Stragglers' Detection in Big Data Analytic Systems: The Impact of Heartbeat Arrival

Thomas Lambert, Shadi Ibrahim, Twinkle Jain, David Guyon

Stragglers' Detection in Big Data Analytic Systems: The Impact of Heartbeat Arrival

Thomas Lambert*, Shadi Ibrahim†, Twinkle Jain‡, and David Guyon§
*Université de Lorraine, Inria, LORIA, Nancy, France.
†Inria, Univ. Rennes, CNRS, IRISA, Rennes, France.
‡Northeastern University, Boston, USA.
§Inria, IMT Atlantique, LS2N, Nantes, France.
thomas.lambert@inria.fr, shadi.ibrahim@inria.fr, jain.t@northeastern.edu, davidguyon@ecomail.fr

***Abstract*—Speculative execution can significantly improve the performance of Big Data applications by launching other copies of stragglers (slow tasks). Stragglers detection plays an important role in the effectiveness of speculative execution. The methods employed to detect stragglers use the information extracted from the last received heartbeats which may be outdated when triggering detection. This, in turn, can mislead Big Data analytic systems to make wrong detection with high inaccuracy. To shed the light on this issue, we carry out extensive simulations to identify how heartbeat arrival, task starting times, and detection methods impact the accuracy of stragglers detection in Big Data analytic systems. We reveal that the asynchrony in heartbeat arrivals not only lead to marking normal tasks as stragglers (false positives) but can also result in overlooking real stragglers (false negatives).**

***Index Terms*—MapReduce; Hadoop; Speculation; Stragglers; Heartbeat**

I. Introduction

The past decade has witnessed a tremendous increase in the deployment of Big Data analytic systems in large-scale environments to meet the ever growing size and velocity of data. Nevertheless, a major challenge – when running Big Data applications at large-scale infrastructures – is performance variability. Performance variability in large-scale environments causes stragglers (tasks performing relatively slower than other tasks) which result in a severe degradation in performance. Traces from production clusters at Facebook and Microsoft pointed out that stragglers can be 7-8 X slower than the median task and can increase the average job duration by up to 47% [1]. Big Data analytic systems (Google MapReduce [2], [3], Apache Hadoop [4], Apache Spark [5]) employ speculative execution to mitigate stragglers at large scale. Specifically, new copies of detected stragglers are launched on available machines. Over the past years, there have been several efforts to improve the effectiveness of speculative execution, such as launching speculative copies on lightly loaded machines [6], making speculative execution resource-aware [7], [8], or integrating speculation and job scheduling decisions [9].

Stragglers detection plays an important role in the effectiveness of speculative execution. Thus, many studies have focused on improving stragglers detection. The basic philosophy of existing methods to detect stragglers is to compare (evaluate) tasks, using the information extracted from the last received heartbeats[1]. These methods can broadly be categorized as either *ScoreBased* or *RateBased*. While *ScoreBased* detection methods [2], [4] use the progress score of tasks (i.e., the fraction of the data has been processed) to judge if a task is a straggler or not (e.g., in Hadoop [4], a task is marked as a straggler if its progress score is smaller than the average progress score of all running tasks minus 20%), *RateBased* detection methods [6], [7] mark tasks which are 20% slower than the average progress rate (or tasks which have 20% longer "estimated" elapsed times compared to the mean or median). Recent studies have shown that current detection methods (*ScoreBased* and *RateBased*) achieve very low precision and recall (i.e., the precision of *ScoreBased* detection, used in Hadoop, can be only 12% in heterogeneous infrastructures [10]). Unfortunately, only a few efforts [10], [11], [12] have touched on how to explain the (in)accuracy of detection methods.

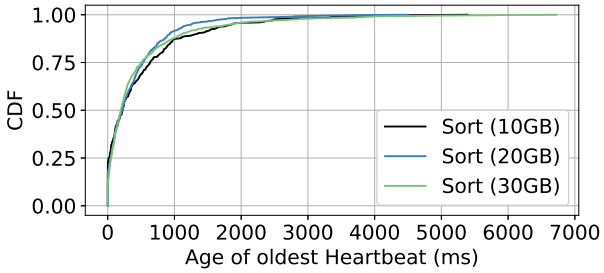


Fig. 1: *CDF of the freshness of heartbeats for Sort applications with different input sizes*

The main contribution of this study is to identify and to quantify the impact of heartbeat arrival on the inaccuracy of stragglers detection in large-scale infrastructures. *To the best of our knowledge, no previous studies have worked on identifying and analyzing the impact of heartbeat arrival on stragglers*

[1]Not to be confused with the heartbeats that report the resource availability (from the NodeManager to the ResourceManager in Yarn) or the ones that report the job status (from the ApplicationMaster to the ResourceManager in Yarn), in this paper, heartbeats refer to the periodic signals that report the progress/status of running tasks: from the tasks to the ApplicationMaster in Yarn (known as HearbeatTask in Spark).

detection. Our study is motivated by two observations. First, having up-to-date information about the running tasks is essential when detecting stragglers. The second observation is that throughout the execution of Big Data applications, the freshness of the information reported by last received heartbeats vary between tasks, depending on the heartbeat window (e.g., $6s$ in Hadoop) and the task starting times. To demonstrate that, we deployed Hadoop 3.2.0 (Yarn) on 23 virtual machines cluster using EnosLib [13]. We use 6 nodes from Ecotype cluster (each machine is equipped with two Intel Xeon E5-2630L v4 10-cores processors, 128 GB of main memory, and 400 GB of SSD) at Nantes site of Grid'5000 [14]. In this cluster, 1 VM (configured with 6 CPU cores and 6 GB of RAM) is running alone on 1 node and acts as ResourceManager. The remaining 22 VMs (each configured with 1 CPU core and 6 GB of RAM) are distributed among the remaining 5 nodes (we deployed 2 VMs, 2 VMs, 4 VMs, 4 VMs, and 10 VMs per node, respectively) and act as NodeManagers and DataNodes. We limit the network bandwidth to 500 Mbps and run 6 applications including three Sort applications with 10, 20, and 30 GB inputs and three Wordcount applications with 10, 20, and 30 GB inputs. We modified Hadoop to log the progress scores and time-stamps of each task whenever a heartbeat arrives and reported the age of the most outdated progress score per second (the age of the heartbeat with the oldest value) for each application. In Fig. 1, we can clearly see that the freshness of heartbeats information for the three Sort applications varies from $0 - 6.7$ seconds.

Unfortunately, current detection methods do not take into account the freshness of heartbeats information. To this end, we have developed a simulator written in Java and implemented two state-of-the-art *ScoreBased* and *RateBased* detection methods used in Hadoop [4] and in Bing [7]. Using synthetic data sets, we carry out extensive simulations over a large number of scenarios to identify how heartbeat arrival, task starting times, and detection methods impact the accuracy of stragglers detection in Big Data analytic systems. We reveal that the asynchrony of heartbeat arrivals (and the resulting stale information) not only result in marking normal tasks as stragglers (false positive) but also lead to the non-detection of real stragglers (false negative).

The rest of this paper is organized as follows. The problem alongside two detection methods are formalized in Section II. The experimental methodology is explained in Section III. Section IV discusses the experimental results. Finally, Section V concludes this study.

II. Model and Detection Methods

A. Model

We present the model to describe the problem and on which we build the simulator for experiments in Sections III and IV.

Let $\mathcal{J}$ be a job (a set of tasks). For each task $T_i \in \mathcal{J}$, we define its duration as d_i and its starting time as s_i. We consider both to be independent from the placement of T_i. We use $d_{\mathcal{J}}$ to denote the average duration of the tasks of $\mathcal{J}$. A task T_i is considered as a *straggler* if and only if $d_i \geq 1.2 \times d_{\mathcal{J}}$, *i.e.*, if its duration is 20% higher than the average duration of all tasks. *Heartbeats* are signals sent by tasks to the master node (e.g., from the tasks to ApplicationMaster) at a fixed interval; we use hw to denote the *heartbeat window* (the default hw is $6s$ in Hadoop and $10s$ in Spark). In addition, heartbeats are also sent when the task starts or ends. Let $st_{i,j}$ be the sending time of the j^{th} heartbeat of T_i. By definition $st_{i,0} = s_i$ and $st_{i,j} = s_i + j \times hw$. We use $j_{end}(i)$ to denote the index of the last heartbeat (end of the task). Then $j_{end}(i) = \lceil d_i/hw \rceil$ and $st_{i,j_{end}(i)} = s_i + d_i$. Each heartbeat has also a *heartbeat latency* $l_{i,j}$ (usually $l_{i,j}$ varies from few hundreds of milliseconds – up to $500ms$ in the experiments in Section I). Thus, a heartbeat is received by the master node at time $rt_{i,j} = st_{i,j} + l_{i,j}$. At time t, we use $j_{last}(t,i)$ to denote the index of the last received heartbeat from task T_i, more precisely $j_{last}(t,i) = \max\{j, rt_{i,j} \leq t\}$. The information transmitted through heartbeats include the *progress scores* of running tasks (for example, the progress score of a map task is the fraction of input data read). To simplify the problem, we assume the computation speed to be constant. Thus, the progress score of a task T_i at time t is $PS_i(t) = (t - s_i)/d_i$. As the master node can only follow the progress of a task through heartbeats, the perceived progress score may differ from the read one – this strongly depends on the heartbeat window. We refer to perceived one as *approximated progress score*. It is donated by $\widetilde{PS_i}(t)$, more precisely, $\widetilde{PS_i}(t) = PS_i(st_{i,j_{last}(t,i)})$.

B. Stragglers Detection Methods

We present two state-of-the-art methods to detect stragglers:

1. ScoreBased: This method relies on the progress scores (precisely, the approximated ones which were received in the last heartbeats) of the running and completed tasks. Specifically, at a time t, the average progress score is computed as:

$$\widetilde{\overline{PS}}(t) = \frac{\sum\limits_{T_i,\ r_{i,0} \leq t} \widetilde{PS_i}(t)}{|\{T_i,\ r_{i,0} \leq t\}|}.$$

Accordingly, *ScoreBased* method marks a task T_i as a straggler if $\widetilde{PS_i}(t) \leq \widetilde{\overline{PS}}(t) - 0.2$ [4].

2. RateBased: In order to be less sensitive to delayed tasks (task with skewed starting times), some detection methods use the notion of *progress rate*, that is the rate at which progress score increases. More precisely, the progress rate of a task T_i at time t is $PR_i(t) = PS_i(t)/(t - s_i)$. However, $PS_i(t)$ represents the approximate progress score, thus the perceived progress rate is an approximate one and can be referred to as *approximated progress rate*. It is denoted by $\widetilde{PR_i}(t)$ and is computed as $\widetilde{PR_i}(t) = \widetilde{PS_i}(t)/(t - s_i)$.

Usually, the progress rate based method (will refer to as *RateBased*) computes the *expected duration* $ETD_i(t)$ of a task T_i at time t, more precisely:

$$\widetilde{ETD_i}(t) = t - s_i + (1 - \widetilde{PS_i}(t))/\widetilde{PR_i}(t).$$

A task T_i is considered as a straggler if and only if $\widetilde{ETD_i}(t) \geq 1.2 \times \widetilde{\overline{ETD}}(t)$. $\widetilde{\overline{ETD}}(t)$ denotes the average expected duration for all started tasks. Similar method is used by LATE [6] and Mantri (with the factor 1.5) [7].

III. Experimental Setup

We build a Java simulator based on the model described in the previous section. We describe the different parameters and the synthetic jobs sets we use in our experiments below.

A. Synthetic Jobs

1. Durations: We generate random task durations. More precisely, we follow uniform distribution with range $[d_{min}, d_{max}]$. d_{min} and d_{max} are fixed from the expected average d_{avg}. More precisely $d_{min} = (1-f) \times d_{avg}$ and $d_{max} = (1+f) \times d_{avg}$. We use the values $10s$, $20s$, $50s$ and $100s$ for d_{avg} and for f we use 0 (homogeneous, all tasks with same duration), 0.1 (slightly heterogeneous, no stragglers), and 0.25 (more heterogeneous durations with some stragglers). In the last case, the amount of stragglers is expected to be 10%. Indeed, with a uniform draw on $[(1-f)d_{avg}, (1+f)d_{avg}]$, the probability of being in $[1.2d_{avg}, (1+f)d_{avg}]$ is $(1+f-1.2)/2f = 0.1$ when $f = 0.25$. We will use "$d_{min} - d_{max}$" to distinguish the different sets of jobs (e.g., $18-22s$ or $75-125s$) in Section IV.

2. Starting Times: For starting times, we use two main possible modes. In the first one, all starting times are equal to 0. We refer to this mode as UniformStarts. In the second one, we use ending times from UniformStarts as starting times, this mode is referred to as SkewedStarts.

B. Hosts

First, we propose two modes for heartbeat latencies, one without (NoLatency) where all heartbeats arrive as soon as they are sent, and one with (WithLatency) where latency follows a *Pareto Distribution* of scale $1s$ and shape 5 (our objective is to mimic the heartbeat latency in real Hadoop cluster – up to $500ms$ in our experiment – and to have few "extreme" events). In 90% of the send events, latency is a few dozen to a few hundreds of milliseconds, but for the last 10%, latency is above $1s$ and more (which is then above the stragglers detection window, *i.e.*, $1s$). A CDF of latencies is shown in Fig. 2 (1 million draw, note that we forced the latency to stay below $2s$ by redrawing any higher value). We always consider the number of hosts to be equal to the number of tasks in the simulated job. For each configuration, if the simulation uses randomness (i.e., WithLatency or for SkewedStarts in synthetic jobs) then we run the experiments 50 times.

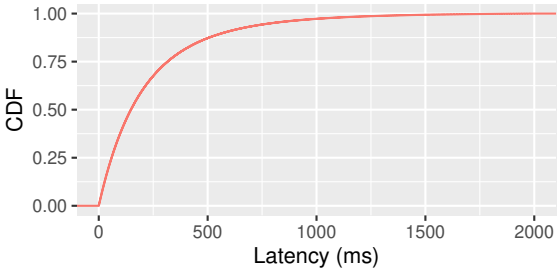


Fig. 2: *CDF for the latency of Heartbeat arrivals*

C. Other Parameters

The other two parameters we are tuning are the *heartbeat window* (hw) and the *speculative lag*, *i.e.*, the time since the starting of a job after which the stragglers detection starts. We use a Hadoop-like configuration, *i.e.*, with $6s$ heartbeat window. In every case, stragglers are detected every second after the speculative lag (i.e., detection starts as soon as one task is finished). In our experiments, the detection considers tasks with at least one received heartbeat.

D. Metrics

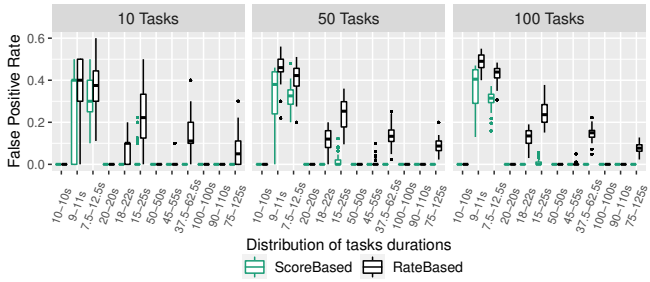
A task is said to be *detected* if there is a moment during the execution where the detection method marked it as a straggler. For example, if for the task T_i there is a time t after the speculative lag where $\widetilde{PS}_i(t) \leq \widetilde{\widetilde{PS}}(t) - 0.2$, then this task is detected by the *ScoreBased* detection method. Note that a task can be detected at time t, but might not be detected anymore later (because of more or less up-to-date information). In such a case, the task is still considered as detected in the results (the task has been reported at least once as a straggler). If a detected task is a straggler (its duration exceeds 1.2 times the average duration of all tasks), then it is a *true positive*. If a detected task is not a straggler, it is then a *false positive*. Finally, a straggler that is never detected is a *false negative*.

IV. Experimental Results

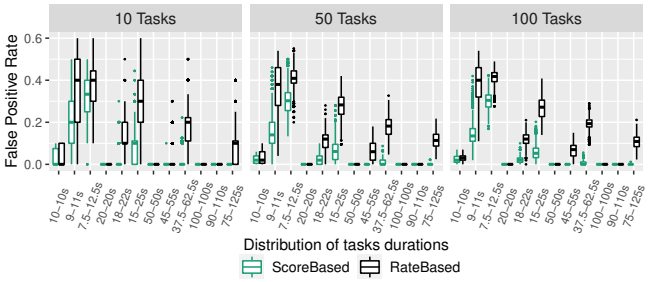
We present all our results with box plots. As a reminder, the horizontal line inside the box is the median of the values for a given setup, frontiers of the box are the limits of first and last quartiles, and the vertical bars at the end are the limits of first and last deciles. Outliers are represented by dots. We will precise the average value in the text when needed.

The false positive rates (respectively, the false negative rates) are depicted in Fig. 3 (respectively, in Fig. 4). In all sub-figures of Fig. 3 and 4, the left figure depicts the case where heartbeats have no latency (they are received as soon as they are sent), the right figure shows the case where heartbeat latency is added (up to $2s$, but most of the time less than $1s$ which is equivalent to the detection window interval). We normalize false positives and false negatives using the number of normal tasks (i.e., all non-stragglers including the false positives) and the number of real stragglers. We refer to them as *false positive rate* and *false negative rate*, respectively. In the case of false negatives, we only show results for duration ranges where there are stragglers (*i.e.*, $7.5-12.5s$, $15-25s$, $37.5-62.5s$, and $75-125s$).

1. False Positives *(Figure 3)*: For *Homogeneous task durations*, as expected, no false positives are detected under NoLatency mode. However, adding latency leads to a few false positives when task durations are set to $10s$ under both *ScoreBased* and *RateBased* (e.g., false positive rates when the job size is set to 10 tasks are 4% and 2%, respectively).
For *heterogeneous task durations*, more false positives appear under both *ScoreBased* and *RateBased* detection methods. In the case of *ScoreBased*, for jobs with $10s$ average task

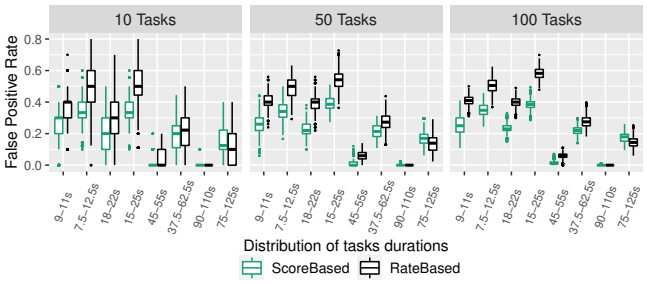


(a) UNIFORMSTARTS-NOLATENCY



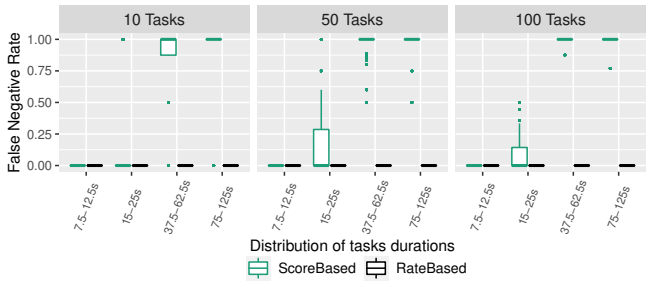
(b) UNIFORMSTARTS-WITHLATENCY

(c) SKEWEDSTARTS-NOLATENCY

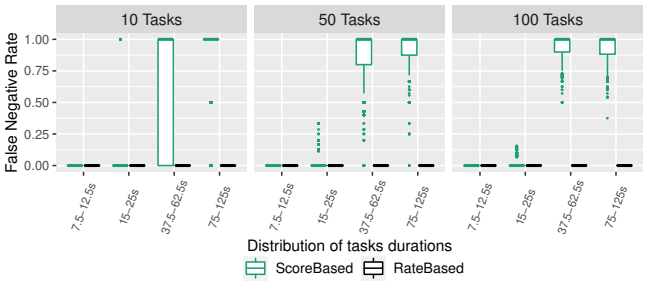


(d) SKEWEDSTARTS-WITHLATENCY

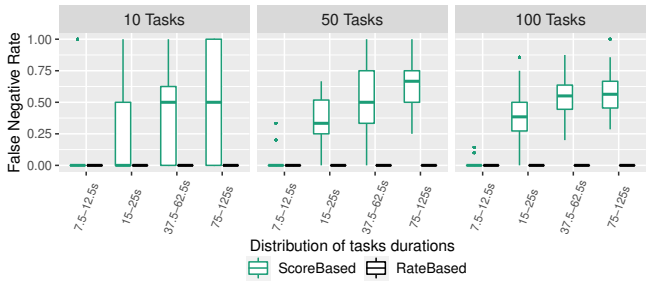
Fig. 3: *False positive rates in Hadoop-like configuration on synthetic jobs*



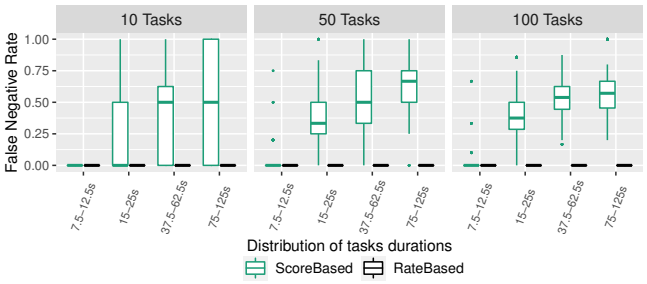
(a) UNIFORMSTARTS-NOLATENCY



(b) UNIFORMSTARTS-WITHLATENCY



(c) SKEWEDSTARTS-NOLATENCY



(d) SKEWEDSTARTS-WITHLATENCY

Fig. 4: *False negative rates in Hadoop-like configuration on synthetic jobs*

duration, the false positive rate is of the same amplitude for both UNIFORMSTARTS and SKEWEDSTARTS modes. For $7.5 - 12.5s$ task durations range and 50-task jobs, the average false positive rates under UNIFORMSTARTS-NOLATENCY mode and SKEWEDSTARTS-NOLATENCY mode are 32.02% and 34.11%, respectively. However, when looking at medium and long task durations ($20s$ and more), false positives mainly appear under SKEWEDSTARTS (compared to very few false positives under UNIFORMSTARTS mode). Under UNIFORMSTARTS mode, the highest false positive rate is for $15 - 25s$ task durations range and 50-task jobs. We observe that the average false positive rate is 1.51% (median 0%) under NOLATENCY mode and is 6.50% (median 5.44%) under WITHLATENCY mode. Comparatively, for the same task durations range and job size, the false positive rate is on average 38.08% under SKEWEDSTARTS and NOLATENCY mode.

In the case of *RateBased* detection method, the number of false positives is larger in general. For example, for the $15-25s$ task durations range and 50-task jobs, the average false positive rate under SKEWEDSTARTS mode is 53.27% under NOLATENCY mode (median 53.26%) and 54.07% under WITHLATENCY mode (median 54.17%). This indicates that for more than half of the runs, more than 50% of the non-stragglers are tagged as stragglers at least once during the run (however, it is important to note that they are not handled as stragglers at the same time, some of them may lose this tag and may have it back later). This issue occurs under UNIFORMSTARTS mode too, but with smaller amplitude (the average false positive rate is 24.31% for the same task durations range and job size).

Regardless of the detection method, we find that the false positive rate is impacted by the average duration of tasks. In general, the false positive rate decreases (sometimes significantly) when the average task duration increases. For example, for 50-task jobs and under NOLATENCY mode, the false positive rate is reduced from 53.27% to 13.27% on average when average task duration increases from $20s$ to $100s$. This is more obvious with *RateBased* than *ScoreBased*. The size of the job seems to have little impact. The presence of heartbeats latency increases the size of the box plots and the outliers count but does not alter the general trend.

2. False Negatives *(Figure 4)*: In all cases, *RateBased* does not result in false negatives. On the other hand, *ScoreBased* results in false negatives. This method does not detect any stragglers for jobs with average task durations of $50s$ and $100s$ under UNIFORMSTARTS-NOLATENCY mode (rates inferior to 1 are outliers). For jobs with $20s$ average task duration, *ScoreBased* detects most of the stragglers (on average, only 6.43% of stragglers are not detected under NOLATENCY mode for 50-task jobs). A surprising result is that heartbeat latency seems to improve (to decrease) the number of missed stragglers for *ScoreBased* detection method. Maybe as surprisingly, the false negative rate decreases under SKEWEDSTARTS mode.

V. CONCLUSION

We study how heartbeat arrivals and using outdated information when detecting stragglers impact the accuracy of stragglers detection. Our findings can be summarized as follows: (1) Not considering the freshness of information during stragglers detection can lead to a large amount of false positives (mainly noticeable with *RateBased*). (2) This phenomenon is greatly amplified when the heartbeats are asynchronous, either because of skewed starting times or because of late heartbeats (due to latency). (3) False negative is mainly an issue with *ScoreBased* methods. As future work, we plan to evaluate detection methods when considering the time-stamp of heartbeats (e.g., they use the time-stamps of heartbeats to estimate the progress scores/rates when the detection is triggered) using synthetic and production data sets.

ACKNOWLEDGMENTS

This work is supported by the ANR KerStream project (ANR-16-CE25-0014-01). The work of the third author was partially supported by National Science Foundation Grant OAC-1740218. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several Universities as well as other organizations (see http://www.grid5000.fr/).

REFERENCES

[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'13. USA: USENIX Association, 2013, p. 185–198.

[2] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, jan 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[3] H. Jin, S. Ibrahim, L. Qi, H. Cao, S. Wu, and X. Shi, *The MapReduce Programming Model and Implementations*. John Wiley & Sons, Ltd, ch. 14, pp. 373–390.

[4] The Apache Hadoop Project. (2022). [Online]. Available: http://hadoop.apache.org

[5] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. USA: USENIX Association, 2010, p. 10.

[6] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08. USA: USENIX Association, 2008, p. 29–42.

[7] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris, "Reining in the outliers in map-reduce clusters using mantri," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. USA: USENIX Association, 2010, p. 265–278.

[8] A. C. Zhou, T.-D. Phan, S. Ibrahim, and B. He, "Energy-efficient speculative execution using advanced reservation for heterogeneous clusters," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3225058.3225084

[9] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu, "Hopper: Decentralized speculation-aware cluster scheduling at scale," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 379–392. [Online]. Available: https://doi.org/10.1145/2785956.2787481

[10] T.-D. Phan, G. Pallez, S. Ibrahim, and P. Raghavan, "A new framework for evaluating straggler detection mechanisms in mapreduce," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 3, sep 2019. [Online]. Available: https://doi.org/10.1145/3328740

[11] T.-D. Phan, S. Ibrahim, A. C. Zhou, G. Aupy, and G. Antoniu, "Energy-driven straggler mitigation in mapreduce," in *Euro-Par 2017: Parallel Processing*, F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, Eds. Cham: Springer International Publishing, 2017, pp. 385–398.

[12] T.-D. Phan, S. Ibrahim, G. Antoniu, and L. Bougé, "On understanding the energy impact of speculative execution in hadoop," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*, 2015, pp. 396–403.

[13] R.-A. Cherrueau, M. Delavergne, A. van Kempen, A. Lebre, D. Pertin, J. R. Balderrama, A. Simonet, and M. Simonin, "Enoslib: A library for experiment-driven research in distributed computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1464–1477, 2022.

[14] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science. Springer International Publishing, 2013, vol. 367, pp. 3–20.