Cost Efficient Batch Processing in Amazon Cloud with Deadline Awareness

Kabin Tamrakar
Dept. of Computer Science
Oslo and Akershus University
College of Applied Sciences
Email: kabin.tamrakar@gmail.com

Anis Yazidi
Dept. of Computer Science
Oslo and Akershus University
College of Applied Sciences
Email: anis.yazidi@hioa.no

Hårek Haugerud
Dept. of Computer Science
Oslo and Akershus University
College of Applied Sciences
Email: harek.haugerud@hioa.no

***Abstract*—Amazon spot instances have become a very popular alternative for cost-saving in the cloud. The spot instances are prone to abrupt termination whenever the spot market price exceeds the bid price. In this paper, spot instances are resorted to in task instances' group of Amazon Elastic Map Reduce (EMR) cluster to process batch jobs with deadline. Amazon EMR makes it convenient to process Big Data with the aid of the Hadoop framework. However, the processed intermediate results in the task nodes of the cluster are lost if the spot instances gets terminated which can cause processing delay. The cost efficiency can be realized by exploiting the non-real time nature of batch computing for Big Data. Two algorithms are devised for achieving cost efficient processing in Hadoop MapReduce. Both algorithms process data in divisions such that abrupt termination of spot instances only affects the last division. Based on monitoring the progress at given checkpoints, task group's capacity is resized to complete the processing within the deadline. Progress is measured in terms of the number of completed work divisions. The first algorithm begins with some spot instances whose number is initially estimated. To complete processing of all data in time, on-demand instances are deployed after a certain threshold time. The second algorithm starts by using higher number of spot instances than required to complete the work within the given deadline. Therefore, it has higher chance to rely solely on instances during the whole execution of the batch job. On-demand instances are deployed only in case of slow progress caused by termination of the spot instances combined with subsequent unsuccessful bids. The experiments show that both algorithms are able to minimize the processing cost. The second algorithm further minimizes the cost in most cases.**

***Index Terms*—Batch Processing, Spot Instances, Elastic Map Reduce, Deadline-Awareness.**

I. Introduction

Cloud computing has become indispensable for both big and small enterprises to perform numerous IT operations today. For dynamic workloads, it is often economical to rent cloud servers than building and configuring dedicated infrastructure [1]. Cloud computing offers major advantages such as reduced costs, automation, flexibility, mobility and consumerization. Despite bringing beneficial aspects to cloud users in terms of costs, flexibility and availability, it poses unique challenges to cloud service providers themselves. Cloud users may demand significant resources during peak hours. In those peak hours, it is vital for service providers to guarantee that sufficient cloud resources are available to meet the service level agreements (SLA) commitments to the cloud users. This means the cloud providers have to arrange significantly large resource pool to serve the users' demand anytime. However, during off-peak hours there is a significant waste of the resource of the cloud infrastructure [2]. To cope with the under-utilization of cloud resources, cloud service providers offer different pricing options so as to facilitate a wide variety of applications depending on computing requirements [3], [4]. The common cloud pricing schemes for virtual machine compute instances are namely reserved, on-demand and spot instances [3]. Reserved instances provide users with a one-time payment. The instance is reserved over a long period ranging from 1 to 3 years with the advantage of hourly discounted prices based on usage [3]. While on-demand instances are offered as an hourly instances without any long-term commitment. When it comes to spot instances, users bid for spare resources without any guarantee on uninterrupted execution. The cloud provider can revoke the spot resources once the market price exceeds the bid price [3], [4]. With higher risks and uncertainty of being revoked anytime, the spot resources are by as much as 10 times cheaper than the equivalent on-demand resources which cannot be revoked by cloud providers for paid hours [4]. Any of the aforementioned compute instances can be used for a large variety of workload use-cases such web services, batch processing, transaction processing, analytics, high performance computing, database computing, etc. The cloud spot resources can be deployed to perform these use-cases with large monetary benefits. However, spot resources might not be always available. Price fluctuations might occur on the basis of imbalance between supply demand which results into immediate termination by cloud provider as soon as bidding price falls under the spot market price. Thus, in order to take full advantage of spot instances, a resource provisioning system should deploy an effective bidding algorithm along with fault-tolerant mechanisms and switching to on-demand resources to maintain the system's availability and reliability.

In this paper, we investigate the problem of batch processing with deadline awareness using amazon spot instances. The intuition behind our work is the fact that the deadlines can give some slack to a batch job processing, and thus, lower cost can achieved by differing partial execution of the job til spot instances are obtained at low prices. Amazon Elastic MapReduce (EMR) cluster [5] will be implemented for processing the batch jobs. Amazon EMR's Hadoop Cluster is composed of Master Instance Group and Slave Instance Groups - Core and Task [6]. In this paper, we choose to run master nodes and core slave nodes on consistent instances since master nodes constitute the central part of cluster and therefore should not be interrupted while executing the batch job. Core slave nodes also run YARN ResourceManager service for both resource management and HDFS NameNode service. According to our design, task nodes group can be re-scaled anytime so that for shorter deadline needs, the capacity can be increased and vice-versa. It is possible to re-scale core nodes, however, shrinking them may introduce the risk of loosing data as they store their data in the HDFS.

A. Related work

There has been a significant amount of research on efficient utilization of spot market to cut down the operation costs in many applications. Spot instances are characterized by fluctuating prices that are driven by the imbalance between supply and demand. In addition, spot instances are characterized by irregular availability. The major setback of using spot instances is probably their abrupt termination as discussed earlier. Thus, making a stable system out of spot instances is a challenging task, and many factors need to be considered and studied. We shall now review some representative studies that addressed those issues. The research conducted in [1] focuses on minimizing the cost of running "always-on" internet-based services by the use of spot markets. Please note that at least 99.99% availability is a widely accepted standard to tag the Internet-based service as always-on. The authors discuss two main bidding schemes to guarantee availability, namely, reactive bidding and proactive bidding. In the case of reactive bidding algorithm, migration to on-demand instances takes place after the spot server is revoked which causes disruption to the service. Therefore, the authors proposed a proactive bidding algorithm which senses the varying spot market beforehand for gracefully shutting down of spot instances and migrating their tasks to on-demand instances. The authors propose three migration steps namely Forced Migration, Planned Migration and Reverse Migration on the conditional basis of current spot price, bidding price and on-demand price. For the migration from spot to on-demand and vice versa, OS mechanisms such as Nested vitrualization, Live migration, Bounded Memory, check-pointing and Lazy VM restore were used. Both proactive and reactive bidding result into a significant cost reduction by a factor ranging from 17% to 33% compared to using all on-demand instances. The unavailability of the service using the proactive algorithm is smaller by a factor of 2.5 to 18 than using the reactive algorithm.

In the study reported in [7], the author investigated building highly available cluster in hybrid cloud. For handling additional compute for spikes during peak hours, an automated cloud bursting solution in public cloud is developed which uses Amazon spot instances to leverage the pricing model. Basically, the paper deals with setting up a hybrid cloud using Apache Mesos to make a unified platform composed of a private cloud and Amazon public cloud with the focus on maximizing availability.

In [8], the authors developed WOHA, a framework that efficiently schedule deadline-aware workflows in MapReduce. Under the WOHA framework, client nodes are responsible for generating scheduling plans locally and sending them to master node which will use them for scheduling plans. The authors propose a scheduling algorithm which assigns priorities among workflows. The experimental evaluation covers three job prioritizing algorithms, namely Highest Level First (HLF), Longest Path First (LPF) amd Maximum Parallelism First (MPF).

The study reported in [9] carried out some benchmark performance testing of MapReduce applications. Even-though MapReduce has emerged as efficient solution for handling huge data analytics, the performance can be questioned in the presence of individual machine failures. Those failures might cause significant delays in execution of jobs as they have to be rescheduled into new nodes. The authors report a test using the *mrbench* benchmark where they launch a single, fixed-work trivial map or reduce task, then, incrementally increase the number of tasks. The authors concluded that the increase in the number of tasks introduced some performance overhead. In fact, using *mrbench* benchmark, the over-head was found to be nearly 0.111 seconds for map task and 0.105 seconds for the reduce task in an experimental setup of 34 compute nodes. Since typical MapReduce clusters consist of hundreds of compute nodes, those overheads can be even more significant.

The study reported in [10] suggested that calculating effective bid price depending on jobs' interruption constraints might decrease the processing cost significantly as well as minimize the job interruption [10]. For determining the bid price, a historical data of 2 months' statistics of spot market price was utilized. The authors employed one-time request and persistent request as bidding strategies for master and slave nodes in MapReduce jobs. One time request was performed for a single spot instance with high bid price. As interruption is allowed in slave node, persistent request was made for each slave node.

II. Design of Our Solution

A. Data Processing and Cluster Scaling

Before beginning the actual data processing, the data should be made available to S3 bucket. Log generating bash script can be run into an EC2 [11] instance in the same region as bucket, then we can compress the generated data and transfer it to S3 bucket. It is also possible to generate sample logs locally and then upload them into S3 bucket. Data processing engine has to be ready. The EMR cluster should be up and running. The cluster will be provisioned with at least 1 master node and 2 core nodes. Later on the basis of value of $\boldsymbol{N_x}$, which is the required number of task instances to complete the job in time, the cluster will be scaled up and down. The value of N_x will be adjusted at predefined checkpoints.

The overall system design of the cluster is depicted in Fig. **1**.

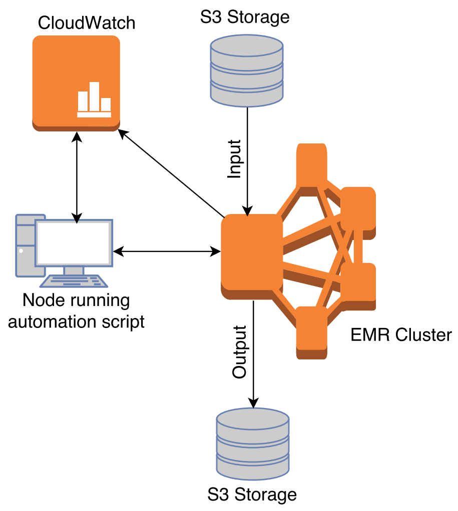

Fig. 1. System Block Diagram

By using Python Boto3 SDK [12], CloudWatch [13] and Log analysis, all the activities in the cluster will be monitored and decision for cluster resizing will be initiated from the node running the automation script. For improved performance of the job flows, all nodes and S3 bucket should be launched in the same Amazon EC2 availability zone to ensure high data access rate and in order to eliminate inter-zone data flow charges.

B. Estimation Phase

In this phase approximate size of cluster will be calculated based on desired deadline in terms of "m1.medium" instances.

- Let $\boldsymbol{N_m}$ be the number of mappers running in parallel in an instance. The default mappers capacity of "m1.medium" is 2. Similarly other instances like "m1.xlarge" can run 8 mappers in parallel by default. If a cluster consists of 1 master node and 1 core node, the total instance mapper capacity would be $\boldsymbol{N_m}$ since only core and task instances are involved in actual processing.
- What is number of mappers the job requires? The data is divided into x sample log files with equal size of about 64 MB. This means that the number S_m of splitters and mappers are equal, i.e, $S_m = x$.
- How long would it take to process sample files? It is assumed that it would take t_m time to process m sample files. It should be noted that number of sample files is equal to number of mappers that can run in parallel in that instance.
- Finally, to meet the desired deadline t_d, approximate estimated number of nodes [14] would be N_x which is given as below:

$$N_x = \frac{S_m \cdot t_m}{N_m \cdot t_d}$$

This number N_x is crucial and needs to be maintained during processing.

C. Challenges of using Spot instances in Cluster

When a cluster starts processing the data, mappers and reducers are assigned to the slave nodes (core and task instances). The output of the mapper is buffered and later dumped into local disk of each individual mapper nodes as the intermediate results when threshold of buffer is reached. As described earlier, spot instances could be terminated whenever the market price exceeds the bid price or the spot pool is empty. When using spot instances as task nodes in the EMR cluster, mappers running on those instances store the intermediate map outputs in those spot nodes themselves and therefore the cluster risks loosing those intermediate results. In order to address this challenge, we propose to divide the job into more partial jobs such as 10 partial jobs for example. After completion of one partial job, we store the output and start next partial job and so on. Therefore, a maximum of 10% processing would be lost as each step contributes to 10% of the total work. For example if we are in the 4th execution step after finishing 3 partial jobs, then 30% work has already been finished. In the case where spot instances gets terminated, only the partial amount above 30% of work would be lost and this partial amount of work could be recovered later with new spot instances or on-demand instances before the deadline is met.

The partial jobs are tasks denoted by Task1, Task2, up to Task10. The total size of file to be processed will be divided into 10 parts and each part will be processed in subsequent steps. For example, if the 5th step is running and the spot instances got terminated for some reason, only 5th step has to be reprocessed again but not step 1 to step 4. This setup is even beneficial in the drastic case of failure of the master node itself because the processed results are already in persistent S3 storage. The extra time to recover the processing would be cluster setup and configuration time which is about 5 to 10 minutes for EMR cluster with core Hadoop.

D. Algorithms

Two algorithms were devised to achieve the goal of minimizing the cost of batch jobs processing in EMR cluster. Both algorithms leverage spot instances to minimize the cost of the processing job in the cluster for timely completion of the batch jobs.

The first algorithm **Deadline Aware Auto Bidding Scaling Algorithm (DAAB)** is based on adjusting task capacity in each time slot using spot instances only until a threshold time. After the threshold time, on-demand instances are eventually used in order to complete the tasks within the given deadline. The bidding strategy chosen for this algorithm is based on the median of the last spot market prices. While the second algorithm **Deadline Aware Progress Adaptive Burst Bidding Algorithm(DPB)** applies the principle of burst bidding for spot instances i.e. using more spot instances than required to finish job before time which also decreases the probability of using on-demand instances because the processing would probably finish before the deadline. The bidding strategy is based on most recent spot prices. Normal and aggressive bidding is also implemented in this algorithm at defined check-points. When the progress monitored at a given checkpoint is *Behind*, aggressive bidding will be employed. While if progress at the given checkpoint is *Ahead* or *In-pace*, the bidding will be normal. The normal and aggressive bidding differs only by the value of the increment percent x. Normal bidding strategy has a value $x = 0.02$ while aggressive bidding has a value $x = 0.05$.

*1) **Deadline Aware Auto Bidding Scaling (DAAB) Algorithm:*** In this algorithm, bidding will be performed for $(N_\text{x} - \beta)$ instances. β is number of core instances running in the cluster which also contribute to the job progression. The bid price is the sum of the median price of the last 10 hours and 2 % of the difference between the on-demand price and the median. Then at every elapsed 10% time of the total deadline, the algorithm checks the progress (which will be between 1 and 10) and recalculates the new number of instances Z_x required to complete the whole processing in time. Therefore, we will bid $(Z_\text{x} - \beta)$. Bidding will be done with the same bidding price calculated before. Thus, task instances will be resized with $(Z_\text{x} - \beta)$. This will continue until 90% of time to deadline, and at that point new Z_x is recalculated and new value of $(Z_\text{x} - \beta)$ will be calculated with only on-demand instances so that to guarantee completion within the deadline.

During every 10% of time to the deadline, if the processing gets delayed either due to spot instances unavailability or abrupt termination in between, then the algorithm will compensate by resizing instances capacity at the start of next time slot. Even if processing was faster than expected which is checked at start of new time slot, the algorithm will decrease the size of instances capacity for next 10% of time to keep the progress in pace with expected line. The "ideal" case occurs whenever the current progress keeps in pace with the base-line progress from start to end. Such ideal case takes place if the bid is successful at every time slot and the allocated instances are not terminated in any of these time slots til 90% of time. The worst case occurs when no bid is successful until reaching 90% of the deadline and thus the cluster has to be provisioned with only on-demand instances. In such worse case scenario, we would need a number of task instances equal to 10 times the estimated beforehand number of instances. The other worst case scenario takes place in case of successful provisioning of spot instances but termination before every 10% progress.

*2) **Deadline Aware Progress Adaptive Burst Bidding (DPB) Algorithm:*** In this algorithm, only spot instances will be used up to 50% of time. The multiplication factor of α enables the cluster to process data as earlier as possible with spot instances. There will be two checkpoints for evaluating progress based on elapsed time and re-adjusting the number of spot instances based on latest price and/or on-demand instances to meet the deadline.

These checkpoints will be set at 50% of elapsed time to deadline and 80% of elapsed time to deadline. Moreover, til 50% reaming time to the deadline and for every elapsed 10% of time, the status of $(\alpha \cdot N_\text{x} - \beta)$ spot instances will be checked and rebidding will be performed if spot instances are terminated for some reason otherwise the same number of spot instances will be running. The value N_x is the estimated number of task instances required and which is computed in estimation phase and β is number of core instances running.

Algorithm 1: Deadline Aware Auto Bidding Scaling (DAAB)

```
Data: Get ClusterID, Cluster-zone, estimated number of
      instances as $\mathbf{N_x}$ , Deadlinetime $\mathbf{t_d}$, $\beta$ is number of
      core instances, $M$ is the median spot market price
      from 10 hours history, $B$ refers to bid price and $D$
      is on-demand price;
1  Initialize t=0;
2  BidPrice $B = M + (D - M) \cdot 0.02$;
3  $Bid(N_\text{x} - \beta, B)$;
4  Resize task instances with $N_\text{x}$ spot instances if bid
     accepted ;
5  while $t < 0.9 \cdot t_d$ do
6      foreach $t = t + 0.1 \cdot t_d$ do
7          Find Progress P;
8          RemainingTime $t_\text{r} = t_\text{d} - t$ ;
9          $\mathbf{Z_x} = (((10 - P) \cdot N_\text{x} \cdot t_\text{d})/(10 \cdot t_\text{r})) - \beta$;
10         $Bid(Z_\text{x}, B)$ ;
11         Resize task instances with $Z_\text{x}$ spot instances for
             bid price ;
12     end
13 end
14 Find Progress P;
15 Calculate $Z_\text{x} = ((10 - P) \cdot N_\text{x}) - \beta$ ;
16 Resize task instances with $Z_\text{x}$ on-demand instances ;
```

At 50% of elapsed time i.e. **Checkpoint 1**, depending on the progress, different strategies will be chosen for further processing (the case of 80% of time elapsed to deadline is similar). The total work will be divided into 10 tasks by dividing the data into equal number of files of same size for each task. So the progress "bar" can be measured in terms of 10% increment where P_0 refers to 0%, P_1 refers to 10%, P_2 means 20%, up to P_{10} which means 100% progress.

On the basis of the progress at these checkpoints, further strategies will be applied. There are two bidding strategies namely, **Bid_A** where the bid price is sum of latest price and 2% of difference between on-demand price and latest price; and **Bid_B** where the bid price is sum of latest price and 5% of difference between on-demand price and latest price.

At **Checkpoint 1**, if the progress status is *Behind*, that means the progress is less than 50% then **Bid_B** will be applied for $(\alpha \cdot N_\text{x} - \beta)$ spot instances along with N_x on-demand instances in order to improve progress against baseline progress. If the progress is *on-track*, $(\alpha \cdot N_\text{x} - \beta)$ spot instances with **Bid_B** will be employed in order to finish work ahead of time. While in case of *Ahead*, **Bid_A** will be employed with $(\alpha \cdot N_\text{x} - \beta)$ spot instances.

Algorithm 2: Deadline Aware Progress Adaptive Burst Bidding (DPB)

```
Data: $\alpha$ is multiplication factor, $\beta$ is number of core
      instances
1  Set ClusterID, Cluster-zone, Instance-number $\mathbf{N_x}$ ,
     Deadlinetime $\mathbf{t_d}$;
2  Find latest spot market price $L$, $t = 0$ ;
3  BidPrice $B = L + (D - L) \cdot 0.02$ where
     $D \rightarrow on - demand\_price$;
4  $Bid(\alpha \cdot N_x - \beta, B)$;
5  Resize task instances with $(\alpha \cdot N_x - \beta)$ spot instances for
     bid price ;
6  while $t < 0.5 \cdot t_d$ do
7      foreach $t = t + 0.1 \cdot t_d$ do
8          if previous $(\alpha \cdot N_x - \beta)$ spot instances stil
             running then
9              Continue
10         else
11             Update BidPrice $B = L + G$;
12             Resize task instances with $(\alpha \cdot N_x - \beta)$ spot
                 instances for new bid price $B$ ;
13         end
14     end
15 end
16 if $ProgressP <= 4$ then
17     Bid for $(\alpha \cdot N_x - \beta)$ spot instances with
         $B = L + (D - L) \cdot 0.05$ and $N_x$ on demand;
18 else
19     if $P == 5$ then
20         Bid for $(\alpha \cdot N_x - \beta)$ spot instances with
             $B = L + (D - L) \cdot 0.05$;
21     else
22         if $P > 5$ then
23             Bid for $(\alpha \cdot N_x - \beta)$ spot instances with
                 $B = L + (D - L) \cdot 0.02$ ;
24         else
25             Readjust all task instances size to zero and
                 exit;
26         end
27     end
28 end
29 while $t < 0.8 \cdot t_d$ do
30 end
31 if $P <= 7$ then
32     Use $((((10 - P)/2) \cdot N_x) - \beta)$ on-demand instances
         + $(N_x)$ spot instances with $B = L + (D - L) \cdot 0.05$
33 else
34     Use $(N_x - \beta)$ on-demand + $N_x$ spot instances with
         $B = L + (D - L) \cdot 0.05$
35 end
```

At 80% of time to the deadline i.e. **Checkpoint 2**, if the progress is *Ahead* or *on-track*, $(N_x - \beta)$ on-demand instances along with N_x spot instances with **Bid_B** would be used. While in case of *Behind*, the required number of on-demand instances will be calculated as $(((10 - P)/2 \cdot N_x) - \beta)$ along with N_x spot instances to finish up ahead of deadline. On-demand instances guarantee the completion of processing in time, while adding spot instances speeds up processing if they are available til the end.

III. Experimental Results

We carried out several experiments to assess the efficiency of our algorithms. We report two base-line experiments with only on-demand instances called **OD-n**, as well as six experiments involving our two proposed algorithms: (**DAAB** and **DPB** represented respectively as **DAAB-n** and **DPB-n**). The instance flavor used was "m1.medium" [15]. Hadoop 2.7.2 was used with Amazon EMR release 4.6.0.

A. Evaluation of base-line experiments OD-n

The two experiments were carried out as base-line experiments using task nodes as on-demand instances only. On the basis of estimated instance numbers (6 in this case) and the deadline time which was fixed to 10 hours, the processing time was expected to be less than 1 hour for each task. The median processing time of the two experiments OD-1 and OD-2 was 3466.5 seconds and 3478 seconds respectively. This negligible difference of 11.5 seconds suggests almost equal processing time. There were no outliers too in both boxplots, so drastic change in processing time was not observed. When comparing the processing time of **OD-1** and **OD-2**, the standard deviation was respectively 108.97 seconds and 87.72 seconds, which means 3.1% and 2.53% deviation from their respective mean. Please note that, if 5% more resources than the value obtained in the initial estimation phase were employed, then the data processing would be finished within the deadline.

We shall use these experiments **OD-1** and **OD-2** as base-line experiments.

The total cost for each of these two experiments was the same. The following Table I summarizes the total cost for running each OD-n experiment with instance flavor "**m1.medium**".

EMR cluster cost for **OD-n** in US dollars					
Instance Type	**Qty**	**Hours**	**EMR Charge/hr**	**on-demand charge/hr**	**Amount**
master	1	10	0.022	0.087	1.09
core	2	10	0.022	0.087	2.18
task	4	10	0.022	0.087	4.36
Total ($)					7.63

TABLE I
EMR Cluster cost for **OD-n** experiment

The progress vs time graph for OD-1 is depicted in Fig. 2. The time at every 10% progress is plotted in this graph. The actual process in OD-1 seemed to be slightly faster than baseline progress (100% processing in exact 10 hours i.e. 600 minutes).

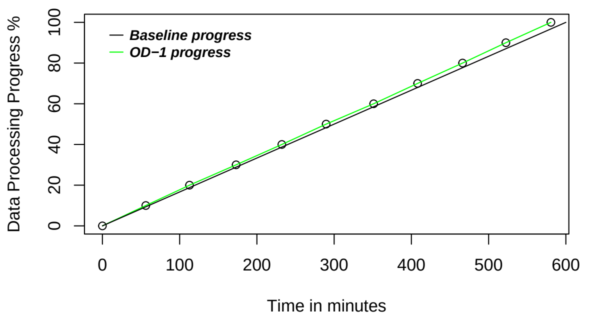

Fig. 2. Data processing progress vs time in Experiment **OD-1**

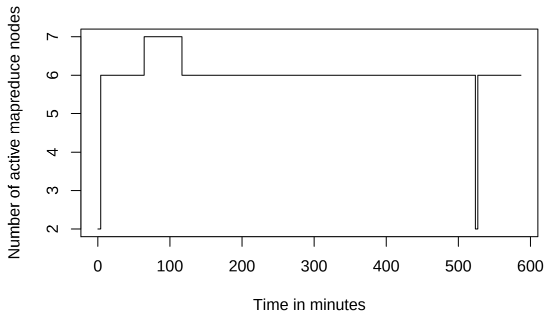

Fig. 4. Number of running MapReduce nodes in **DAAB-1**

B. Evaluation of experiments DAAB-n

Three experiments out of ten for **DAAB** are presented in details. Six experiments were not interrupted by spot termination while the other four experiments experienced termination at some point of time. **DAAB-1** experiment is discussed below. There was no spot termination once the spot instances were allocated in **DAAB-1** and while there were some spot instances terminations due to increased price in **DAAB-2** and **DAAB-3**.

1) DAAB-1 Experiment: The progress vs time graph for DAAB-1 is depicted in Fig. 3.

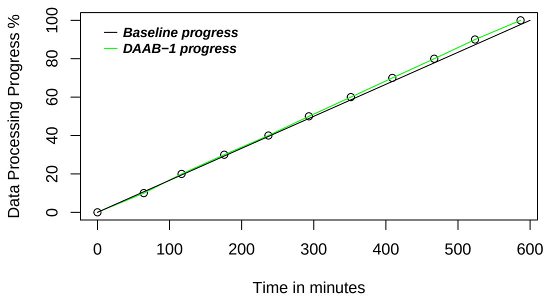

Fig. 3. Data processing progress vs time in **DAAB-1**

From Fig. **3**, all the spot instances did not terminate throughout cluster lifetime and the progress seemed to be linear with baseline progress. The job was even finished a little earlier than deadline because most of the 10% partial jobs were processed in less than an hour. As in Fig. **4**, first a bid for 4 instances was issued and after the bid being successful, the number of active Mapreduce nodes became 6 (i.e 4 spot instances plus 2 core instances running continuously). The first part of processing took more than an hour, namely, about 64 minutes and 15 seconds. Thus, the estimated number of task nodes required for completing remaining processing was 7, therefore, the task capacity was increased by 1. With 7 task nodes, the processing of data from 10% to 20% was completed in just 52 minutes and 25 seconds. At the second hour, the task nodes capacity was resized back to 6. After 90% of time, only on-demand instances were used to complete remaining processing. Since before 90% of time the last 10% progress was already started with 2 core instances, the work was completed before the deadline. If the 10th of processing started after on-demand instances were running, the processing would have been delayed because it takes between 4 and 6 minutes for running new on-demand instances. Better precaution would be required by always keeping 10% tolerance to deadline.

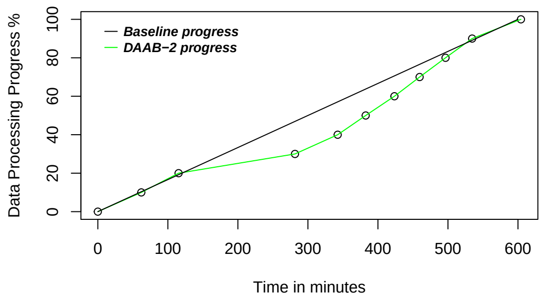

Fig. 5. Data processing progress vs time in **DAAB-2**

2) DAAB-2 Experiment: In this experiment, the spot instances were terminated after 2 hours and 3 minutes. As a consequence, Mapreduce nodes decreased from 7 to 2 (i.e. core instances only), and the part-3 processing time increased to 166 minutes and 12 seconds. This is because spot instances bidding was not successful at 3rd and 4th hour bidding. The bidding strategy of DAAB-2 was the same as the one used for the first bid calculated from the median of the last 10 hours. The spot instances were available on the start of sixth hour again. The number of required new task instances became 9, and at the beginning of 8th hour, it turned to 10 and ran for two hours. After 9 hours, only on-demand instances were used for processing. However, it took 4 minutes and 14 seconds extra time because of the extra time required for provisioning on-demand instances as task nodes. As discussed in Section III-B1 experiment, 10% tolerance to deadline would achieve the work completion in time. Due to interruption of spot instances for around 3 hours, the number of spot instances was adjusted to higher values to keep progress in pace with baseline progress.

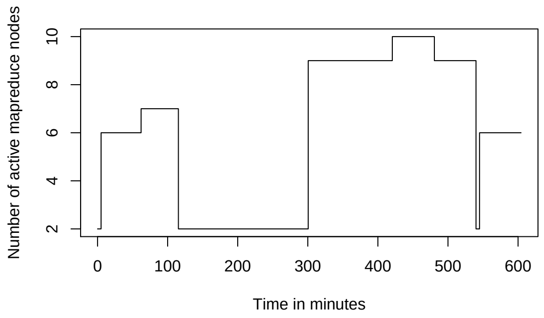

Fig. 6. Number of running MapReduce nodes in **DAAB-2**

3) DAAB-3 Experiment: In this experiment, up to 60% of progress went on pace with the baseline progress. But the 7th part took longer time (170 minutes) because of loosing spot instances. At the end, the algorithm required 18 instances to complete the processing. 16 on-demand instances as task nodes were deployed after 90% of time. It was noted that when provisioning new instances at 90% of time to deadline, consideration should be made for time the on-demand instances would take to become up and running. In this case also, we exceeded the deadline by 4 minutes.

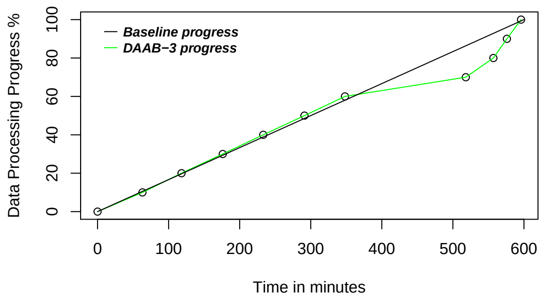

Fig. 7. Data processing progress vs time in **DAAB-2**

C. Evaluation of experiments DPB-n

1) DPB-1 Experiment: In this experiment, α was fixed to 2. Since the spot instances did not get terminated, the progress was smooth and the total data processing was finished in 290 minutes. **DPB** algorithm uses latest price for bidding. If the bid was unsuccessful, a new bid would be issued with the new calculated price at every 10% of time til 50% of total time. Due to the burst factor α, the processing was completed before half of the time.

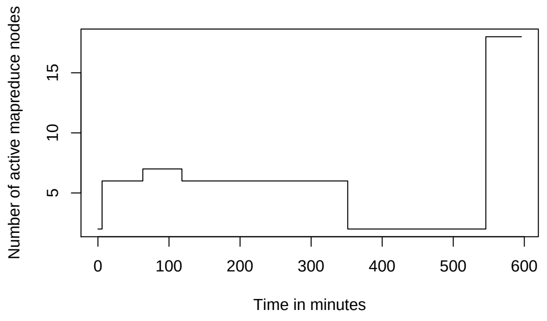

Fig. 8. Number of running MapReduce nodes in **DAAB-3**

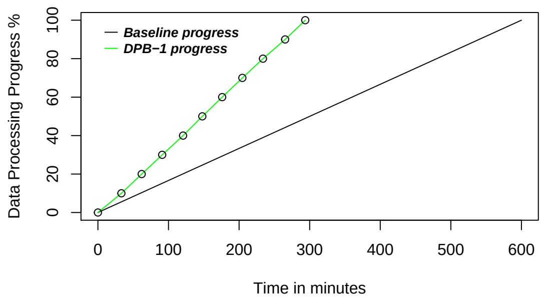

Fig. 9. Data processing progress vs time in **DPB-1**

2) DPB-2 Experiment: In the DPB-2 experiment, we experienced once spot termination after 2 and half hours. Therefore, the processing was delayed for Part-6 i.e. 50% to 60% data which took 1 hour and 5 minutes to process. The number of Mapreduce nodes over time for **DPB-2** is depicted in Fig. **11**. New bid was issued in the 4th hour after 3 hours of processing. The bid was successful and again 8 instances were resized resulting into a total of 10 map reduce nodes including 2 core nodes. After 50% of time, the progress was already 90%. Therefore, according to the terminology of **DPB**, the progress is reckoned "Ahead".

D. Cost Analysis

At this juncture, we shall analyse the cost of the aforementioned experiments. The total EMR cluster cost and task instances implementation cost were calculated. The total cost can be referred as **OD-n** experiment cost from Table **I**. This OD-n experiment sets benchmark for both performance and cost. The cost of **OD-n** experiment is the base-line price for processing batch jobs in this work. On-demand instance charge per hour for "m1.medium" instances is US $0.087. There is also an additional charge called EMR charge which is US $0.022 per hour for "m1.medium" instance [16]. The total cluster's cost for each experiment and total task instance's cost for each experiment is recorded in **Table II**.

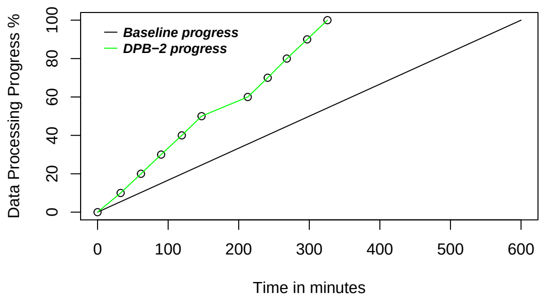

Fig. 10. Data processing progress vs time in **DPB-2**

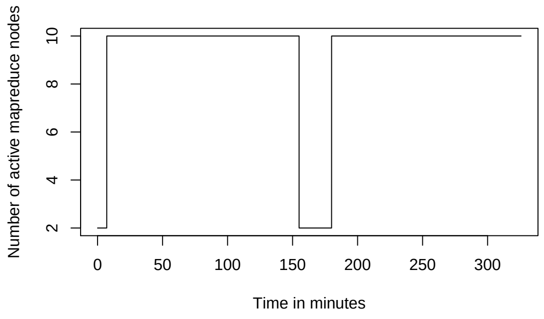

Fig. 11. Number of running MapReduce nodes in **DPB-2**

EMR cluster cost and task instants' cost in US $		
Experiment Name	**Total EMR Cost**	**Task instants' Cost**
OD-n	7.63	3.48
DAAB-1	4.99	0.83
DAAB-2	5.07	0.87
DAAB-3	5.94	1.75
DPB-1	3.31	0.58
DPB-2	3.8	0.83

TABLE II
EMR CLUSTER COST FOR ALL EXPERIMENTS

OD-n was referenced as the base experiment. From Fig. **12** and **13**, total EMR cost prices and instant task prices can be compared. The total EMR cost was US $7.63 while the breakdown cost is found Table **I**. This represents the baseline price for the cluster to process the given amount of data mentioned used in the experiments. As task nodes are on-demand instances too, the cost for task nodes is high in OD-n experiments: $3.48 for on-demand instance. The total EMR cluster price drops by 35%, 34% and 22% in experiments DAAB-1, DAAB-2 and DAAB-3 respectively. In DAAB-1 and DAAB-2, only after 90% of time, 4 on-demand instances were used as task instances. While in DAAB-3, the spot instance termination in last few hours caused the need of 16 on-demand instances for last hour processing. This increased the cost considerably.

On other hand, DPB-1 and DPB-2 are able to decrease the

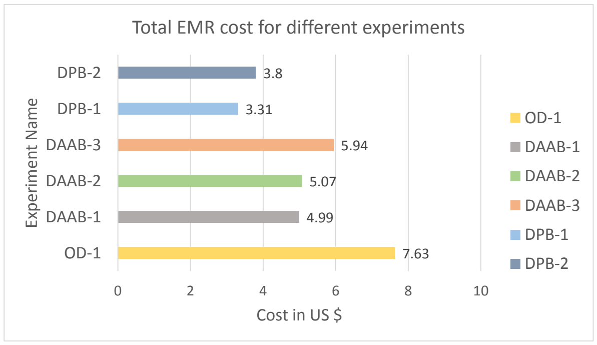

Fig. 12. Total EMR Cluster price for different experiments

cost by 56% and 52% respectively. The huge cost drop in DPB-1 and DPB-2 is also accompanied with fast processing. Within around 50% of time of the deadline, the total processing was completed, and so the cost for running 1 master and 2 core instances decreased by almost half.

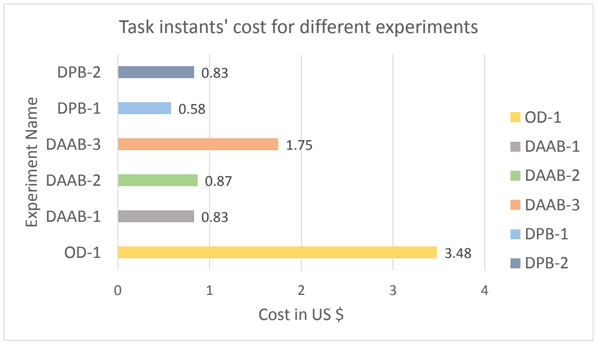

Fig. 13. Task instances price for different experiments

It is interesting to compare the prices of the task instances only. As depicted in Fig. 13, the price for task instances is US $3.48. For DAAB-1 and DAAB-2 in which only spot instances were used til 90% of time, the price had dropped significantly by 76% and 75% respectively. However, the cost for DAAB-3 dropped by only 50% due to need of using large number of spot instances at the end to meet deadline. DPB-1 and DPB-2 performed better as the price was reduced by 83% and 76%. Many experiments were done with DPB and they performed better as they ended up with the use of spot instances only with no on-demand instances. During the experiments, the price ranged from US \$0.009 to US \$0.021 in Oregon (us-west-2) region in Amazon.

The starting spot market price for DAAB-1, DAAB-2 and DAAB-3 were US \$0.0134, US \$0.0121 and US \$0.0132 respectively. While they were US \$0.0113 and US \$0.0166 respectively for DPB-1 and DPB-2 which changed over time based on the supply-demand principle of Amazon.

E. Comparison of DAAB and DPB algorithms

The bidding price should be chosen carefully in such a way that it is not very high neither too low compared to the spot market price. Higher bid price could get better priority but would eventually lead to increased costs. In **DAAB** algorithm, bid price was set as $(M + (D - M) \cdot x)$ where M stands for median price of 10 hours, D stands for on-demand instance price and x is an increment percent. This price would be used for bidding from start to the end. The median price was used to avoid possible spikes however it could be the case that the bid price calculated at the beginning would always be lower than market price. Hence, it could cause the need of more on-demand instances at the end. In **DPB** algorithm, the latest market price was used. The bid price was $(L + (D - L) \cdot x)$ where L represents latest price. The advantage of using latest price was observed in Section III-C2 experiment, where the instances were terminated due to the bid price falling under the market price. Later the bid was successful in next round of bidding after the instance was terminated. In Section III-B3 experiment, no subsequent bid was successful once the spot instances were terminated. Thus the algorithm required a larger number of on-demand instances to complete the processing in time. **DPB-n** experiments seemed to have cheaper cost than **DAAB-n** experiments. This is due to the need of using on-demand instances at the end in **DAAB-n** experiments. The situation was even worse in **DAAB-3** experiment because of need of using more on-demand instances at the end. However, the likelihood of needing on-demand instances after the first checkpoint in **DPB-n** experiments is low due to use of the multiplier factor α for the required number of instances as well as the implementation of latest price in each new bid. Thus, there is a high chance that the **DPB-n** experiments only require spot instances for task nodes due to use of α, which plays the role of "bursting" factor. We performed ten experiments for **DPB-n** and they all ended up using only spot instances.

The total EMR cost dropped by 22% to 35% in DAAB-n experiments and by almost 50% in DPB-n. Task capacity price was dropped heavily up to 83% considering task nodes only.

IV. Conclusion

The aim of this paper was to design and develop mechanisms which can minimize the cost for processing deadline aware batch jobs in Amazon EMR by leveraging spot instances. The intuition behind our work is the fact that deadlines can give some slack to a batch job processing, and thus, lower cost can achieved by differing partial execution of the job til spot instances are obtained at lower prices than on-demand instances. Two algorithms: **DAAB** and **DPB** were devised and tested in Amazon EMR with managed Hadoop capability. The algorithms are responsible for varying the task pool capacity of Amazon EMR and issuing appropriated bidding decisions. **DAAB** algorithm focuses on adjusting the number of task instances in order to ensure that the current progress is in pace with the baseline progress. When 90% of the deadline is elapsed, **DAAB** would vary task capacity and continue execution using only on-demand instances. While **DPB** is based on initially bidding for a high number of instance to complete processing earlier. Another notable difference between **DAAB** and **DPB** lies in their different bidding strategies. **DAAB** uses the median of last 10 hours for calculating bid prices. However, in the case of **DPB**, only latest market price combined with the on-demand price is used to calculate the bid price. Comprehensive experiments conduced in Amazon EMR show that both algorithms are able to significantly minimize the processing cost compared to some base-line execution with only on-demand instances while meeting the job deadline.

References

[1] X. He, P. Shenoy, R. Sitaraman, and D. Irwin, “Cutting the cost of hosting online services using cloud spot markets,” in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 207–218.

[2] Y. Song, M. Zafer, and K.-W. Lee, “Optimal bidding in spot instance market,” in *2012 Proceedings of INFOCOM*. IEEE, 2012, pp. 190–198.

[3] I. Menache, O. Shamir, and N. Jain, “On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud,” in *Proc. of USENIX International Conference on Autonomic Computing*, 2014.

[4] S. Subramanya, T. Guo, P. Sharma, D. Irwin, and P. Shenoy, “Spoton: a batch computing service for the spot market,” in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 329–341.

[5] “Amazon emr - amazon web services.” [Online]. Available: http://aws.amazon.com/emr/

[6] “Instance groups - amazon emr.” [Online]. Available: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/InstanceGroups.html

[7] N. Xue, “Automated cloud bursting on a hybrid cloud platform,” *Master Thesis (University of Oslo)*, 2015.

[8] S. Li, S. Hu, S. Wang, L. Su, T. Abdelzaher, I. Gupta, and R. Pace, “Woha: Deadline-aware map-reduce workflow scheduling framework over hadoop clusters,” in *2014 IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2014, pp. 93–103.

[9] J. Gray and T. C. Bressoud, “Towards a mapreduce application performance model,” in *Midstates Conference*, 2012.

[10] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang, “How to bid the cloud,” in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 71–84.

[11] “Elastic compute cloud server & hosting.” [Online]. Available: https://aws.amazon.com/ec2

[12] “Boto3 documentation - boto3 docs 1.4.1 documentation.” [Online]. Available: https://boto3.readthedocs.io/en/latest/

[13] “Amazon elastc mapreduce dimensions and metrics - amazon cloudwatch.” [Online]. Available: http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/emr-metricscollected.html

[14] “Aws webcast - amazon elastic map reduce deep dive and best practices.” [Online]. Available: http://www.slideshare.net/AmazonWebServices/amazon-elastic-map-reduce-deep-dive-and-best-practices

[15] “Amazon ec2 m1.medium - live performance benchmarks - cloudlook.” [Online]. Available: http://www.cloudlook.com/amazon-ec2-m1-medium-instance

[16] “Aws | amazon elastic mapreduce (emr) | pricing.” [Online]. Available: https://aws.amazon.com/elasticmapreduce/pricing/