler classification. Static threshold for straggler identification based on their execution duration may not scale well for Cloud workloads because of the task heterogeneity. Such static thresholds are computed as a temporal difference of the duration between a running task and the average task duration during execution. Computing this temporal difference is possible only during the run time.

Existing strategies of mitigating stragglers during runtime include speculative execution, which is commonly being used in Hadoop and Map Reduce environments and in production clusters [5] such as Google and Bing. This strategy increases the probability of early completion of the straggling tasks by creating multiple replicas of the long tails, the replica completed first is stored for task completion and the remaining replicas are terminated. Though, duplicating the same execution instance demands more resource allocation and hence incur excess energy consumption. Furthermore, resources spent on all other replicas other than the one completed first are needless energy expenditures. The execution of such replicas is not known a priori which doubts their completion rate in a way that the created replicas might be even execute longer and consume more resources than the actual stragglers. Another drawback of speculative execution is the definition of the threshold to trigger replicas. If the replicas are created earlier during the execution, there is still a possibility for the actual task to progress smoothly, in which case the created replicas are simply terminated without extracting any useful information services. If the replicas are created later in the execution, changes are the created replicas may not finish on time causing unnecessary delays in the actual job completion, whereby wasting not only the resources spent on the straggling task but also on the created replicas. Most of the existing works identifies runtime stragglers during the later stage [3] of the actual execution's lifecycle, causing needless replicas. Since the threshold identification for creating replicas can only be achieved during runtime, a pro-active measurement is not possible with speculative execution. Furthermore, creating replicas when the system utilisation [5] is higher may pose threat of straggling the created replicas.

A progress score based threshold [5] computes the task progress score as the ratio of proportions of tasks completed to the proportions remaining for execution, and classifies a task as straggler when its progress score falls behind a defined threshold than the average progress score of the given job. Hadoop scheduler [17] uses this progress score to classify tasks as stragglers when their progress score falls behind 80% of the average progress score of the job. Both the task progress rate and the process bandwidth within an execution phase [18] have been utilised to identify straggling tasks. The process speed of tasks has been predicted during the runtime and the remaining task execution time has been computed [19] for triggering a new set of speculative execution for maximising the cost performance. When the progress rate, determined based on a slow-node threshold, of a given task falls behind 50% of the other co-located tasks then the corresponding task is reported as stragglers. Whilst a higher threshold delays straggler identification and may not identify any stragglers at all, a lower threshold might lead to false positives. LATE [19] speculates the tasks those are estimated to finish farther into the future for reducing the job response time. Since tasks within a single job do not progress at a stable rate, process bandwidth might not provide suffice inferences for accurate estimation of stragglers. Furthermore, tasks within a single job may not start at the same time in the case of jobs with cascaded tasks, whereby a common progress rate cannot be applied to all the tasks within a single job.

Straggler tasks are characterised as exhibiting a normalised duration [20] as an increasing multiple of the median of the normalised duration of the other co-located tasks within the same job, with the normalised duration computed as the ratio of task execution time to the amount of work done, but assuming a static median may not scale well in spite of the task heterogeneity. Mantri is a straggler mitigation approach [6, 21] focused on conserving the computing resources of the server nodes. It employs a strategy of backing up tasks for multiple execution at an early stage and kills the original task instance when the cluster becomes busy and restarts the task in a different node instance. Though Mantri addresses conserving energy by an early speculation of straggling tasks, terminations are unavoidable at the process level. Furthermore, the newly created instances of the terminated tasks are not often guaranteed to complete within a defined time-scale, causing long tails of the created replicas.

Addressing the issues of speculative execution, a co-worker based scheme [22] has been proposed for mitigating stragglers to shorten the job completion time with less resource consumption. This approach transfers a portion of data from the straggling tasks to the co-worker, whereby the workload is shared by the co-worker. Whilst addressing energy efficiency, data transfer and migration costs are unavoidable in this scheme. Another way of avoiding runtime stragglers is server blacklisting [2, 4, 23], by avoiding scheduling tasks onto the nodes those spotted as stragglers in the past. Though this scheduling strategy helps to avoid node-level stragglers, it is not always true that a straggler node in the past should remain the same in the future, this increases the probability of classifying a non-straggling server node as straggler.

Most of the existing works on straggler identification are focused on identifying and mitigating the stragglers only during the actual execution. It is commonly being argued that straggling tasks can only be identified after a few minutes of the task execution. Furthermore, the state-of-the-art straggler identification techniques are focusing more towards long tails, leaving the resource intensity of the tasks unnoticed. Though identifying task-level stragglers before the actual execution might facilitate better management of stragglers, forecasting the task-level stragglers before the actual execution is still an aspiration to date which laid the foundation for this research paper, aimed at an energy management strategy driven by a prior estimation of the task behaviours at the datacentres.

3 Analytics Framework

The proposed analytics framework is illustrated in Fig. 1, which encompasses various components for functionalities such as sample selection, imputation, execution trend analysis, resource estimation, straggler classification and resource level optimisation. The primary purposes of these encompassed functionalities are detailed as follows.

Sample selection: The primary objective of the sample selection component is to choose the most appropriate samples from historical traces for analytics. Most suitable historical samples are chosen and validated based on a statistical similarity measure with the actual sample.

Imputation: It is common that the extracted historical samples might be incomplete in such a way that the execution profile for certain tasks within a given job might not be available. The imputation module is responsible to obtain a complete execution profile for jobs by inferring and imputing the missing values.

Execution trend analysis: An analysis in now conducted on the complete execution profile of the validated historical samples to observe the actual historical execution behaviours of the jobs.

Resource estimation: Driven by the descriptive analytics of the historical execution instances, the anticipated resource consumption levels of every individual task within a given job are estimated for resource provision.

Straggler classification: Based on their resource intensiveness, every task within a given job are subjected to a classification framework to forecast the anticipated execution behaviour of tasks within jobs. This classification is intended to isolate energy-aware stragglers within jobs from non-stragglers for further optimising their level of resource provisioning.

Resource level optimisation: Based on the estimated resource levels and straggler classification, optimum level of resource provision for every individual task within a given job are determined by considering various runtime and execution factors.

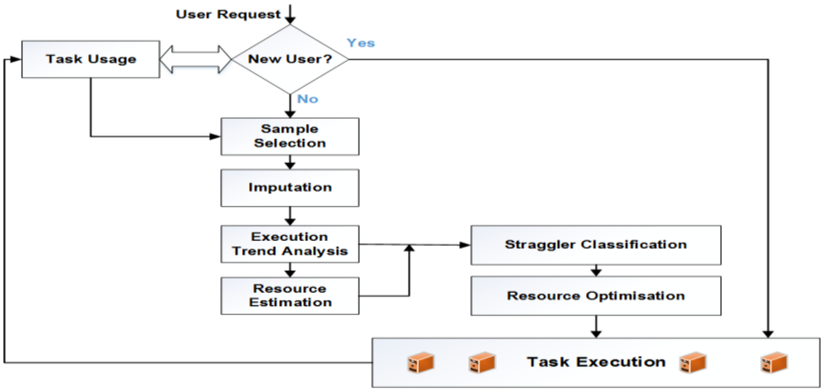


Fig. 1. Analytics Architecture

TABLE 1 PROFILE INFORMATION TABLE FOR JOB 0

Profile Composite / Similarity	Day-of-the-Week Sample			Time-of-the-Day Sample		
	Day 3 Week 1 Wednesday 12-1 am			Day 9 Week 2 Tuesday 12-1 am		
Execution	1	2	3	1	2	3
Job ID n_j	6275968532	6275636968	6275804419	6314856996	6314956139	6315082527
Total number of tasks n_t	50 0	49 -1	48 -2	48 -2	48 -2	48 -2
Job Scheduling p_j	0 1	0 1	0 1	0 1	0 1	0 1
Task Priority p_t	4 1	4 1	4 1	4 1	4 1	4 1
CPU Requirements r_c	0.03125 1	0.03125 1	0.03125 1	0.03125 1	0.03125 1	0.03125 1
Memory Requirements r_m	0.007767 1	0.007767 1	0.007767 1	0.007767 1	0.007767 1	0.007767 1
Task Termination	Evict -1	Kill -2	Kill -2	Kill -2	Evict -1	Kill -2
Similarity Score	3	1	0	0	1	0

RESOURCE ESTIMATION MODULE

4.1 Sample Selection

Firstly for a currently arrived job, similar execution profiles from historical instances are chosen for descriptive analytics by exploiting the inherent periodicity such as the time-of-the-day and day-of-the-week effects among the user behaviours, detailed information about this sample selection can be found in our earlier work [24, 25]. It is always recommended to choose the complete execution profile which can only be obtained from finished jobs, however, it is possible that jobs facing terminations may or may not be resubmitted again. In this event, the historical sample selection might return more than one execution instances. Thus, it is essential to validate the most suitable historical sample for a given job profile. A similarity weight is computed for the chosen historical samples by measuring the quantitative association of the statistical properties between the current and historical samples in terms of the explicitly known number of tasks encompassed within the jobs, resource requests, job scheduling priorities and task priority levels and the termination pattern available from the historical execution traces. A Profile Information (PI) table for similarity measure is constructed based such statistical properties. The execution instances exhibiting a close quantitative association with the currently arrived job profile are validated by the PI table and are then utilised for further predictive analytics for respective jobs.

The construction of the PI table, for a job named Job 0, arrived during 12 - 1 am on Day 10 Wednesday of Week 2, is illustrated in Table 1. The statistical composite for the currently arrived job is extracted as $J_0 = \{12.15\, am, 6323881198, 50, \ 0\}$, implying that Job 0 has been submitted at 12.15 am and has been assigned with a job index of 6323881198, encompasses a total of 50 tasks and has a scheduling priority of 0 (low latency sensitivity). The task profile, named T_0, for the job J_0 has been extracted as a statistical composite $T_0 = \{t_{st,}$ 6323881198, (0.03125, 0.007767), 4} implying that task T_0 has arrived at a time t_{st}, belongs to the Job 6323881198 and characterise a CPU request of 0.03125 cores and a memory request of 0.007767 bytes, and includes a task priority level of 4. Task profile is constructed for all the encompassed tasks within a given job. The sample selection component has returned three similar job profiles from the historical traces for both the time-of-the-day and day-of-the-week samples. The PI table assigns a similarity weight for profile composite, identically associated metrics are assigned a weight of 1 and the metrics deviating from the current job profile are assigned with corresponding deviation for similarity measure, for instance if the quantitative association of the historical job is less than one than the current job, then the corresponding similarity weight will be -1. This computation also includes the measure of task termination proportions in the historical execution instances and assigns a weight of 1 for instances without any task terminations and -1 for evicts, -2 for kill and fail events accordingly in the historical execution traces. Kill [26, 27] is a user triggered termination of the executing tasks, usually it does not involve the intervention of the service providers, whereas fail is a natural event due to several runtime causes such as resource scarcity and resource level breaches. Evict is a provider triggered event, where the execution of the running instances are paused temporarily to accommodate jobs with more priority, and later resumed when resources become available. A similarity score is generated as the summation of all the individual parametric score for all the identified execution instances. Based on the association measure, all the six execution instances in Table 1 can be validated as different execution instances of the same job profile, but the similarity score helps to choose the execution instance with minimal measurable deviation from the currently arrived job profile. Based on the similarity score, instance 1 of the day-of-the-week sample and instance 2 of the time-of-the-day sample are validated as the representatives of the two periodical effects respectively.

4.2 Imputation

Imputation is a process of replacing or substituting missing data in statistical analysis, which is required for execution samples due to the higher probability of execution profiles suffering data ambiguity, missing data and possible anomalies. Missing values in incomplete profiles are estimated using Maximum Likelihood Estimation (MLE), as shown in equation 1. MLE assumes X_i to be normally distributed around a constant mean and variance for a random sample $X_1, X_2, \dots, X_n$. for estimating the value of missing X_i. MLE substitutes or estimates the missing value with the predicted value that maximises the probability of likelihood and minimises imputation error.

$$L(\theta) = P(X1 = x1, X2 = x2, \dots, Xn = xn) = f(x1;\theta) \cdot f(x2;\theta) \cdots f(xn;\theta) = \prod i = 1nf(xi;\theta) \qquad (1)$$

The first equality is the definition of the joint probability mass function and the second equality comes with the consideration that the sample is a random function, implying Xi is independent. The last equality uses the shorthand mathematical notation of the indexed sample values. Expectation-Maximisation (EM) algorithm is used to estimate the maximum likelihood. EM converges in n number of iteration, and estimates the unknown value through n number of equalities.

4.3 Resource Estimation

The estimation module exploits the validated historical samples as the baseline for usage estimation in terms of the total CPU consumption, anticipated duration and CPU usage rate for every individual task within a given job. The CPU usage rate of the tasks and their execution duration of the historical samples are used as input training sets for estimating the anticipated equivalents during the current execution. A dynamic weighing causal moving average (DWCMA) filter is proposed to estimate the CPU usage trend and duration of tasks. Traditional causal moving average filter assumes that a given output sample depends only on the corresponding inputs occurred earlier and usually assigns more weights to the most recent samples, as shown in equation 2.

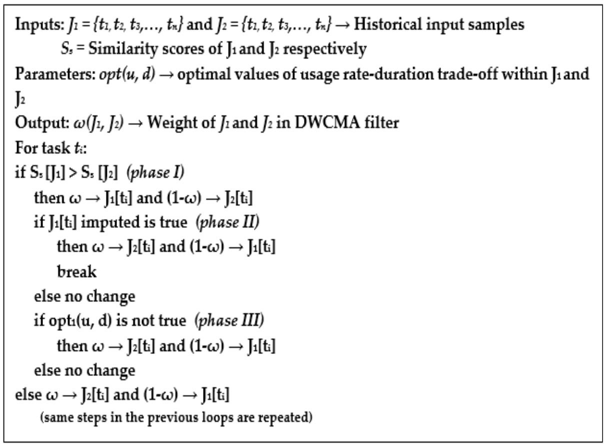
$$y(n) = b(1) * x(n) + b(2) * x(n-1) + \dots + b(Nb+1) * x(n-Nb) \qquad (2)$$

where $y(n)$ is the output response depending on the previous occurrences based on $x(n)$, $x(n-1)$ etc., and $b(1)$, $b(2) \dots b(n)$ are the exponentially assigned weight functions to past occurrences. It is obvious that such a filter is linear and shift-invariant such that $y(n)$ is the output response to $x(n)$, and $y(n–k)$ is the response of the system to $x(n–k)$.

Definition. Linear shift-invariant. For input sets of variables $X = \{x_1, x_2, x_3, \dots x_n\}$ and $Y = \{y_1, y_3, y_3, \dots y_n\}$, the response variables within the output $Z = \{z_1, z_2, z_3, \dots z_n\}$ are internally independent such that the response z_i depends only on its corresponding past instances x_i and y_i.

Hypothesis: Though the task behaviours are dynamic within a given job in such a way that a given job might include energy-aware stragglers and tasks may not satisfy the usage rate-duration trade-off whereby the usage rate and duration are inversely proportional, it is initially assumed that the tasks within a given job will behave normally as non-stragglers and will satisfy the usage rate duration trade-off, in such a way that tasks characterising a lower CPU usage rate runs longer and vice versa.

The proposed resource estimation module adopts the above hypothesis for initially estimating the resource consumption levels of tasks within jobs, however this hypothesis may not necessarily be always true for job executions. Tasks failing to meet the hypothesis are moderated accordingly described as follows. Now from the two sets of historical inputs, the sample set exhibiting better association with the currently arrived job profile is naturally assigned more weights by the DWCMA filter. But this initially assigned weight is dynamically swapped for every individual task execution profile depending on several runtime factors. Unlike the traditional casual moving av-



```
Inputs: $J_1 = \{t_1, t_2, t_3, \dots, t_n\}$ and $J_2 = \{t_1, t_2, t_3, \dots, t_n\}$ → Historical input samples
        $S_s$ = Similarity scores of $J_1$ and $J_2$ respectively
Parameters: $opt(u, d)$ → optimal values of usage rate-duration trade-off within $J_1$ and $J_2$
Output: $\omega(J_1, J_2)$ → Weight of $J_1$ and $J_2$ in DWCMA filter
For task $t_i$:
if $S_s[J_1] > S_s[J_2]$  (phase I)
  then $\omega \rightarrow J_1[t_i]$ and $(1-\omega) \rightarrow J_2[t_i]$
  if $J_1[t_i]$ imputed is true  (phase II)
    then $\omega \rightarrow J_2[t_i]$ and $(1-\omega) \rightarrow J_1[t_i]$
    break
  else no change
  if $opt_1(u, d)$ is not true  (phase III)
    then $\omega \rightarrow J_2[t_i]$ and $(1-\omega) \rightarrow J_1[t_i]$
  else no change
else $\omega \rightarrow J_2[t_i]$ and $(1-\omega) \rightarrow J_1[t_i]$
   (same steps in the previous loops are repeated)
```

Fig. 2. Dynamic Weight Assignment Protocol

erage filter, the proposed DWCMA filter assigns more weights to the most appropriate or (in-trend) sample for a given task execution from the two sets of historical profiles. The term in-trend refers to the satisfactory level of a task execution profile being a non-straggler and satisfying the usage rate-duration trade-off. As discussed earlier, usage rate and duration are inversely proportional to each other for a healthily executed task. Based on the actual usage behaviours of tasks within a given job, an optimal value for usage rate-duration trade-off $opt(u, d)$ for tasks within a job J is defined as in equation 3. The healthily execution trend is unique for every job execution, which is calculated from the execution profile of jobs.

$$opt(u, d) = \underbrace{mean}_{\forall t_i \in J} \{u_i, d_i\} \tag{3}$$

It is not a practical reality for every individual task execution within a given job to exhibit the expected level of healthily execution trend. Hence measurable deviations are always evident among the individual task execution within a job, however the proportional relationship in this trade-off is not obvious. In addition to the execution trend, the task profiles extracted from the actual execution are treated with more weights than those imputed. The protocol for assigning weights to the two samples by DWCMA is presented in Fig. 2. The weight assignment is executed in three cascaded phases, Phase I verifies the Similarity Scores assigned by the PI table, Phase II verifies the correctness of the samples depending on the availability of the actual execution profile and Phase III verifies the usage rate-duration trade-off for every task execution. The two samples sets are assigned with weights depending on the execution profile satisfying the three phases, task profiles within the respective two samples satisfying Phase I and Phase III are assigned with increasing weights and the weights are decreased with a degradation function when Phase II is violated. This is because the EM algorithm often imputes the missing values with an overestimated value. The weights are assigned to the sample sets based on equation 4, where n is the total number of sample sets.

$$\omega = \begin{cases} \frac{1}{n}, & phase\ II\ violated \\ \frac{2}{n+1}, & otherwise \end{cases} \tag{4}$$

The CPU usage rate and duration and the total CPU consumption for every individual task within a given job are estimated by the dynamic weighing CMA filter as a tuple shown in equation 5. Whilst the total CPU consumption provides inferences for optimum level of resource provision, the estimated usage rate and duration are expected to provide inferences for straggler classification, dealt in the following section.

$$P_{out}[i] = \{u_i, d_i, c_i\} \tag{5}$$

where, $P_{out}[i]$ is the estimated tuple for task i, encompassing its estimated values of CPU usage rate u_i, duration d_i and total core consumption c_i. The total CPU consumption reflects the total amount of cores consumed during the entire period of the task execution, whereas the CPU usage rate reflects the mean of the CPU usage rate witnessed at any time throughout the period of the task execution.

5 Straggler Classification Module

After the resource estimation of tasks encompassed within a given job, it is vital to classify the tasks within a single job based on their resource intensiveness.

Definition: *Energy aware stragglers*. For a job set $J = \{t_1, t_2, t_3, \ldots\ldots\ t_n\}$, where n is the total number of tasks within job J, with an average job CPU rate of μ and an average task length of ω, then tasks characterising both a CPU usage rate higher than μ and running longer than ω are termed as energy-aware stragglers within job J.

From the analysis of the selected historical job profiles, two initial lists of energy-aware stragglers are initially generated as S_{t1} and S_{t2}, respectively in the day-of-the-week and time-of-the-day samples. Tasks commonly witnessed as energy-aware stragglers in the two generated lists can anticipated to be a definite straggler during the actual job execution, as shown in equation 6, S_{th} is the initial list of energy-aware stragglers anticipated during the actual job execution. It is also a possibility that S_{th} can be an empty set at this point if none of the straggling tasks overlap in the generated lists of historical stragglers.

$$S_{th}[i] = S_{t1}[i] \cap S_{t2}[i] \tag{6}$$

5.1 Straggler Prediction

This section presents the proposed analytics methodology for straggler classification before the initialisation of the job execution. Based on the initial lists of stragglers in the two historical samples, an n^{th} percentile distribution of energy-aware straggles within the two historical samples is extracted. Now, tasks not impacted by the abrupt behaviours of CPU usage rate and duration are isolated and categorised as non-stragglers based on the observations falling beyond the *(100-n)*th distribution, using equation 7.

$$N_{st} = W_{(1,2)}[i] \begin{cases} i_l < P_{(100-n)} \\ i_u < P_{(100-n)} \end{cases} \tag{7}$$

where, N_{st} is the sample containing non-stragglers, $W[i]$ is the chosen historical sample, i_l is the task duration, i_u is the mean CPU usage rate of i^{th} task respectively, $P_{(100-n)}$ is the *(100-n)*th percentile value and n is the proportions of stragglers identified in the historical sample. After filtering out the energy-aware stragglers, threshold score for the CPU usage rate and duration for non-stragglers is obtained for the two samples using equation 8.

$$N_{s_{(\alpha_{(1,\ 2)},\ \beta_{(1,\ 2)})}} = (\textstyle\sum_{i=1}^{n} \frac{N_{st}[u_i]}{n}, \sum_{i=1}^{n} \frac{N_{st}[l_i]}{n}) \tag{8}$$

where, n is the total number of non-stragglers, α and β are the average values of the CPU usage rate and task duration of the non-stragglers within the two samples. The non-straggler threshold values are computed separately for the two samples as $N_{s1(\alpha,\ \beta)}$ and $N_{s2(\alpha,\ \beta)}$. These two values form the upper and lower confidence limits for the average duration and CPU usage rate for the non-straggling tasks during the current execution. Now, this confidence limits are applied to the predicted output obtained in equation 5 to generate the list of non-stragglers bounded with a two-sided confidence limit for the target job respectively, as shown in equation 9 and 10.

$$Pu_{con}[i] = \alpha_1 < P_{out}[i] < \alpha_2, \quad for\ CPU\ usage\ rate \tag{9}$$

$$Pl_{con}[i] = \beta_2 < P_{out}[i] < \beta_1, \quad for\ task\ duration \tag{10}$$

where, $Pu_{con}[i]$ and P_{lcon} $[i]$ are the list of tasks satisfying the confidence bounds of CPU usage rate and duration thresholds for non-stragglers respectively in the predicted usage profile. Tasks anticipated to execute within the limits of P_{con} should satisfy the non-straggler criterion. However, the trade-off between task duration and the CPU usage rate is crucial in deciding the energy-aware straggling behaviours of tasks. Furthermore, tasks anticipated to exhibit a duration less than the value projected by the confidence bounds are considered as not satisfying the non-straggler criterion, with the presumption that lower duration might characterise a higher usage rate, thereby not satisfying the usage rate-duration trade-off. Anticipated task execution behaviours not falling within the healthily execution criterion is vulnerable to become potential stragglers during execution. An optimised CPU usage rate and duration for the non-stragglers during the actual execution is given by equation 11.

$$S^{th}{}_{(\alpha,\ \beta)} = \left(\frac{\sum_{i=1}^{n} P_{ucon}[i]}{n_u}, \frac{\sum_{i=1}^{n} P_{lcon}[i]}{n_l}\right) \tag{11}$$

where n_u and n_l are the total number of tasks in P_{ucon} and P_{lcon} respectively. The predicted output obtained from equation 5 is now subjected to this trade-off criterion and the tasks not meeting $S^{th}{}_{(\alpha,\ \beta)}$ are further isolated and labelled as anticipated energy-aware stragglers during execution. $S^{th}{}_{(\alpha,\ \beta)}$ is expected to present the CPU usage rate and duration threshold values for both the usage and duration confidence limit samples. Ideally, α_1 and α_2 are the upper and lower thresholds for the CPU usage rate for non-stragglers in the predicted output, where α_1 is obtained from the P_{ucon} and α_2 is obtained from P_{lcon} respectively, and β_1 and β_2 are the duration counterparts, where β_1 is obtained from P_{lcon} and β_2 is obtained from P_{ucon} respectively, based on equation 11. The thresholds for non-stragglers anticipated in the predicted output is computed based on equation 12 and equation 13, which weighs α_1 and β_1 more than their counterparts α_2 and β_2.

$$S^{th}{}_{\alpha} = (\omega * \alpha_1) + (1 - \omega)\alpha_2 \tag{12}$$

$$S^{th}{}_{\beta} = (\omega * \beta_1) + (1 - \omega)\beta_2 \tag{13}$$

Now, an initial classification of the energy-aware stragglers anticipated during the execution of the target job is achieved based on equation 14.

$$S_{t-off} = \{P_{out}[i]\{(i_u > \alpha) \cap (i_l > \beta)\} \cup (S_{th}[i]) \tag{14}$$

where i_u is the CPU usage rate and i_l is the task duration of i^{th} task contained in P_{out}, α is the optimised CPU usage rate and β is the optimised task duration respectively given by equation 12 and equation 13, and S_{th} is the initial list of stragglers given by equation 6.

S_{t-off} is the probability of stragglers identified from the descriptive analytics of the historical events and predicted resource usage profiles. However, the actual task behaviour depends on several run-time factors such as the node efficiency, resource consumption fluctuation, running duration, task intensity etc., and are impacted by the sole effects of CPU usage rate and task duration respectively. Thus, it is important to further optimise this classified straggler list S_{t-off} through a categorical analysis of straggler probability by incorporating the behavioural heterogeneity for every individual task. Hence, this initial classification of tasks is used as a hypothesis for task behaviours during actual execution, which is further subjected to a Naïve Bayes classifier for enhancing the preciseness of the dependability of S_{t-off}. A bayes rule scales well for a categorical classification when the dimensionality of the inputs is high. Now, P_{out} delivered by equation 5 with an initial task classification based on S_{t-off} is trained as input data for Naïve Bayes classifier to obtain the final list of energy-aware stragglers anticipated in P_{out} during the actual job execution.

Definition. Conditional independence. For a set of predictor $X = \{x_1, x_2, x_3, \ldots.x_k\}$, Naive Bayes classifier assumes that the effect of the value of a predictor x_i on a given class c is independent of the values of other predictors.

Conditional independence is the property of the involved predictors having their independent influence upon the prediction output irrespective of the presence and influence of the other predictors. Based on this conditional independence, the influence of CPU usage rate, task duration and total core consumption from the two historical samples and the predicted output are evaluated individually on the task classification given by S_{t-off}, using equation 15. This is due to the fact that the total CPU consumption for a given task can be determined by the product of the task's runtime duration and the mean CPU usage. Furthermore, the CPU usage rate of a given task determines its resource intensity.

$$P(c/x) = \frac{P(x/c)P(c)}{P(x)} \tag{15}$$

The overall influence of all the predictors on the resource intensity of a given task for straggler classification is given by equation 16.

$$P(C/X) = P(x_1/c)^{*} P\left(\frac{x_2}{c}\right) * P\left(\frac{x_3}{c}\right) * \ldots\ldots.{*}\, P\left(\frac{x_n}{c}\right) = \prod_{k=1}^{n} P(x_k/c) \tag{16}$$

where, $P(c/x)$ is the posterior probability of *class c* for a given predictor x, $P(c)$ is the prior probability of *class c* (straggler or a non-straggler), $P(x/c)$ is the likelihood for the probability of class c for a given predictor x, and $P(x)$ is the prior probability of predictor x. After evaluating the posterior probability, Naïve Bayes classifier categorises a given observation as belonging to the class of stragglers C_i or non-stragglers C_j using equation 17.

$$C(obs) = \begin{cases} C_i, & for\ P(C_i/X) > P(C_j/X) \\ C_j, & otherwise \end{cases} \tag{17}$$

Since the predictors including CPU usage rate, duration and total CPU consumption are numerical values, all such values should be discretised into respective categories before Naïve classifier estimates the posterior probability. For discretisation, the classifier assumes a normal distribution for observations within a given job based on the measure of mean and standard deviation functions.

5.2 Runtime Mitigation

Stragglers in the past do not necessarily behave as a future straggler due to the runtime heterogeneity, simply classifying the energy-aware stragglers based on their historical behaviour may not be sufficient for Cloud executions. A task execution profile is consistent only when it exhibits statistical correlations among different execution instances. $S^{th}{}_{(\alpha,\ \beta)}$ obtained in the offline analytics can be referred as the threshold point to label a running tasks as potential straggler when the task execution breaches the threshold defined by $S^{th}{}_{(\alpha,\ \beta)}$. Due to the uncertainty in the task execution behaviour and the usage rate-duration trade-off can only be achieved after the execution, the proposed scheme alerts a task as possible energy-aware straggler when the CPU usage rate breaches its corresponding threshold. Long tail stragglers can be identified based on the duration threshold, though evaluating the duration threshold for long-tail straggler identification is not within the scope of this paper. The threshold for CPU usage rate and duration is calculated by subjecting the corresponding threshold values of the non-stragglers obtained in the chosen historical sample analytics, the offline straggler threshold and the threshold of non-stragglers in the prediction output to a two-tier nested exponential smoothing filter, as shown in equation 18 and equation 19, in such a way that the factual values enjoy a better weighing than the anticipated values.

$$S_{ru} = \{\omega(\omega\,\alpha_1 + (1 - \omega)\,\alpha_2) + (1 - \omega)\,(\omega\,\alpha_{off} + (1 - \omega)\,\alpha_{pred})\} \quad (18)$$

$$S_{rl} = \{\omega(\omega\,\beta_1 + (1 - \omega)\,\beta_2) + (1 - \omega)\,(\omega\,\beta_{off} + (1 - \omega)\,\beta_{pred})\} \quad (19)$$

where, $\omega = 2\,/(n{+}1)$, α_1 and α_1 are the mean CPU usage rate of the non-stragglers identified in the historical sample analytics, α_{off} is the mean CPU usage threshold used in the offline straggler identification based on $S^{th}{}_{(\alpha,\ \beta)}$ and α_{pred} is the mean CPU usage threshold of the non-stragglers in the predicted output (based on P_{con}), and β is the duration equivalent respectively.

Since Cloud workloads are dynamic in nature, the heterogeneity among the Cloud workloads can be witnessed from two different perspectives: firstly, tasks within a job characterising increased CPU usage rate fluctuation with fairly even distribution of task length, and secondly tasks within a job charactering increased CPU usage rate with uneven distribution of task length. Whilst the former does not have an impact on the overall completion time of the job, the latter can significantly impact the overall job completion time. In other words, the former is a job containing only energy-aware stragglers and the latter consists of both energy-aware stragglers and long tails. In general, long tails depend on the runtime factors such as co-located tasks, node efficiency, node-level stragglers etc., thus it is optimum to mitigate the long tail stragglers during runtime rather than attempting to predict them before execution. The characteristics of long tails are postulated to exhibit an execution duration of 50% greater than the duration threshold S_{rl}, as shown in equation 20.

$$S_{lt} = t_l\,[i] > 1.5 * S_{rl} \quad (20)$$

Thus, during the actual job execution, the runtime stragglers are identified using equation 21.

$$S_t = \begin{cases} t_u[i] > S_{ru}\,, & for\ energy - aware\ stragglers \\ t_l[i] > S_{lt}, & for\ long\ tail\ stragglers \end{cases} \quad (21)$$

where $t_u[i]$ and $t_l[i]$ are the current CPU usage rate and task duration of the i^{th} task within a given job during execution. The initial task classification facilitates deciding their resource provisioning levels before the actual execution. But the energy-aware stragglers identified during execution needs dynamic scaling of resources which can be achieved through dynamic vertical scaling, however this is considered to be out of scope of this paper.

6 Resource Provision Module

The resource provisioning optimisation problem can be witnessed from two perspectives: firstly, commitment of resources levels should not be vulnerable to cause resource idleness, and secondly the task execution should not breach the provisioned level of resources. From the resource prediction perspectives, the former usually results from the over-estimated resource levels and the latter results from under-estimation. This section is focused on optimising the under-estimated resource levels delivered by the proposed resource estimation module in consideration of the straggler classification. A resource usage and straggler-aware over commitment policy of resource levels has been proposed to meet the objectives of termination less execution with minimal resource idleness.

6.1 Task Categories

Based on the early discussed behaviours and characteristics of tasks within jobs, tasks are classified as non-stragglers and energy-aware stragglers for resource provision. A static resource provisioning policy for these two categories may not scale well for energy efficiency, since the resource consumption level of non-stragglers and stragglers are significantly different.

Case (1): Non-stragglers

Non-stragglers are expected to execute without any notable abnormal resource consumption level, thus it is recommended to initially rely on the resource usage estimation whilst provisioning resource levels to non-stragglers. Thus, the anticipated usage of non-stragglers is expected as per the usage prediction output, as in shown equation 22, where $R_{Pred}[i]$ is the initially predicted resource usage level for task i within a given job given by equation 5.

$$R_{ns(c)}[i] = R_{Pred}[i] \quad (22)$$

Case (2): Energy-aware Stragglers

The resource consumption level for energy-aware straggles are usually expected to exceed the level of non-stragglers by a significant margin. Further to the usage prediction of stragglers, it is also recommended to rely on the historical execution instances to determine the resource provisioning level of the classified energy-aware stragglers. Thus, the resource consumption level of stragglers $R_{st(c\,)}$ is anticipated in accordance with the identified maximum CPU consumption levels among the two historical samples and the resource prediction as shown in equation 23, where $R_{s1}[i]$ and $R_{s2}[i]$ are the corresponding CPU consumption level of a given straggling task i in the two historical execution profiles respectively.

$$R_{st(c)}[i] = \max(R_{s1}[i], R_{s2}[i], R_{Pred}[i]) \quad (23)$$

6.2 Over Commitment factor

The extravagant heterogeneity of tasks within a job and the dynamic datacentre runtime environment is naturally insisting the need for overcommitting the resource levels for achieving termination-less execution. To this end, a dynamic over-allocation policy has been further proposed for every individual task within a given job to optimise the resource provisioning levels. An over commitment of factor γ is adopted for all the task categories, whereby it is proposed to overcommit the resources by a margin of γ to the level insisted by equation 22 and equation 23 accordingly. This over commitment factor is dynamically evaluated for every individual task depending on three important factors: task consistency based on the process efficiency of tasks in the historical execution profiles, process capacities of tasks determined for the actual execution based on the resource prediction output, and task classification delivered by the straggler classification module. Three classes of reliability have been adopted for every individual factor determining the over commitment factor as high, medium and low, with high insisting a highly reliable measure through to low being less reliable for a task to behave normally during execution. Over allocation proportions for the three defined levels of reliability classes are adopted as 1.3, 1.4 and 1.5 respectively for high, medium and low classes of task reliability. Based on these over-commitment factors and the respective reliability class of tasks, every individual task will be dynamically evaluated to moderate the over commitment factor.

6.2.1 Process Efficiency

The reliability of resource consumption consistency of tasks is estimated based on the process efficiency of the task execution during its historical instances. Despite the allocated level of resources, it is usual for the process efficiency of certain tasks to fall behind than a majority of tasks within the same job, impacted by the task nature.

Definition: Process Efficiency. Given a Job set J with n number of tasks $J=\{T_1, T_2, T_3,......, T_n\}$, executed for a duration $D=\{t_1, t_2, t_3,...,t_n\}$ consumed a resource level of $R=\{r_1, r_2, r_3,....r_n\}$, where the Task T_i is executed for a duration of t_i, and consumed r_i amounts of resources, the task process efficiency Pe_i of task T_i is defined as the ratio of the executed duration to the amount of resources consumed, where the job duration and resource consumption of the job are given by $max(t_i)$ and $\sum_{i=o}^{n} r_i$ respectively. The task process efficiency and the job process efficiency can be computed as shown in equation 24 and equation 25.

$$Pe_i = \frac{t_i}{r_i} \qquad (24)$$

$$Pe_j = \frac{\sum_{i=0}^{n} Pe_i}{n} \qquad (25)$$

In general, the task process efficiency is usually the measure of an individual task, and an average of all the task efficiency encompassed within a job reflects the efficiency of the entire job. If certain tasks are allocated with less process efficiency within a single job, such tasks are vulnerable to behave either as long tails or energy-aware stragglers resulting from the node-level process efficiency. It is postulated that the task efficiency of a given task is extremely low if its process efficiency falls below 50% of the job process efficiency as shown in equation 26.

$$T_{le}[i] = \begin{cases} J[t_i] & if\ P_{ei}[i] < 0.5 * Pe_j\ is\ true \\ nil & otherwise \end{cases} \qquad (26)$$

Now, based on the task process efficiency of the two historical samples, the consistency of a given task is measured among the two historical usage profiles to evaluate its class of reliability. A task is considered to be highly reliable when the process efficiency of the corresponding task stays internally consistent among the two historical usage profiles. If the task process efficiency is not consistent in both the two historical samples, it is less reliable and if the task process efficiency is consistent only in one of the two historical samples it characterises a reliability class of medium.

6.2.2 Process Capacity

It is worthy of note that the task efficiency is not only affected by the node-level stragglers, tasks those are more resource intensive within a single job naturally demands more resources than the other co-located tasks.

Definition: Process capacity. Given a Job set J with n number of tasks $J=\{T_1, T_2, T_3,......, T_n\}$, predicted to run for a duration $D=\{t_1, t_2, t_3,...,t_n\}$ anticipated to consume a resource level of $R=\{r_1, r_2, r_3,....r_4\}$, where the Task T_i is expected to run for a duration of t_{pi}, and to consume r_{pi} amounts of resources, the task process capacity Pc_i of task T_i is defined as the ratio of the anticipated duration to the predicted level of resource usage, where the anticipated job duration and resource consumption of the job are given by $max(t_i)$ and $\sum_{i=o}^{n} r_i$ respectively. The task process capacity and the job process capacity can be computed as shown in equation 27 and equation 28.

$$Pc_i = \frac{t_{pi}}{r_{pi}} \qquad (27)$$

$$P_{cj} = \frac{\sum_{i=0}^{n} P_{ci}}{n} \qquad (28)$$

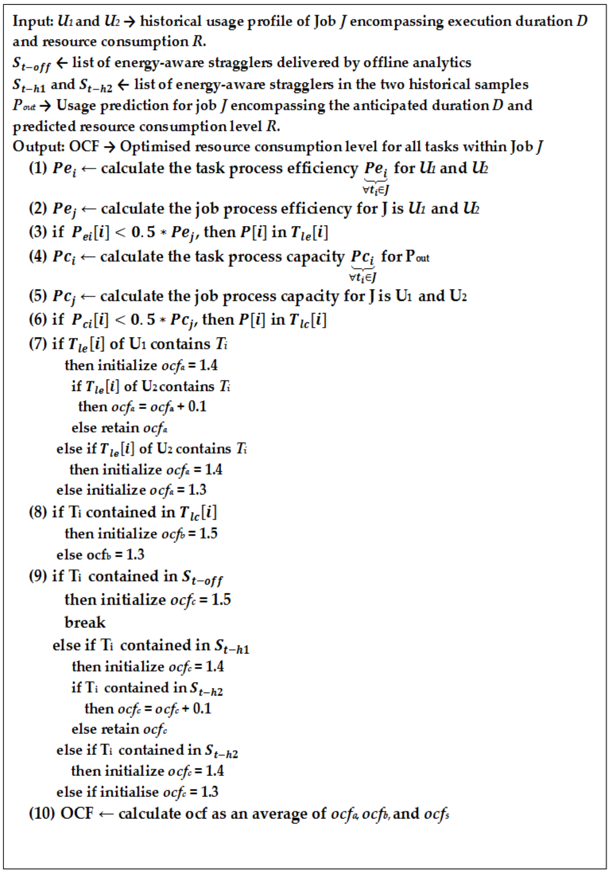
Process capacity will determine the effectiveness of the provisioned level of resources in processing the job within the determined time-scale. Higher the value of Pc_i, greater is the process efficiency and shorter is the execution duration. Lower the value of Pc_i, higher is the possibility of the corresponding task to behave as a straggler. Similar to the task process efficiency, tasks with low process capacity $T_{lc}[i]$ are identified using equation 29.

$$T_{lc}[i] = \begin{cases} J[t_i] & if\ P_{ci}[i] < 0.5 * Pc_j\ is\ true \\ nil & otherwise \end{cases} \qquad (29)$$

Tasks with lower process capacity identified by equation 29 are considered as less reliable whilst optimising their initially estimated resource provisioning level.

6.2.3 Task Category

Furthermore, the over-commitment factor will give special emphasis to the straggler classification with the objective of overcommitting the resource levels of energy-aware stragglers. For a given task, if it behaved as non-straggler in the two historical usage profiles and further classified to be a non-straggler based on the resource prediction, it is highly reliable to stay as a non-straggler during execution. For a given task, if it is classified as energy-aware straggler in the offline analytics, then it is less reliable to behave as a non-straggler during execution. For a given task, if it has behaved as energy-aware straggler



```
Input: $U_1$ and $U_2$ → historical usage profile of Job $J$ encompassing execution duration $D$
and resource consumption $R$.
$S_{t-off}$ ← list of energy-aware stragglers delivered by offline analytics
$S_{t-h1}$ and $S_{t-h2}$ ← list of energy-aware stragglers in the two historical samples
$P_{out}$ → Usage prediction for job $J$ encompassing the anticipated duration $D$ and
predicted resource consumption level $R$.
Output: OCF → Optimised resource consumption level for all tasks within Job $J$
  (1) $Pe_i \leftarrow$ calculate the task process efficiency $\underbrace{Pe_i}_{\forall t_i \in J}$ for $U_1$ and $U_2$
  (2) $Pe_j \leftarrow$ calculate the job process efficiency for J is $U_1$ and $U_2$
  (3) if $P_{ei}[i] < 0.5 * Pe_j$, then $P[i]$ in $T_{le}[i]$
  (4) $Pc_i \leftarrow$ calculate the task process capacity $\underbrace{Pc_i}_{\forall t_i \in J}$ for $P_{out}$
  (5) $Pc_j \leftarrow$ calculate the job process capacity for J is $U_1$ and $U_2$
  (6) if $P_{ci}[i] < 0.5 * Pc_j$, then $P[i]$ in $T_{lc}[i]$
  (7) if $T_{le}[i]$ of $U_1$ contains $T_i$
          then initialize $ocf_a = 1.4$
           if $T_{le}[i]$ of $U_2$ contains $T_i$
             then $ocf_a = ocf_a + 0.1$
           else retain $ocf_a$
        else if $T_{le}[i]$ of $U_2$ contains $T_i$
          then initialize $ocf_a = 1.4$
        else initialize $ocf_a = 1.3$
  (8) if $T_i$ contained in $T_{lc}[i]$
          then initialize $ocf_b = 1.5$
        else $ocf_b = 1.3$
  (9) if $T_i$ contained in $S_{t-off}$
          then initialize $ocf_c = 1.5$
          break
        else if $T_i$ contained in $S_{t-h1}$
           then initialize $ocf_c = 1.4$
           if $T_i$ contained in $S_{t-h2}$
             then $ocf_c = ocf_c + 0.1$
           else retain $ocf_c$
        else if $T_i$ contained in $S_{t-h2}$
           then initialize $ocf_c = 1.4$
        else if initialise $ocf_c = 1.3$
  (10) OCF ← calculate ocf as an average of $ocf_a$, $ocf_b$, and $ocf_s$
```

Fig. 3. Over-Commitment Protocol

during both its historical execution instances and not classified as energy-aware straggler, then it is less reliable to behave as a non-straggler during execution. For a given task, if it has behaved as energy-aware straggler in either one of the two historical samples and not classified as energy-aware straggler, then it characterises a medium reliability to behave as non-straggler during execution.

6.3 Over-commitment Percentage

The above discussed factors determining the over commitment level of resources for task execution are expected to influence each other. For instance, an energy-aware straggler task with process efficiency consistency will enjoy a higher reliability weightage at step 6.2.1 and a lower reliability weightage at step 6.2.3, such that the level of over commitment postulated in the two respective levels are expected to cancel out the effects of each other so as to deliver a final optimised level of over-commitment. Thus, the optimised over-commitment percentage of resources OCF for a given task is achieved as an average of the over commitment level determined by the above three factors as shown in equation 30, where ocf_a, ocf_b, ocf_c are the over commitment level of resources computed based on the three influencing factors respectively. The over-commitment protocol is illustrated in Fig. 3.

$$OCF = \frac{1}{3}\,(ocf_a + ocf_b + ocf_c) \tag{30}$$

The final level of recommended resource provision for a task i is given by equation 31. The OCF is calculated for all the task groups since the optimisation framework uniquely treats every individual tasks and further adopts an over commitment policy for all tasks in an attempt to avoid under provisioning of resources.

$$R_p[i] = \begin{cases} R_{ns(c)}[i] * OCF, & for\ non\ stragglers \\ R_{st(c)}[i] * OCF, & for\ stragglers \end{cases} \tag{31}$$

7 Performance Evaluations

The efficiency of the proposed analytics framework has been evaluated both theoretically and experimentally by training real-world Google Cloud trace logs [28]. In order to display the impacts of job heterogeneity upon the efficiency of the proposed framework , six different jobs have been chosen as representatives of different job types based on the heterogeneous combinations of encompassed tasks within jobs, straggler composition, task duration etc, as summarised in Table 2.

7.1 Resource Estimation Performance

The efficiency of the proposed dynamic weighing CMA algorithm in estimating the resource requirements of the jobs has been evaluated against the existing state-of-the-art techniques [29-34] including Simple Moving Average filter, Exponential Moving Average, Low Pass filter, Auto Regressing Moving Average and Linear Regression.

TABLE 2 Job Profile Representation

Job	No. of Tasks	Periodicity		Straggler Proportions (%)	Job Nature
		Day-of-the-week	Time-of-the-day		
Job 0	50	Yes	Yes	8	Uneven distribution of task length and resource intensity.
Job 1	100	Yes	Yes	26	Even distribution of task length, with only a very few tasks characterise a lower duration but the resource intensity is extremely heterogeneous.
Job 2	200	Yes	Yes	35	Evenly distributed task length and resource intensity, only a few tasks exhibit lower resource intensity.
Job 3	182	Yes	No	13	Limited periodicity due to the lack of the time-of-the-day sample.
Job 4	488	Yes	Yes	40	Extremely heterogeneous distribution of task length and resource intensity.
Job 5	1050	Yes	Yes	2	Fairly homogeneous distribution of both task length and resource intensity.

Fig. 4 presents the estimated CPU requirements against the actually consumed for Job 0, Job 1, Job 2, Job 4 and Job 5 respectively. Since Job 3 does not satisfy the historical window requirements of the proposed methodology, evaluation of resource estimation for Job 3 is not included. The unusual spikes in the actual trend illustrates the increased CPU consumption of the energy-aware stragglers. This exhibits the increased analytics complexity in capturing and predicting the resource requirements of energy-aware stragglers. 3 out of 4 energy-aware stragglers have not been captured by any of the evaluated techniques in the case of Job 0. In the case of Job 1, ARMA presents a linear prediction for all the encompassed tasks, and it is over-predicting the resource requirements of non-stragglers by a considerable margin. This is due to the inefficiency of ARMA model in capturing minute deviations among the observation. In the case of Job 2, all the techniques are vulnerable to under-estimate the resource requirements of energy-aware stragglers and ARMA model delivers a flat prediction. Similar behaviours of energy-aware straggler prediction can be observed for Job 4, where a few of the non-stragglers are over estimated by a significant margin by all the techniques. Interestingly, all the prediction techniques are closely estimating the resource requirements of the tasks encompassed within Job 5. This is because the proportional presence of energy-aware stragglers is insignificant in Job 5, accounting for only around 2%.

Fig. 5 presents the over and under-prediction ratio of all the evaluated techniques, presented as an average of all the studied jobs. Considering all the task classification, the proposed DWCMA over-estimates 44.65% of tasks and under-estimates 55.34% of tasks accordingly. The over estimation of the evaluated techniques is observed at an average of 45.34%, 46.16%, 54.04%, 54.14%, 47.32%, and the under estimation is observed at an average of 54.65%, 53.83%, 45.93%, 45.58% and 53.67% respectively for SMA, EMA, LPF, ARMA and LR. The estimation efficiency of all such techniques are further evaluated by isolating the energy-aware stragglers and long tails, so as to evaluate the resource estimation efficiency for non-stragglers. Now, the proposed DWCMA over estimates 59.06% and under-estimates 40.93% of the non-straggler tasks. Further, the over estimation of the compared techniques is observed at 59.51%, 60.75%, 71.40%, 72.72% and 59.71%, and the under estimation is observed at 40.48%, 39.24%, 28.59%, 27.27% and 40.28% respectively for SMA, EMA, LPF, ARMA and LR. It is clearly evident that the under prediction for non-stragglers is much lesser than those of the total tasks (including energy-aware stragglers, non-stragglers and long tails), illustrating the impacts of energy-aware stragglers in prediction analytics.

The estimation trade-off of the proposed DWCMA, SMA, EMA and LR fairly remains the same for both the task classifications. But the over estimation ratio of LR

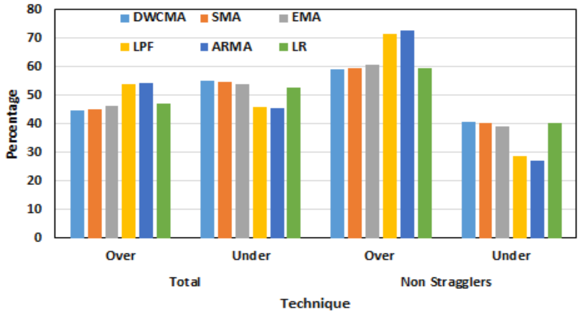


Fig. 5. Over and Under Prediction Ratio

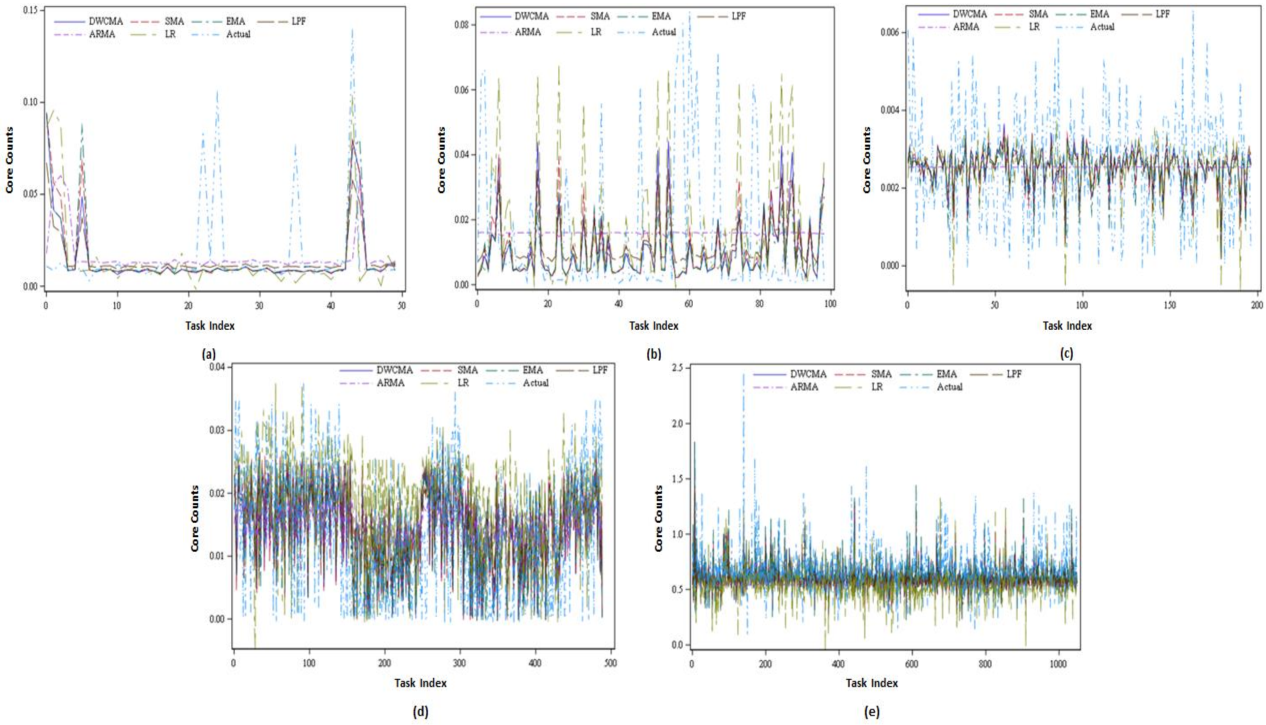


Fig. 4. Resource Estimation Observation (a) Job 0 (b) Job 1 (c) Job 2 (d) Job 4 (e) Job 5

and ARMA are higher than the remaining techniques by a considerable margin. Whilst is optimum for a prediction technique to over-estimate the resource requirements, this over estimation should always be marginally higher than the actual requirements. Over-estimating the resource level by a significant margin leads to resource wastages, which is actually the case of ARMA. In general, ARMA model fits well for time series trend prediction and usually presents the upper and lower confidence limits for prediction, but does not scale well in the context of resource requirement estimation. LPF presents a better over estimation ratio, however LPF depends on the average of prediction outcome of the previous iteration and adds a degradation function for the current sample. This increases the computational complexity by incurring multiple iterations of prediction analytics and the degradation function is vulnerable to over-commit the resource levels.

Furthermore, the resource estimation accuracy of the stated techniques is evaluated in terms of the Average Accumulative Error (AAE) for under-estimated and over-estimated tasks within a single job respectively, presented in Fig. 6 as an average of all the studied jobs. It can commonly be observed that all the evaluated techniques are exhibiting a better AAE Percentage for the under-estimated tasks than those of the over-estimated tasks, though the over-estimation ratio is better for non-stragglers. In other words, a majority of the non-straggling task proportions are over-estimated with high error percentage and a minority of the non-straggling task proportions are under-estimated with minimum error percentage. Due to space constraints, the discussion of individual jobs is not presented in this section. The AAE for under-estimated tasks are observed at 28.59%, 30.9%, 30.04%, 32.01%, 19.63% and 36.72% for DWCMA, SMA, EMA, LPF, ARMA and LR respectively for all the job categories. Similarly, the AAE for the over-estimated tasks are observed at 45.24%, 49.69%, 49.14%, 45.19, 67.98% and 56.24% respectively for DWCMA, SMA, EMA, LPF, ARMA and LR for all job categories.

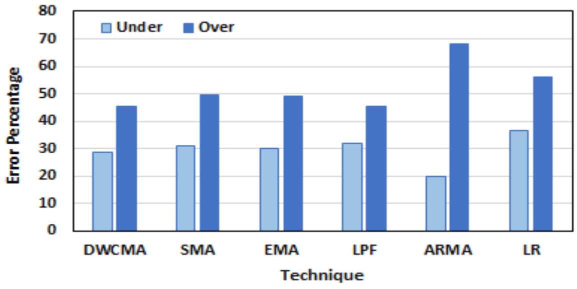


Fig. 6. Average Accumulative Error for Resource Prediction

TABLE 3 REPLICATED CLOUD EXECUTION STATISTICS FOR EXPERIMENTS

Job Name	Task Index	Cloud Execution (seconds)	Experiment Duration (seconds)	Imposed Delay (seconds)	Resource Intensity (%)
Job 0	0	156	148.2	5.93	28
Job 1	12	178	177.9	4.1354	43
Job 1	91	161	161.07	2.68	22
Job 2	31	46	45.84	0.88	52
Job 2	191	50	49.80	1.13	44
Job 3	44	40	39.71	0.23	171
Job 4	138	229	228.17	3.93	58
Job 4	420	222	221.47	5.98	37
Job 5	233	1698	1689.97	67.55	25
Job 5	1008	1734	1729.26	86.4	20

Overall, it can be concluded that the proposed DWCMA protocol achieves a better prediction accuracy trade-off between the under and over-estimated non-straggling tasks within the given jobs than the compared techniques with better AAE. Though marginal, the under-estimated tasks are vulnerable for terminations due to under-commitment of resources, thus needs further optimisation.

7.2 Straggler Classification Performance

7.2.1 Experiment Setup and Workload Generation

The efficiency of the proposed straggler classification methodology has been evaluated in two different phases. Firstly, the offline analytics has been evaluated to demonstrate the efficiency of the proposed methodology in classifying the task-level energy-aware stragglers before the start of the actual job execution. Secondly, the identification accuracy of the proposed methodology in detecting energy-aware stragglers during the actual execution has been evaluated. To facilitate the evaluation of runtime identification, the proposed framework along with the compared techniques are modelled to identify energy-aware stragglers when the jobs are executed in a Kubuntu VM characterising 2 core processor and 1GB RAM, every task within a given job are executed on individual threads to reflect the isolated LXCs of a typical Cloud datacentre. Tasks within a single job are parallelised and all the threads used for a single job execution are started at the same time to ensure an even start time for all the tasks within a given job. Heterogeneity among the running tasks within a single job in terms of their progress rate is achieved by imposing various level of delays and computation intensities among the tasks, whereby tasks are executed with varied resource consumption and completion times. After an initial random run of the chosen jobs in the evaluation system, the datacentre equivalent (near identical) execution time and proportional resource intensity of the individual tasks within the respective jobs are achieved by moderating the initially imposed delay within the task progress using equation 32.

$$d_{req} = (d_c * t_{req}) \,/\, t_c \tag{32}$$

where, d_{req} is the required delay within the task progress to identically replicate the Cloud execution, d_c is the delay imposed during the initial run, t_{req} is the task execution duration of the Cloud execution and t_c is the task completion time of the initial run respectively. Table 3 presents the replicated task completion time to that of the typical Cloud execution along with the heterogeneity in resource intensity for a randomly chosen 10 tasks from the studied jobs. The resource intensity percentage of the tasks are presented in accordance with the intensity of the other co-located tasks within the same job.

The straggler detection efficiency of the proposed methodology has been evaluated against the existing state-of-the-art threshold calculation [5, 17, 18, 20] methods including static threshold, progress score based threshold, task progress based threshold, estimated finish time based threshold and a normalised duration based threshold accordingly.

7.2.2 Classification Accuracy

This section presents the analysis of the experiment results for the proposed energy-aware straggler classification technique against the compared techniques. Due to space constraints, Job 0 and Job 4 have been chosen to discuss the obtained performance in detail as they represent the two performance extremities impacted by the presence of energy-aware stragglers, and an overall average statistics are further presented.

Fig. 7 presents the classification efficiency of the proposed and the compared techniques in terms of the total number of correctly identified energy-aware stragglers and the true and false positive proportions of classified tasks respectively for Job 0. It can be observed that the proposed offline analytics identifies 3 out of 4 energy-aware stragglers even before the execution starts and the proposed runtime analytics threshold identifies all the energy-aware stragglers. In addition, the true positive rate of the proposed runtime threshold is significantly better than the compared techniques witnessed at 66.66% and further reducing the false positives rate down to 33.33%. Though the static mean threshold identifies all the energy-aware stragglers during runtime, the true positive and false positive rates are witnessed at 36.36% and 63.63% respectively, much worse than the proposed technique. It can further be confirmed that the rest of the evaluated techniques are not efficient in identifying energy-aware stragglers and further characterise significant proportions of false positives, resulting in wrong classification of non-stragglers as energy-aware stragglers. Thus, it is clear that despite the job heterogeneity, the proposed methodology is effective in accurate classification of energy-aware stragglers with minimal proportions of false positives.

Fig. 8 presents the classification accuracy statistics of the evaluated techniques for Job 4. Job 4 is an interesting sample since it comprises more than 40% of energy aware stragglers, which means nearly half of the tasks are energy intensive. In this regard Job 4 itself can be regarded as an energy-intensive job rather than comprising energy-aware stragglers. It is evident that the proposed classification technique outperforms the evaluated techniques in terms of the achieved trade-off between identified stragglers and reduction in false positives, with the proposed technique identifying 161 out of 201 energy-aware stragglers at a true positive rate of 64.9%, against the static mean threshold identifying 171 stragglers at a true positive rate of 60.6%. The total number of correctly identified stragglers by the remaining techniques are very insignificant, and the benefits availed to resource provision is little of merit.

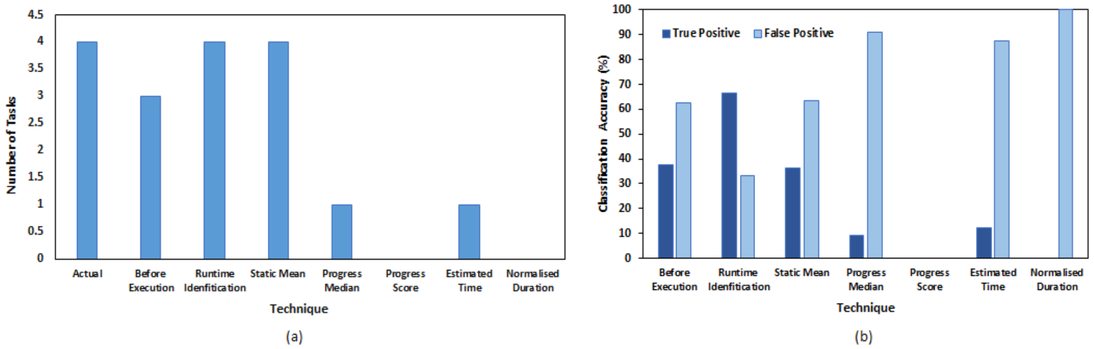


Fig. 7. Classification Accuracy for Job 0 (a) Identified Stragglers (b) Error Proportions

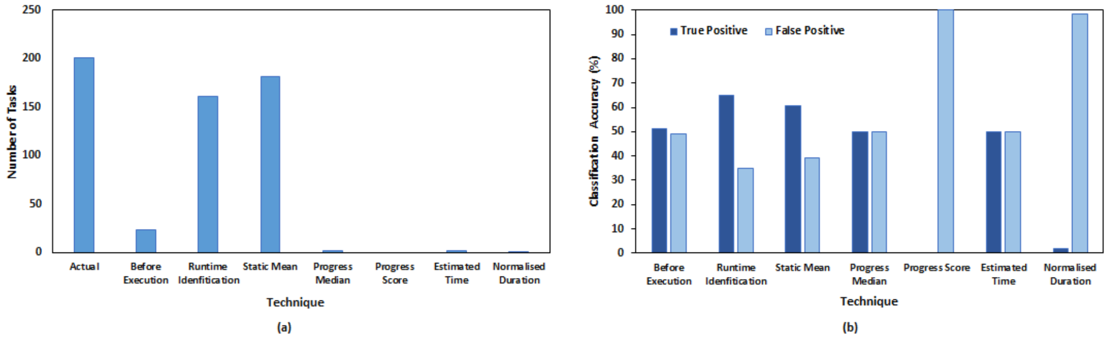


Fig. 8. Classification Accuracy for Job 4 (a) Identified Stragglers (b) Error Proportions

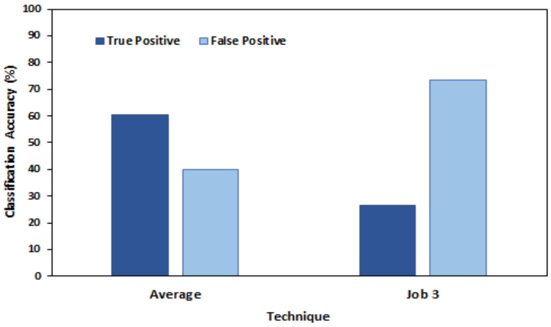


Fig. 9. Classification Efficiency of the Proposed Technique

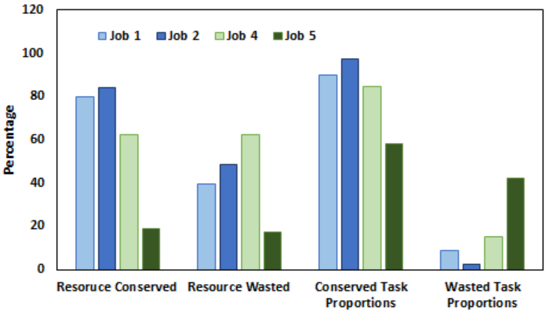


Fig. 10. Straggler and Heterogeneity Consequence

Fig. 9 presents the classification efficiency of the proposed technique in terms of the true positive and false positive rates averaged across all the studied jobs presented against the accuracy proportions of Job 3. Job 3 is a job sample without sufficient historical traces which restrains the efficiency of the proposed analytics both in terms of resource prediction and straggler classification. From Fig. 9, the effects of this limited availability of the appropriate historical samples are clearly evident, with the true positive rate for the rest of the jobs is witnessed at 60.23% which reduces to 26.53% for Job 3. Apart from this, it can be concluded that the proposed straggler analytics technique performs significantly better than the evaluated techniques. It is worthy of note that most of the evaluated techniques are focused on classifying stragglers only based on the task duration related metrics, ignoring CPU usage rate restrains their efficiency in identifying energy-aware stragglers. The classification effectiveness of the proposed analytics methodology can be attributed to the fact that it incorporates the combinational effects of both the task duration and CPU usage rate of respective tasks. The energy-aware straggler prediction before execution is still believed to be an aspiration to date, in which sense it can be postulated that the proposed straggler classification methodology is efficient in identifying task-level stragglers before the start of the actual job execution.

7.3 Resource Optimisation Performance

This section presents the efficiency of the proposed resource optimisation methodology, evaluated from the perspectives of reducing the server energy expenditures, reducing task terminations and mitigating the presence of energy-aware stragglers. The failure probability in terms of under-estimated resource levels has been verified by theoretical analysis, and further the server energy expenditures of the actual and proposed resource provisioning levels have been evaluated experimentally through simulations. GreenCloud [35, 36] simulation platform has been used to simulate the task execution within jobs. The tasks are devised to be scheduled by the green scheduler of the GreenCloud across a selected range of servers. For energy management, DVFS has been enabled on the both the physical servers and the processing VMs to dynamically adjust the internal scheduling of tasks, so that the VMs try to extend the task execution time to exploit DVFS in accordance with the workload intensity. The resource intensity of the tasks has been reflected in the task size (presented in bytes) and the proportional idleness has been reflected by imposing corresponding load on the servers depending on the idle proportions incurred in the actual and proposed level of resource provisioning accordingly.

7.3.1 Straggler and Heterogeneity Consequence

Fig. 10 presents the proportions of tasks achieving resource conservation and resource wastages within the studied jobs based on the proposed methodology. Whilst the conserved task proportions reflect the achieved reduction in the originally provisioned resource level for tasks within jobs along with the conversation percentage achieved, the wasted task proportions depict the tasks for which the resources are over-estimated than the actual level along with the percentage of resources wasted for the tasks within the studied jobs. None of the over-estimated tasks within Job 0 experience resource wastages, (i.e.,) the proposed resource provisioning level are much less than the actual level for all the tasks within Job 0. Thus, an average of 82.24% of resource levels are conserved across all the tasks encompassed within Job 0.

From Fig. 10, resource conservation has been achieved for 90% of tasks with an average of 79.93% of conservation in Job 1, 97.43% of tasks with an average of 84% of conservation in Job 2, 84% of tasks with an average of 62.54% of conservation in Job 4 and 58% of tasks with an average of 19.12% of conservation in Job 5 respectively. Conversely, the proposed methodology has over-estimated the resource level than the originally provisioned level for 9% of tasks with an average of 39.35% of wastages for Job 1, 3% of tasks with an average of 48.36% of wastages for Job 2, 15% of tasks with an average of 62.65% of wastages for Job 4 and 42% of tasks with an average of 17.3% of wastages for Job 5 respectively. It is clearly evident that resource conservations are achieved for a majority of the task proportions with significant reduction in the actually provisioned resource levels and a minority of the task proportions are suffering marginal resource wastages. Job 5 is exhibiting a different trend, since it incurred only around 16% of resource idleness during its actual Cloud execution. Furthermore, the proportional presence of stragglers is insignificant within Job 5 and also the energy-aware stragglers do not show any notable increase in their resource consumption level from that of non-stragglers. To this end, it can be postulated that jobs exhibiting fair resource consumption trend with

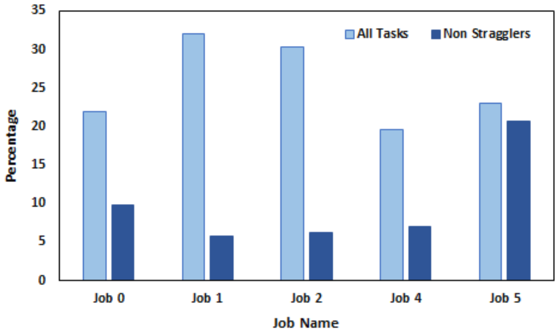


Fig. 11. Failure Probability Proportions

less than 20% resource idleness can be ignored for further analysis for exploring the scope of resource conservation.

As expected, the proportional presence of energy-aware stragglers does have a considerable impact upon the efficiencies of resource provision optimisation module. Job 0 with 8% of energy-aware stragglers is not suffering any excess resource wastages resulting from over-estimation. Increased proportions of energy-aware stragglers at around 27%, 36% and 40% are witnessed in Job 1, Job 2 and Job 4 respectively, which has reflected in the corresponding wasted proportions of resources within the respective jobs. Such observations are reflecting the fact that increased proportions of energy-aware stragglers may result in respective increase in the proportional resource wastages through over-estimation of resource levels based on the proposed analytics methodology. However, the proportions of task for which their resource levels are over-estimated are still insignificant, whereby the impacts of the over-estimation factor have been maintained at the minimum possible level to reduce their energy impacts upon the overall energy consumption.

7.3.2 Failure Probability

Fig. 11 presents the failure probability proportions of tasks within their respective job during the actual execution, resulting from the under-predicted resource levels by the proposed methodology. The effectiveness of the proposed methodology is evaluated from two different perspectives: firstly, the failure probability has been evaluated for all the tasks encompassed within a given job despite their resource consumption behaviour, and secondly the actual and classified non-stragglers within jobs are isolated to evaluate the failure probability of non-stragglers. From Fig. 11, around 11% of the total tasks are vulnerable for resource related terminations which reduces to 9% for non-stragglers for Job 0. Similarly, the failure probability has been witnessed at 32% for all tasks and 5% for non-stragglers for Job 1, 30% for all tasks and 6% for non-stragglers for Job 2, 19% for all tasks and 7% for non-stragglers for Job 4 and 23% for all tasks and 20.64% for non-stragglers for Job 5. Again, Job 5 is not performing as expected exhibiting an increased proportions of job failure probability for all tasks and non-stragglers. This can again be attributed to the fair execution profile of Job 5, again it can be recommended that jobs with fair execution trend do not project the scope for further reduction in resource expenditures.

It is clearly evident that the failure probability for non-stragglers are significantly lower than the entire task group including stragglers, exhibiting the efficiencies of the proposed methodology in reducing the failure probability for non-stragglers. This failure probability of tasks can further be reduced through vertical scaling during runtime. Failure probability can also be reduced by increasing the margin of the over-commitment factor proposed in the over-estimation protocol, in such a way that the over-commitment factor for the discussed categories can be scaled up to further reduce the proportions of under-estimation of resources. However, increasing the over-commitment margin will increase the resource provisioning levels of other task groups for which the resources have already been over-estimated. This may lead to a reduction in the overall energy conservation of resources estimated by the proposed methodology.

7.3.3 Energy Efficiency Analysis

This section is intended to exhibit the effectiveness of the proposed methodology in conserving the server energy expenditures achieved through the reduction in the amounts of resources spent on the job execution. Fig. 12 displays the energy expenditure statistics obtained from the simulation of Job 0, Job 1 and Job 5 (however all the jobs have been discussed in detail in this section), for the actually assigned amounts of resources and the proposed level of resource provision. The effectiveness of the proposed methodology has been presented for the classified energy-aware stragglers and non-stragglers respectively in Fig. 12, for an equivalent selection of random tasks representing the two group of classification. On a coarse-grain, it is clearly evident that the proposed methodology performs better in reducing the server energy expenditures for tasks classified as non-stragglers than those classified as energy-aware stragglers within a given job. This is because of the straggler classification effectiveness of the proposed methodology well before the start of the actual execution. For instance, in the case of Job 0, tasks 0, 1, 2, 7, 43, and 48 are classified as energy-aware stragglers by the proposed offline analytics. Though the proposed methodology proposes only a marginal reduction for tasks 0, 1 and 2 and 43, a significant reduction in server energy expenditures can be achieved for tasks 7 and 48 within the classified energy-aware straggler group. Despite being classified as energy-aware stragglers, the resource provisioning level of task 7 and 48 has been moderated by the proposed resource optimisation module based on the runtime factors. For non-straggler tasks within Job 0, a significant proportions of energy conservation have been achieved fairly across all the tasks. Though a significant reduction in the energy expenditures has been achieved for non-stragglers within Job 1, only a marginal reduction in server energy expenditures has been achieved by the proposed methodology for most of the classified energy-aware stragglers. Furthermore, task 23 within Job 1 is a typical example where the proposed methodology is estimating the resource provisioning level that exceeds the actual level of resources originally

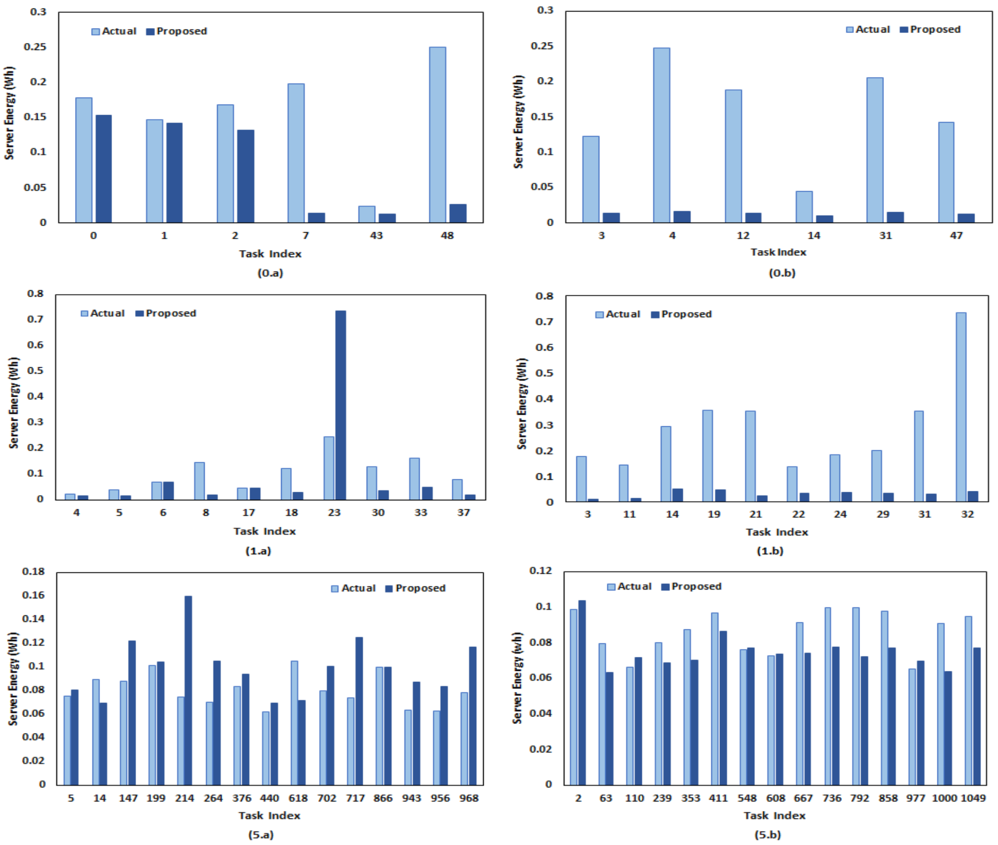


Fig. 12. Server Energy Expenditures (a) Classified and Actual Stragglers (b) Classified Non-Stragglers {(0) Job 0 (1) Job 1 (5) Job 5

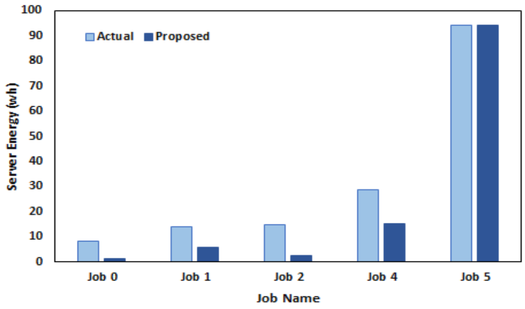


Fig. 13. Overall Energy Conservation Statistics

provisioned, in which sense this would incur energy wastages by incurring more proportions of resource idleness. This is attributed to the fact task 23 being inconsistent in its behaviour during historical execution profiles in terms of its straggling behaviour and process efficiency and further the process capacity being less reliable for the current execution. However, considering the overall impacts of this over-estimation upon the server energy conservation for all the tasks within Job 1, the excess energy consequence of task 23 is insignificant.

Job 2 is a typical example where task 134 has been estimated to consume more resources which is then turning out to be a non-straggler during the actual execution, again this is due to task 134 being less reliable in its straggler behaviour. But, a significant reduction in sever energy conservation has been achieved for both the task groups within Job 2 based on the proposed methodology. The influence of an increased proportions of straggling tasks within Job 4 are clearly evident, where only a marginal reduction in server energy expenditures has been achieved for most of the classified energy-aware stragglers. Furthermore, the achieved reduction in the proportional server energy expenditures for non-stragglers is much less than those of the other studied jobs, however still better than those of the actually provisioned level of resources for non-stragglers within Job 4. Job 5 has been suggested not to reduce the actually provisioned resource levels based on the theoretical verification, driven by the effects of straggler consequence, failure probability and importantly the witnessed level of reduced resource idleness (being less than 20%) during the historical execution profiles. The simulation experiments are also presenting us with similar inferences, since most of the energy-aware stragglers have been over-estimated than the actual level of resource provision. Also, only a marginal reduction in the server energy expenditures can be achieved for the group of non-stragglers.

Fig. 13 presents the overall energy conservation statistics across all the encompassed tasks within the studied jobs, comparing the server energy expenditures incurred by the proposed level of resource provision against the actually provisioned level of resources. The statistics presented in Fig. 13 has been moderated with the proportions of tasks achieving energy conservation and energy wastages within a given job accordingly. From Fig. 13, an overall reduction in the server energy expenditures can be achieved by a margin of 82.75%, 58.18%, 82.53%, 47.04% and 0.11% based on the proposed resource provisioning level across all the tasks encompassed within Job 0, Job 1, Job 2, Job 4 and Job 5 respectively. The difference in the achieved reduction in server energy expenditures across the studied jobs are attributed to the job heterogeneity in terms of the straggler proportions, difference between the resource consumption level of stragglers and non-stragglers within a given job, task behaviour consistency in terms of straggling behaviours, process efficiency and process capacity.

8 Conclusion

This paper proposed a novel analytics driven resource optimisation framework to optimise the level of resource provisioning whilst executing tasks in the datacentres, for the purpose of reducing resource wastages incurred though the presence of resource idleness during task execution. The proposed framework includes three integral components such as the resource estimation module, straggler classification module and the resource optimisation module. Estimating the resource consumption levels of the tasks encompassed within jobs a priori, tasks within a given job are classified based on their resource intensity and execution trend. Further, the estimated resource levels are optimised based on their classified intensity and several runtime factors affecting the task execution. The effectiveness of every integrated module is evaluated both theoretically and through practical experiments, which proves the effectiveness of the proposed analytics methodology in estimating the resource requirements of the tasks with reliable level of accuracy. The straggler classification module is efficient in classifying energy-aware stragglers well before the start of the actual task execution, and also effectively identifies the energy-aware stragglers during runtime. Furthermore, the resource optimisation module incorporates the descriptive knowledge of the task execution and postulates a resource provisioning level for tasks within jobs, in such a way that the originally provisioned resource level is considerably reduced, whereby the incurred proportions of resource idleness can be significantly reduced with minimal probability of task failures.

The proposed approach of resource provision optimisation performs better for non-stragglers than the energy-aware stragglers, in such a way that the resource estimation of non-stragglers presents better reliability. Further, the straggler classification framework includes a marginal proportion of false positive rate which has reflected in a marginal level of excess resource provision for a very few tasks within jobs. Though marginal, reducing such false positive overheads benefits achieving better reduction in server energy expenditures. Investigating the possibility of enhancing the crispness of resource estimation of energy-aware stragglers and reducing the false positive rates of the classification approach is one of our future research directions. Additionally, evaluating the efficiencies of the proposed framework in classifying jobs with cascaded tasks is another future objective of our research.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China under Grants No. 61502209 and 61502207, the Natural Science Foundation of Jiangsu Province under Grant BK20170069 and UK-China Knowledge Economy Education Partnership. Lu Liu is the corresponding author.

References

[1] J. Patel, V. Jindal, I.-L. Yen, F. Bastani, J. Xu, and P. Garraghan, "Workload Estimation for Improving Resource Management Decisions in the Cloud," presented at the Twelfth International Symposium on Autonomous Decentralized Systems, Taichung, 2015.

[2] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, "Bobtail: Avoiding Long Tails in the Cloud," in Proc. of 10th USENIX conference on Networked Systems Design and Implementation, Lombard, 2013, pp. 329-342.

[3] P. Garraghan, X. Ouyang, P. Townend, and J. Xu, "Timely Long Tail Identification through Agent Based Monitoring and Analytics," presented at the 18th International Symposium on Real-Time Distributed Computing, Auckland, 2015.

[4] N. J. Yadwadkar and W. Choi, "Proactive Straggler Avoidance using Machine Learning," University of Berkeley, 2012.

[5] X. Ouyang, P. Garraghan, D. Mckee, P. Townend, and J. Xu, "Straggler Detection in Parallel Computing Systems through Dynamic Threshold Calculation," presented at the 30th International Conference on Advanced Information Networking and Applications (AINA), Crans-Montana, 2016.

[6] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: attack of the clones," presented at the Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation Pages, Lombard, 2013.

[7] Q. Chen, P. Grosso, K. v. d. Veldt, C. d. Laat, R. Hofman, and H. Bal, "Profiling energy consumption of VMs for green cloud computing," in Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC, 2011, pp. 768-775.

[8] I. Takouna, W. Dawoud, and C. Meinel, "Energy efficient scheduling of HPC-jobs on virtualize clusters using host and VM dynamic configuration," ACM SIGOPS Operating Systems Review, vol. 46, pp. 19-27, 2012.

[9] Z. Zhang and S. Fu, "Characterizing power and energy usage in cloud computing systems," in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 146-153.

[10] P. Raycrofta, R. Jansena, M. Jarus, and P. R. Brennera, "Performance bounded energy efficient virtual machine allocation in the global cloud," Sustainable Computing: Informatics and Systems, vol. 4, pp. 1 - 9, March 2014.

[11] N. Bessis, S. Sotiriadis, V. Cristea, and F. Pop, "Modelling requirements for enabling meta-scheduling in inter-clouds and inter-enterprises," in Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on, 2011, pp. 149-156.

[12] C. Chen, B. He, and X. Tang, "Green-aware workload scheduling in geographically distributed data centers," in 4th Int. Conf. on Cloud Computing Technology and Science (CloudCom), 2012, pp. 82-89.

[13] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," Future Generation Computer Systems, vol. 28, pp. 755-768, 2012.

[14] P. Graubner, M. Schmidt, and B. Freisleben. (April 2013) Energy-Efficient Virtual Machine Consolidation. IT Professional. 28 - 34. Avail-

able: http://ieeexplore.ieee.org/document/6193088/

[15] A. Corradi, M. Fanelli, and L. Foschini, "VM consolidation: A real case based on OpenStack Cloud," Future Generation Computer Systems, vol. 32, pp. 118 - 127, 2014.

[16] J. Rosen and B. Zhao, "Fine-Grained Micro-Tasks for MapReduce Skew-Handling," 2012.

[17] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "SAMR: A Self-adaptive MapReduce Scheduling Algorithm In Heterogeneous Environment," presented at the 10th IEEE International Conference on Computer and Infomation Technology (CIT 2010), Bradford, 2010.

[18] Q. Chen, C. Liu, and Z. Xiao, "Improving MapReduce Performance Using Smart Speculative Execution Strategy," IEEE Transactions on Computers, vol. 63, pp. 954-967, 24 January 2014.

[19] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," presented at the 8th USENIX conference on Operating systems design and implementation, San Diego, 2008.

[20] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and Faster Jobs using Fewer Resources," presented at the Proceedings of the ACM Symposium on Cloud Computing, Seattle, 2014.

[21] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, et al., "Reining in the outliers in map-reduce clusters using Mantri," presented at the Proceedings of the 9th USENIX conference on Operating systems design and implementation Pages, Vancouver, 2010.

[22] S.-W. Huang, T.-C. Huang, S.-R. Lyu, C.-K. Shieh, and Y.-S. Chou, "Improving Speculative Execution Performance with Coworker for Cloud Computing," presented at the IEEE 17th International Conference on Parallel and Distributed Systems, Taiwan, 2011.

[23] X. Ouyang, P. Garraghan, C. Wang, P. Townend, and J. Xu, "An Approach for Modeling and Ranking Node-Level Stragglers in Cloud Datacenters," IEEE Int. Conf. on Services Computing, 2016.

[24] J. Panneerselvam, L. Liu, and N. Antonopoulos, "InOt-RePCoN: Forecasting user behavioural trend in large-scale cloud environments," Future Generation Computer Systems, pp. 1 - 20, 1 June 2017.

[25] J. Panneerselvam, L. Liu, and N. Antonopoulos, "Characterisation of Hidden Periodicity in Large-Scale Cloud Datacentre Environments," presented at the 13th IEEE International Conference on Green Computing and Communications, Exeter, 2017.

[26] J. Panneerselvam, L. Liu, N. Antonopoulos, and M. Trovati, "Latency-Aware Empirical Analysis of the Workloads for Reducing Excess Energy Consumptions at Cloud Datacentres," presented at the IEEE Symposium on Service-Oriented System Engineering (SOSE), Oxford, 2016.

[27] P. Garraghan, I. S. Moreno, P. Townend, and J. Xu, "An Analysis of Failure-Related Energy Waste in a Large-Scale Cloud Environment," IEEE Transactions on Emerging Topics in Computing, vol. 2, pp. 166-180, 04 February 2014..

[28] "Google Cluster Data V2," Google, Ed., 2 ed, 2011.

[29] C.-F. Wang, H. Wen-Yi, and C.-S. Yang, "A Prediction Based Energy Conserving Resources Allocation Scheme for Cloud Computing," presented at the IEEE International Conference on Granular Computing (GrC), Noboribetsu, 2014.

[30] W. Fang, Z. Lu, J. Wu, and Z. Cao, "RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center," presented at the IEEE Ninth Int. Conf. on Services Computing, Honolulu, HI, 2012.

[31] N. Roy, A. Dubey, and A. Gokhale, "Efficient Autoscaling in the Cloud using Predictive Models forWorkload Forecasting," presented at the IEEE 4th International Conference on Cloud Computing, Washington, DC, 2011.

[32] I. S. Moreno and J. Xu, "Customer-Aware Resoruce Overallocation to Improve Energy Efficiency in Real-Time Cloud Computing Data Centres," presented at the International Conference on Service-Oriented Computing and Applications (SOCA), Irvine, CA, 2011.

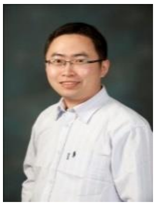
[33] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen, "Workload Predicting-Based Automatic Scaling in Service Clouds," presented at the Sixth International Conference on Cloud Computing, Santa Clara, CA, 2013.

[34] T. Li, J. Wang, W. Li, T. Xu, and Q. Qi, "Load Prediction-based Automatic Scaling Cloud Computing," presented at the International Conference on Networking and Network Applications, 2016.

[35] GreenCloud 'Simulating Energy-Efficient Clouds'. Available: https://greencloud.gforge.uni.lu/publications.html

[36] D. Kliazovich, P. Bouvry, and S. U. Khan, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," The Journal of Supercomputing, vol. 62, pp. 1263–1283, December 2012.

John Panneerselvam received his Ph.D. from the University of Derby and currently is a Lecturer in Computing at the University of Derby, United Kingdom. His current research is focused on energy efficient cloud systems and he has published his recent research works in notable peer reviewed international conferences, journals and as book chapters. He is an active member of IEEE and British Computer Society, and his research interests include Cloud Computing, Big Data Analytics, Opportunistic Networking and P2P Computing.



Lu Liu is currently the Head of the Department of Electronics, Computing and Mathematics in the University of Derby and adjunct professor in the School of Computer Science and Communication Engineering at Jiangsu University. Prof. Liu received his Ph.D. degree from University of Surrey. He is the Fellow of British Computer Society and Member of IEEE. Prof. Liu's research interests are in areas of Cloud Computing, Social Computing, Data Analytics, Service-Oriented Computing and Peer-to-Peer Computing.

Nick Antonopoulos is currently the Pro Vice-Chancellor of Research in the University of Derby and the University of Derby Technical Coor-dinator of the framework collaboration with CERN as well as the ALICE experiment. Nick holds a PhD in Computer Science from the University of Surrey in 2000. His research interests include Cloud Computing, P2P Computing, software agent architectures and security. Nick has many years of academic experience and has pub-lished more than 150 articles in fully refereed journals and internation-al conferences.