An effective straggler tolerant scheme in Big Data processing systems using Machine Learning

Shyam Deshmukh, K. Thirupathi Rao, B. Thirumala Rao, Vaibhav Pawar

Abstract:* The priority of major vendors of data storage is always to the faster job completion time and efficient resource utilization in cloud envi-ronment. Slow running or poor performing cluster nodes continue to be a major hurdle for faster job execution in cloud environment. Var-ious existing mitigation techniques which neglects these slow processing nodes i.e. stragglers and try to optimize resource utilization as well as response time are discussed with their limitations. In this paper, the aim is to build a blacklisting-enabled machine learning-based straggler tolerant technique using Apache spark framework which identifies straggler in a cluster. This straggler tolerant scheme act as a decision support system for the scheduler which predicts and avoid the task assignment to the straggler node regardless of internal and external causes of Straggler. Decision tree is constructed using job utilization and time execution metrics. Various experiments were carried out using default apache spark scheduler , blacklisting-enabled apache scheduler and blacklisting-enabled machine learning based scheduler using input workloads such as Word Count and Tera Sort. The results shows that our approach reduces the Job Completion Time of task by 19% and gives better utilization of resources in cloud environment.

***Keywords:* Apache Spark; Cloud Environment; Distributed Systems; Machine Learning; Straggler.**

I. INTRODUCTION

In distributed processing framework mechanism, master divide the data into chunks and distribute it amongst all slaves wherein it takes data intensive job and copies it to slaves, which are then executed in parallel on commodity clusters to accomplish faster job completion. A natural consequence of such a parallel processing model is the condition called stragglers; i.e. slow processing nodes, potentially delay the overall job completion. Straggler tasks remain to be a major barrier in attaining faster completion of data parallel and computation intensive applications running on modern big-data processing frameworks in cloud environment. For this purpose, we need straggler tolerant schemes which supports scheduling algorithm for mitigating the effect of stragglers. While detecting the straggler nodes, it is necessary to focus on the factors causing a node to be a straggler node. To improve the performance of straggler mitigation as compared to that of existing techniques we have implemented a machine-learning based mechanism to dynamically compute whether a processing node is a straggler or not on Apache Spark. Apache Spark is designed to be fault tolerant to a range of classes of faults. To solve straggler problem more efficiently, apache spark scheduler is designed with a blacklisting mechanism to improve the scheduler's capability to keep track of failures. After a certain number of failures, the spark scheduler will not allocate the work any more on the node as it is blacklisted.

The main objective here is to design a straggler tolerant scheme for scheduler which avoids scheduling the jobs to straggler nodes using Spark framework in cloud environment which results in faster job Completion for input workloads like WordCount and TeraSort. The Apache Spark framework blacklist the node on the basis of few parameters which it observes. We have been monitoring this blacklisted nodes, recalculating it's performance dynamically and predicting the node as straggler using the decision tree. The construction of decision tree is based on various utilization and execution time parameters. This information of decision tree is then provided to the scheduler at the master node in the spark cluster which then blacklist the straggler node for certain time interval which has been calculated dynamically and allows non-straggler nodes to execute the tasks. The proposed scheme aims to not completely blacklist the node; but after the dynamic blacklist-timeout is completed, the node again can participate in execution of jobs.

2. LITERATURE SURVEY

Slow running tasks are impediment in faster execution of applications on modern data processing frameworks. The problem with big data framework is that, while the tasks are assigned to various nodes, few of them may slowdown the execution of whole program. There are various ways through which the mitigation has been approached like using Speculative Execution (SE) [1], Blacklisting [2] and Machine Learning Algorithms [3]. On the basis of their methods to mitigate, the stragglers are classified as passive methods used in speculative execution, blacklisting and active methods like cloning approaches, proactive approaches. In SE based algorithms, when and where to launch the speculative task execution is very critical problem. There are situations where the tasks executing on executors, compete for the resources within the node. This creates a race condition among tasks within the node for resources. In order to avoid this situation, a technique that replicates the instances, so that no two tasks will compete for same resource at a time called as cloning which reduces the job completion time by 34% [4]. Some of SE based approaches like LATE, SAMR, ESAMR, MANTRI, Dolly

Revised Manuscript Received on March 10, 2019.

Shyam Deshmukh, Research Scholar, KL Deemed to be university AP, India(E-Mail: sbdeshmukh@pict.edu)
K. Thirupathi Rao, KL Deemed to be university, AP, India
B. Thirumala Rao, KL Deemed to be university, AP, India
Vaibhav Pawar, SP Pune University, AP, India

[4,5,6,7,8] etc., clones the speculative copies with some variations like updating weights through execution history, but consumes more resources. One of the speculative strategies like Maximum Cost Performance (MCP) [9] attempted to improve the speculative execution by accurate and smart identification of stragglers through Statistics like moving average task execution time.

In Apache spark, majorly the straggler mitigation is performed by node blacklisting [2], but this will lead to under utilization of that resource entirely without proper detection of it. So, we have come up with a better strategy where we defined the level of confidence of task execution on all the nodes and use it to assign the tasks. So, on a machine where more stragglers are seen, a very few tasks will be allocated for execution. So small amount of work can be allocated to the machine instead of penalizing the machine by blacklisting it. Self-adaptive map-reduce scheduling algorithm (SAMR) makes use of historical information stored on each node in cluster to predict the progress score more accurately than LATE, but, it does not take into consideration that distinct job types may not have similar weights for map stages and reduce stages in Hadoop. Each of these existing approaches addresses small subset of possible problems. Many straggler mitigation techniques like Wrangler [3], Multi-task Learning [10], etc. have started using machine learning approaches and proved to be beneficial to a large extent. Wrangler, a predictive model with confidence level is developed for each node to predict the behavior of straggler in scheduler may lead to huge improvements in execution time of jobs. A training model for each new node is to be developed with limited training data leads to lower quality models which is addressed using Multi task learning formulation. The main focus was to capture structure of the tasks which helped in reducing completion time of jobs by approximately 59%. The proposed scheme uses the existing Apache spark framework of blacklisting and dynamically sets it's timeout depending on the few parameters of each node which improves the blacklisting mechanism.

3. PROPOSED SYSTEM FOR STRAGGLER MITIGATION

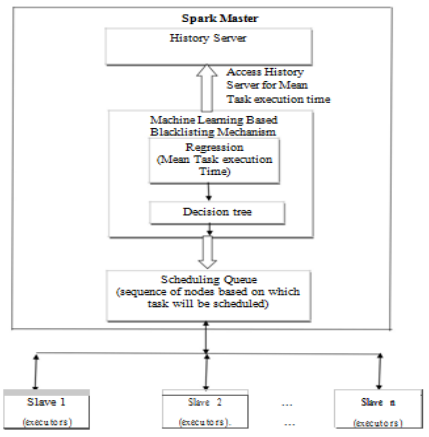
Spark Master
History Server
Access History Server for Mean Task execution time
Machine Learning Based Blacklisting Mechanism
Regression (Mean Task execution Time)
Decision tree
Scheduling Queue (sequence of nodes based on which task will be scheduled)
Slave 1 (executors)
Slave 2 (executors)
...
Slave n (executors)

Fig. 1: Proposed System for Straggler Mitigation

The proposed straggler mitigation uses blacklisting-enabled Decision tree based machine learning algorithm. Execution History of WordCount and TeraSort jobs is logged in the Spark history server. The set of tuples specifying the parameters that are considered to decide whether the node Ni is straggler or not includes parameters like CPU utilization, memory utilization, disk utilization, task execution time, and Location of a node. These tuples are then provided to construct the decision tree, based on this the Predictive model is generated which indeed predicts the job completion time. The Apache spark scheduler consults Blacklisting-enabled machine learning based mechanism to determine which node is straggler. While identifying straggler node, it is necessary to see the node performance metrics of the node as well as considering the factors considered by Spark for blacklisting nodes. The history of a node is collected in the first run whenever some tasks are executed after which the scheduler learns the behavior of the node. The blacklisting mechanism keeps up the record of hosts and executors that have encountered prior failures. Whenever any task get fails on particular executor, the blacklist keeps track which executor and host was related with that failure. Scheduler will not schedule the task or work on that particular node or executor, after keeping track of failures for certain threshold. Sometimes, the scheduler may kill a executor which is failing, or even may kill each and every single failing executors on a particular node. The feature of blacklisting permit for configuration of the number of retries for a task, number of failures before the resource is unavailable for scheduling, total time amount for which blacklist entry is acceptable or valid, and whether to attempt to kill the blacklisted resources or not.

Pseudo Algorithm:

Let N = (Ni : i = **1, … , n** nodes) : set of the nodes in spark cluster.

Let N_{Bi} be the set of nodes which are blacklisted.

Let PARAM(Ni) be the set of tuples specifying the parameters that are considered to decide whether the node Ni is straggler or not. The tuple consist of following normalized parameters:

{ CPU_USAGE,MEMORY_USAGE, DISK_USAGE, TASK_EXECUTION_TIME, LATENCY_TO_GET_THE_RESOURCE, STRAGGLER_FREQUENCY, maxTaskAttemptsPerExecutor, maxTaskAttemptsPerNode, maxFailedTasksPerExecutor, maxFailedTasksPerNode, maxFailedExecutorsPerNode.}

Where,

$$\text{Straggler Frequency} = \frac{\text{No. of times the node is blacklisted}}{\text{Overall Job Completion Time}}$$

Also, the values in defaults-conf file in Apache spark home directory are being set as per below for better results:

CONFIGURATION_VARAIBLE (Ni) = {maxTaskAttemptsPerExecutor, maxTaskAttemptsPerNode, maxFailedTasksPerExecutor, maxFailedTasksPerNode , maxFailedExecutorsPerNode
} =2

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

Let Sq be the Scheduling queue consisting the sequence of nodes which will be used to execute the tasks.

I) BLACKLIST_OF_NODES (Ni, Spark_Conf(Ni))

```
1:      Spark_Conf(Ni)  PARAM (Ni)
 // Let us Construct a decision tree as per below:
MEAN_PARAM(Ni) = $\sum_{i=1}^{n} \frac{PARAM(Ni)}{Ni}$
```

II) DECISION_TREE (PARAM(NI), EAN_PARAM(Ni), CONF_VARAIBLE(Ni))

```
1: Let OLD_MEAN_PARAM (Ni) = MEAN_PARAM (Ni)
NEW_MEAN_PARAM = $\sum_{i=1}^{n} \frac{PARAM(Ni)}{Ni}$
2: Let OLD_CONF_VARIABLE (Ni) = CONF_VARAIBLE (Ni)
3: If ( NEW_MEAN_PARAM != OLD_MEAN_PARAM (Ni)
{
$N_{Bi!}$ = Ni // $N_{Bi}$ is Straggler-node
}
else
{
  Ni Sq[rear]  // Node Ni is to be placed at end of queue.
}
```

III) FAIR_SHARE_SCHEDULING (Sq).

```
DRIVER_PROGRAM(INPUT_DATA,INPUT_PROGRAM)
{
BLACKLIST_OF_NODES (Ni,Spark_Conf (Ni));
DECISION_TREE (PARAM(NI), MEAN_PARAM (Ni), CONF_VARAIBLE(Ni));
FAIR_SHARE_SCHEDULING (Sq);
Return JCT;
}
```

4. EXPERIMENTAL WORK

4.1 Cluster Setup

The 4 node Spark cluster consists of one master node with better configuration than three slave nodes. Using wireless network, the nodes have been interconnected. In this multi-node setup, all systems use RHEL 7 as an operating system, Java 8 or JDK 1.8 and Apache Spark version 2.2.1 for the performance. We have built Apache Spark framework of the heterogeneous nodes or hosts for estimating the proposed solution in discovering nodes which are stragglers.

Table 1: Spark Heterogeneous Cluster Setup

Nodes	Node Configuration	Software
Master	Intel-Core CPU i5 - 420U @ 2.70 GHz, RAM 8 GB , HDD 1 TB.	Operating System: RedHat Ent. Linux7. Languages: Scala 2.11, Java 8.
Slave 1	Intel-Core CPU i3-3110M @ 2.40 GHz, RAM 4 GB, HDD 500 GB.	
Slave 2	Intel-Core CPU i3-3110M @ 2.40 GHz, RAM 2 GB, HDD 500 GB.	Framework: Apache Spark 2.2.1. Workload: WordCount, TeraSort
Slave 3	Intel-Core CPU i3-3110M @ 1.60 GHz, RAM 2 GB, HDD 500 GB.	

4.2 Workloads

Presently, Hadoop and Spark are used equally for both iterative and batch processing. The selection of two representative benchmark applications: WordCount and TeraSort, are the representative workloads of several applications used in real-life. WordCount workload counts frequent occurrence of words from the file containing textual data. WordCount is very simple metric used in measuring the quality through counting total number of occurrences of every word. It also is one of the good fit in evaluating an aggregation component in every framework, because Spark uses a map-side combiner to trim down intermediate data. First benchmark workload i.e. wordcount is evaluated using nodes with fixed number and different datasets of increased size. The dataset has 4 different sizes like 50MB, 100MB, 150 MB, 200 MB. TeraSort is popular benchmark sorting algorithm which is suitable to measure an I/O performance as well as communication performance of Spark engine. This TeraSort benchmark workload is evaluated using nodes with fixed number and different datasets of increased size. The input file has 4 different sizes like 50MB, 100MB, 150 MB, 200 MB. There are 3 experiments carried out which requires input considered as mentioned above. In first experiment, the comparison is being done between default Spark Scheduler and Blacklisting-enabled scheduler, and noted the Job Completion Time (JCT) for each of these workloads. In second, the comparison is being done between default spark scheduler and Blacklisting-enabled machine learning based scheduler, and the JCT for each of these workloads is observed. Last experiment does the comparison between Blacklisting-enabled scheduler and Blacklisting-enabled machine learning based scheduler and the JCT for each of these workloads is observed. The analysis of the result of all these experiments is accomplished by comparing their job completion time. The blacklisting mechanism in Apache Spark will be enabled through configuration keys as mentioned below and are being used for comparison during decision tree construction:

```
spark.blacklist.enabled
spark.blacklist.task.maxTaskAttemptsPerExecutor
spark.blacklist.task.maxTaskAttemptsPerNode
spark.blacklist.application.maxFailedTasksPerExecutor
spark.blacklist.stage.maxFailedTasksPerExecutor
spark.blacklist.application.maxFailedExecutorsPerNode
spark.blacklist.stage.maxFailedExecutorsPerNode
spark.blacklist.timeout
spark.blacklist.killBlacklistedExecutors.
```

5. RESULT ANALYSIS

As mentioned in the experimental work, the outcome of all the experiments and its analysis is summarized below:

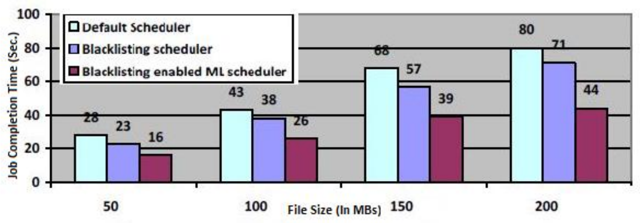

Fig. 2: Iteration on Default Scheduler vs. Blacklist-enabled Scheduler Vs. Blacklisting-enabled machine learning based Scheduler (WordCount)

The Fig. 2 graph above shows WordCount job execution time with Default Scheduler vs. Blacklist-enabled Scheduler and Blacklist- enabled machine learning based scheduler. It is observed that the overall completion time by the job (JCT in Sec.) is reduced by 28% on an average when Blacklisting-enabled machine learning scheduler is evaluated against Default Spark Scheduler Engine and overall job completion time (in seconds) is reduced by 11% on an average when Blacklisting enabled scheduler is compared with Default Scheduler. It is noticed that, when input data size is small there is a reduction in completion time of job but, it rises significantly with the increase in dataset size. Four iterations are carried out on different input data sizes and the Job Completion Time is observed and the averages of the iterations for each input datasets are shown in the graph above.

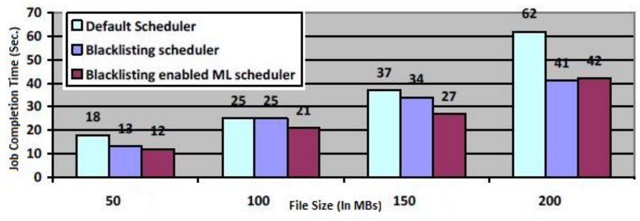

Fig. 3: Default Scheduler Vs. Blacklist-enabled Scheduler vs. Blacklisting-enabled machine learning based Scheduler (TeraSort)

In Fig. 3 graph above shows average of 4 iterations Terasort job execution time with Default Scheduler Terasort Vs. Blacklist-enabled Scheduler and Blacklist-enabled machine learning scheduler. It is seen and noted that overall completion time of job (JCT in Sec.) is reduced by 19% on an average when Blacklisting-enabled machine learning scheduler is evaluated against Default spark scheduler engine and overall job completion time (in second) is reduced by 13% on an average when Blacklisting-enabled scheduler is compared with Default Scheduler. It is noticed that, when input data size is small there is a reduction in completion time of job but, it rises significantly with the increase in dataset size. The 4 iterations are carried out on different input data sizes and the Job Completion Time is Observed and the averages of the iterations for each input data size are shown in the graph above.

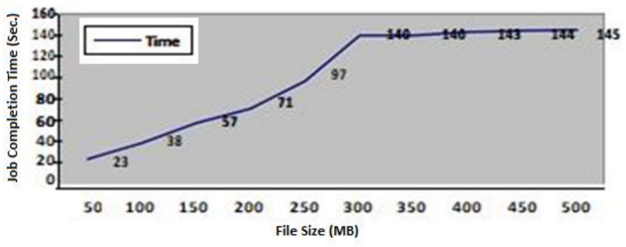

Fig. 4: JCT vs. File Size (WordCount)

The Fig. 4 graph shows the result for WordCount Job being ran with different file sizes. From the graph it is evident that after of size of approximately 500 MB the job completion time doesn't change drastically and remains constant nearby which is nearly 140 sec. and hence this is the value of convergence.

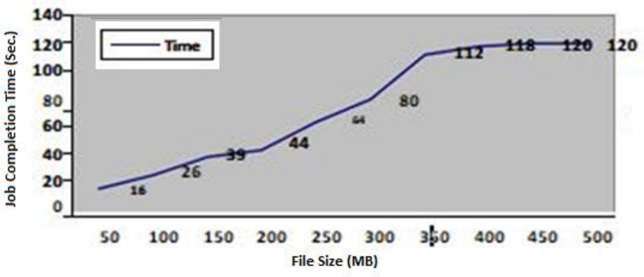

Fig. 5: JCT vs. File Size (TeraSort)

The graph shows the result for TeraSort Job being ran with different file sizes. From the graph it is evident that after of size of approximately 500 MB the job completion time doesn't change drastically and remains approximately constant which is nearly 120 sec. and hence this is the value of convergence.

6. CONCLUSION

In this work, blacklisting-enabled machine learning based straggler mitigation technique is implemented. The decision tree is generated for each node, to identify the straggler node using node parameters and job parameters. For straggler mitigation, the blacklisted nodes are re-examined by allowing them to participate in scheduling. The job completion time of input workload is compared using default Spark Framework and blacklisting-enabled machine learning based Spark Framework. The comparative result shows that there is an improvement in Job Completion Time of approx. 28% for WordCount and 19% for TeraSort using blacklisting enabled Machine Learning based scheduler over default Spark Scheduler. Thus, it is concluded that proposed approach effectively identified and mitigate the straggler effect which ultimately optimizes the resource utilization. In future, by considering few more parameters from spark logging mechanism and workload characteristics, the performance and throughput of the Spark Cluster can be improved.

REFERENCES

1. J. Dean , S. Ghemawat, "MapReduce: simplified data processing on large clusters", Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, Vol.6, (2004), pp.10-10. https://dl.acm.org/citation.cfm?id=1251264.
2. J. Saltren, "Blacklisting in Apache Spark", Cloudera Engineering Blog. 2017 [Online]. Available: https://blog.cloudera.com/blog/2017/04/blacklisting-in-apache-spark/. [Accessed: 29-Jul - 2018].
3. N. Yadwadkar, G. Ananthanarayanan, R. Katz, "Wrangler: Predictable and Faster Jobs using Fewer Resources", Proceedings of the ACM Symposium on Cloud Computing, (2014), pp. 1-14, https://dl.acm.org/citation.cfm?doid=2670979.2671005.
4. G. Ananthanarayanan, A. Ghodsi, S. Shenker, I, Stoica, "Effective straggler mitigation: attack of the clones", Proceedings of 10th USENIX conference on Networked Systems Design and Implementation, (2013), pp.185-198.http://dl.acm.org/citation.cfm?id=2482626. 2482645.
5. M. Zaharia, A. Konwinski, A. Joseph, R. Katz, I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments", Proceedings of the 8th USENIX conference on Operating systems design and implementation, (2008), pp.29-42, https://dl.acm.org/citation.cfm ?id=1855744.
6. Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "SAMR: A self adaptive mapreduce scheduling algorithm in heterogeneous environment", Proceedings of the 10th IEEE International Conference on Computer and Information Technology, (2010), pp. 2736–2743. https://ieeexplore.ieee.org/document/5578538.
7. X. Sun , Chen He , Ying Lu, "ESAMR: An Enhanced Self-Adaptive MapReduce Scheduling Algorithm", Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems,(2012), pp. 148-155, https://ieeexplore.ieee.org/document/6413702.
8. G. Ananthanarayanan, S. Kandula , A. Greenberg , et.al,, "Reining in the outliers in map-reduce clusters using Mantri", Proceedings of the 9th USENIX conference on Operating systems design and implementation, (2010), pp.265-278, https://dl.acm.org/citation.cfm?id=1924962.
9. Chen Q, Liu C, Xiao Z, "Improving MapReduce performance using smart speculative execution strategy", IEEE Transactions on Computers, Vol.63, No.4, (2014), pp. 954–967, available online: https://ieeexplore.ieee.org/document/6419699, last visit:17-01-2018.
10. N. J. Yadwadkar, B, Hariharan, et. al., "Multi-task Learning for Straggler Avoiding Predictive Job Scheduling", The Journal of Machine Learning Research, Vol.17, No.1, (2016), pp. 3692--3728, available online: http://dl.acm.org/citation.cfm?id=2946645.3007059, last visit: 12-09-2017.
11. J. Dhok, V.Varma, "Using Pattern Classification for Task Assignment in MapReduce", Proc. 10th IEEE/ACM Int. Conference CCGrid, (2010), https://researchweb.iiit.ac.in/~jaideep/learning-scheduler.pdf.
12. L. Wang, R. Ren, et.al., "Characterization and architectural implications of big data workloads", 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), (2016), pp. 145-146, https://ieeexplore.ieee.org/document/7482083.
13. M. Zaharia, A. Konwinski,et.al., "Improving MapReduce performance in heterogeneous environments", Proceedings of the 8th USENIX conference on Operating systems design and implementation, (2008), pp.29-42. https://dl.acm.org/citation.cfm?id=1855744.
14. W. Dai, M. Bassiouni, "An improved task assignment scheme for hadoop running in the clouds", Journal of Cloud Computing: Advances, Systems and Applications, vol. 2, No.1, (2013), pp. 23, available online: https://doi.org/10.1186/2192-113X-2-23, last visit:21-07-2018.
15. W. Dai, I. Ibrahim and M. Bassiouni, "A new replica placement policy for Hadoop Distributed File System", Proceedings of 2016 IEEE 2nd International Conference on High Performance and Smart Computing, (2016), pp. 262-267, https://ieeexplore.ieee.org/document/7796164.
16. A. Rosà, L.Chen, W.Binder, "Understanding the Dark Side of Big Data Clusters: an Analysis beyond Failures", Proceedings of 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,(2015),pp.207-218.https://ieeexplore.ieee.org/document72 66851.
17. Z. Liu, Q. Zhang, R. Ahmed, R. Boutaba, Y. Liu, Z. Gong, "Dynamic Resource Allocation for MapReduce with Partitioning Skew", IEEE Transactions on Computers, Vol.65, No. 11, (2014), pp. 3304-3317, available online: https://ieeexplore.ieee.org/document/7415958, last visit: 21-07-2018.
18. Z. Jia, J. Zhan, L. Wang, R. Han, S. A. McKee, Q. Yang, C. Luo, J. Li, "Characterizing and subsetting big data workloads", IEEE International Symposium on Workload Characterization (IISWC), (2014), pp. 191-201. https://arxiv.org/pdf/1506.07742.pdf.