Received: March 15, 2018

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

SAGS: A SLA-Aware Green Scheduling in Heterogeneous Cloud Using Hadoop YARN

Yadaiah Balagoni [1]* **Ramisetty Rajeswara Rao [2]**

[1]*Department of Computer Science Engineering,*
Mahatma Gandhi Institute of Technology, Hyderabad, India
[2]*Department of Computer Science Engineering,*
University College of Engineering, Vizianagaram, AndhraPradesh India
* Corresponding author's Email: yadaiahbalagoni@mgit.ac.in

Abstract: The Apache Hadoop with cloud had become an emerging and popular service. Irrespective of its huge dominance in large scale data processing, it has challenges yet to be addressed. The primary challenges in yarn scheduler are the abilities to automate and control the resource allocation to different workloads in order to meet the deadline-based Service Level Agreement (SLA) in the cloud environment with optimal energy consumption. Our study with the Hadoop YARN addresses this problem in a controlled homogeneous environment. In cloud data-centers, heterogeneity had become a normal phenomenon. Hence, this paper proposes the problem of energy-aware heterogeneous Hadoop Yarn cloud with deadline based SLA. We proposed a SLA-Aware Green Scheduling (SAGS), a Dynamic Voltage/Frequency Scaling (DVFS) based approach along with SLA-Aware scheduling algorithm in the heterogeneous environment. We evaluated SAGS by using benchmark datasets and, compared its performance with previously proposed solutions. Our observation with experimental results shows that, the proposed approach outperforms existing approaches.

Keywords: Energy aware, Heterogeneous, Hadoop YARN, Resource allocation, DVFS, Task scheduling, Cloud computing, Big data processing.

1. Introduction

With the emergence of cloud computing paradigm, social media and Internet of Things (IoT), a huge amount of data is generated continuously. Thus, efficiently processing and extracting the required information (value) from such a huge amount of data is a tedious process. MapReduce was originally designed by Google [1], to address the scalability problem in their search system. MapReduce is a large scale distributed data processing framework which has become a De-facto platform for processing massive social and web graph (e.g. at Yahoo, and Google). The open source implementation of MapReduce was developed by Yahoo and released as apache Hadoop. Both industries and academia, have adopted Hadoop for performing large scale data processing. In order to address the scalability issues with respect to resources, users move to cloud for easy, faster, and pay-as-u-go solutions. Hence, Hadoop ecosystem in the cloud had become a common platform for the data engineers and industries for deploying their data pipeline with less time and cost effective manner.

In cloud computing, the service providers like Amazon, Azure have to provide service level agreement (SLA) for their customers. The deadline has become an important metric in SLA which provides the strict time window for completing the given workload by the customers in order to meet their business requirement. The cloud service providers are primarily focused on SLA's, where the resources are underutilized. According to former CEO of Google, Eric Schmidt, "What matters most to the computer designers at Google is not speed,

but power, low power, because data-centers can consume as much electricity as a city"[2]. In a survey, data-centers power consumption grows 4% every year which is higher than the worldwide electricity consumption in the same time frame [3]. A data-center with 50,000 nodes, can consume more than 100 million kWh/year [4] which is equivalent to 100,000 populations urban in one year.

Dynamic voltage/frequency scaling (DVFS) [5] is a technique used to reduce the power consumption in computing node on the fly. This technique is supported by many modern processors (Intel and AMD). DVFS tunes the processor's core voltage depending on the computation needs. The DVFS processors can operate at multiple performance states which range from P_0 to P_n. P_0 is the highest of all and P_n is lowest performance state where the number of states (n+1) varies with respect to the processors. The modern operating systems provide support Advanced Configuration and Power Interface (ACPI), which provides better over the power management.

This paper is focusing DVFS enabled machines in the cluster. In cloud computing environment, the nodes (machines) in the data-center clusters are heterogeneous since newly bought or upgraded system with old nodes co-exists in the cluster. In addition, Heterogeneity also comes from a variety of hardware and architectures of the cluster nodes. Although, previous efforts have been made for providing power-aware scheduling of Hadoop cluster nodes using DVFS, the heterogeneity in the Hadoop cluster nodes is not yet to be addressed. In this paper, we propose SAGS, a novel approach for handling power-awareness in heterogeneous Hadoop cluster along with deadline SLA. Our contributions are summarized below,

1. Energy-Aware Eco-friendly SLA-Based scheduler for the heterogeneous environment in the cloud.
2. SAGS is implemented in the Hadoop 2.x open source.
3. Finally, we evaluated our approach with the naive approach and compare the performance improvement.

The rest of the paper is organized as follows. We give a brief description about MapReduce framework and Hadoop in section 2. In section 3, we provide the motivation for designing SAGS. We discuss the proposed system architecture, job model, and its components in section 4. In Section 5, gives elaborated details about the performance evaluation, which includes experimental settings, benchmark datasets & workload, and the performance metrics. In section 6 and 7, we discuss the related work and summarize our contributions respectively.

2. Background

This section discusses about MapReduce programming model and its open source implementation, Hadoop. We also discuss about the YARN (Yet Another Resource Negotiator), which does resource management in Hadoop.

2.1 MapReduce

The Hadoop is a load data first architecture which means before processing the data it has placed in the HDFS. Once we uploaded the dataset from the local or remote file system into HDFS. The HDFS split the complete dataset into the block (in our case 64 MB) and the block is replicated based on the replication factor (factor = 3). For instance, we have a dataset of 150 GB size, they are divided into 2400 blocks and replicated based on the replication factor which is 7200 blocks each of 64 MB of size (totally of 450 GB storage space). The complete set of blocks is distributed evenly across the Worker or DataNodes and their references are maintained in the namespace of the NameNode which acts as directory or lookup for future reference. After the dataset is loaded into HDFS, the user writes an application to process them. The user application is referred as User Defined Operation (UDO) which is written by extending the mapper and reducer classes of MapReduce API. Each application time complexity varies with respect to UDO, the type of dataset and the performance of worker nodes.

When the user submits the application to the Hadoop as a job, MapReduce API fetches the data blocks from HDFS as a <key, value> pairs, where keys refer to the offset of the file segment and values has the block of file content. Map tasks process the input <key, value> pairs and store the intermediate <key, value> pairs in the local memory of the container (not in the HDFS). The MapReduce framework uses the hashing function to partition the intermediate keys to the reducer tasks based on the range of keys, which is illustrated in Fig. 1. In most of the cases, the number of reduce tasks will be less than map tasks. After receiving intermediate <key, value> pairs from the map, reduce tasks start processing them based on the threshold parameter, it is calculated by using the percentage of map task completed (for instance 33%). Finally, the output (<key, value> pairs) of reduce tasks are stored back to the HDFS.

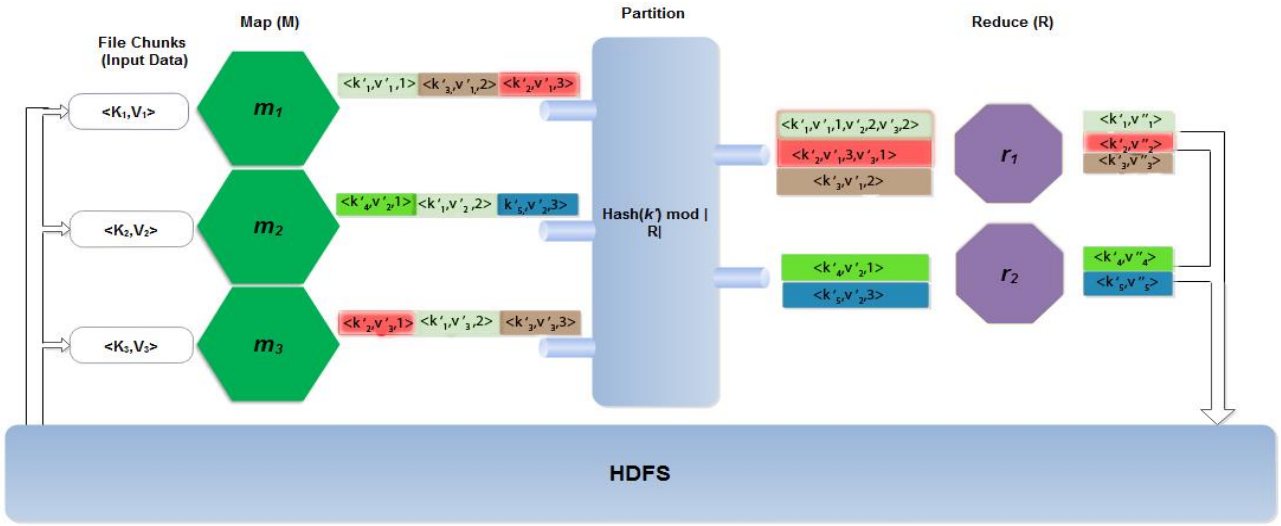


Figure. 1 MapReduce architecture

2.2 Hadoop

Due to its simplicity, Hadoop was developed and released as open source version of MapReduce (MR) framework. The Hadoop 1.x has its own drawbacks, such as scalability, fault-tolerance and inefficient resource management where the resources are managed in a coarse-grained manner as computation slots in each node. At a given instance of time, Hadoop 1.x only support a dedicated MR application, hence, other types of applications cannot be deployed in the cluster. To address the issues in Hadoop 1.x, YARN (Yet Another Resource Negotiator) was proposed which is also known as Hadoop 2.x [6]. The YARN is a novel architecture which is completely different from Hadoop 1.x. YARN provides an abstraction layer to address the cluster resource management problem where each node resources are allocated as containers to the requested application in a fine-grained manner. It also addresses the scalability, collaborative workspace for other application and provides higher fault-tolerance than its predecessor. YARN has a slow-start option which defines when the reducers should start. The Job Tracker functionality in MR is divided into Resource Manager (RM) per cluster, ApplicationsManager (AsM) for all applications and Application Master (AM) per application. RM is responsible for the allocation of the resource for a given application.

3. Motivation

Most of the energy-aware task scheduling algorithms with the deadline-based SLA uses the DVFS techniques [5] on homogeneous cloud environment. Heterogeneity is common problem to be addressed in the cloud environment. The previous approaches [7, 15] does not consider the heterogeneity of the cloud, hence the proposed solutions are homogenous.

Our proposed solution provides the energy-aware resource provision and tuning based on the available workload with deadline-based SLA. In our approach, we primarily consider the heterogeneous aspect of the cloud environment and tune the frequency inside the container where the container has more than one vCore.

4. SAGS

This section discuss about the proposed architecture, its components, job model, SLA- aware scheduler, DVFS frequency tuner and its execution flow. In SAGS, users submit multiple jobs to the ResourceManager (RM), where W represent worker nodes. The components of our proposed architecture (refer Fig. 2) are explained below.

4.1 Job profiler

The job in MapReduce can be formalize by J (refer Eq. (1)), where *I* refers to input dataset, *M* refers to number of map tasks, R refers to number of reduce tasks, D refers to deadline of the MapReduce task based on the SLA. P_M refers to the parallelism of map tasks, and P_R refers to the parallelism of reduce tasks.

$$J = \langle I, M, R, D, P_M, P_R \rangle \quad (1)$$

Where, M^j and R^j (refer Eq. (2) and (3)) represents set of map and reduce tasks for the given job (J).

$$M^j = \left(m_1^j, m_2^j, \cdots, m_{|M|}^j\right) \quad (2)$$

$$R^j = \left(r_1^j, r_2^j, \cdots, r_{|R|}^j\right) \quad (3)$$

In our proposed system, each job (J) is associated with a completion time goal (D^j). The job profilers tracks the related information of (J) from the past logs or sample run of the application. M and R can be calculated based on the block size and on-fly respectively or user specified.

The performance metrics of MapReduce jobs are listed based on the relationship between the resources. Hence, we can categorize them in resource dependent and independent (Refer Table 1).

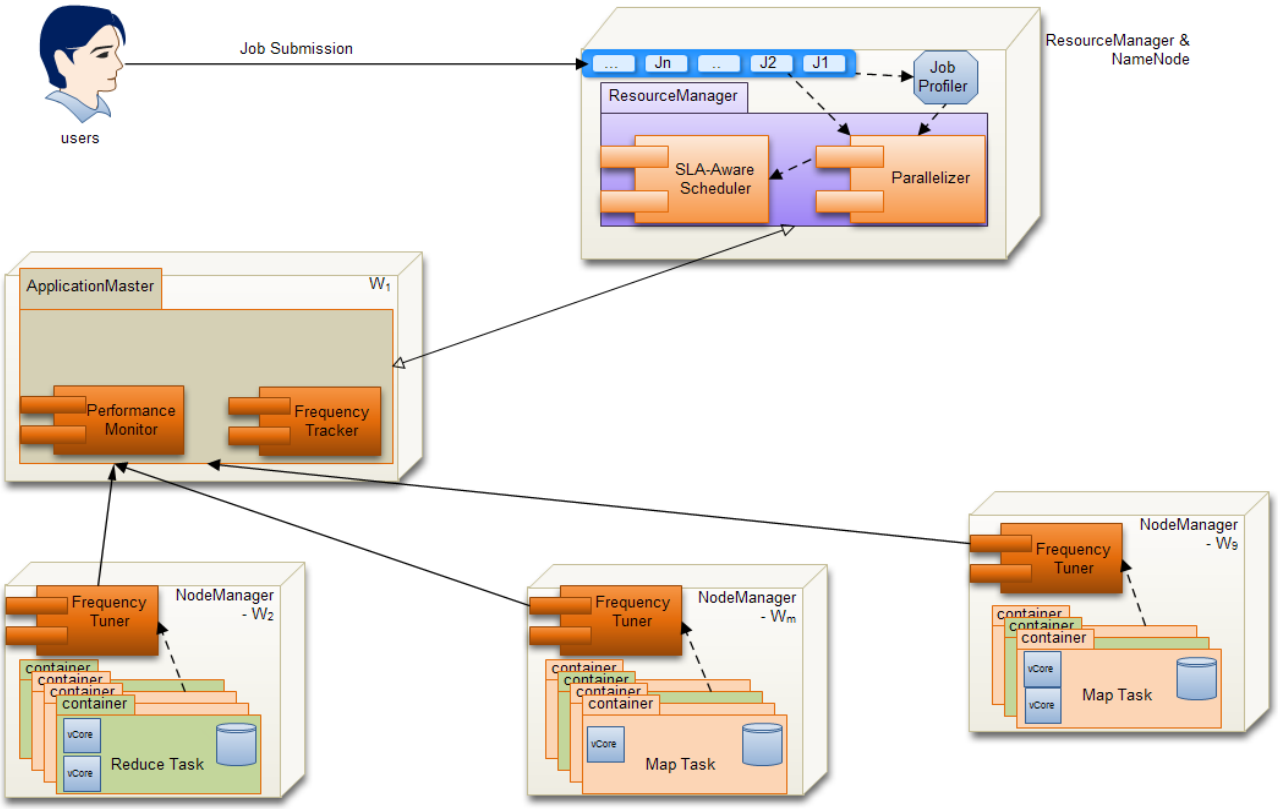


Figure. 2 SAGS Architecture

Table 1. Performance metrics of MapReduce job lifecycle with resource dependencies

Phase	Category	Metric	Description
Initialization	Independent	T_{init}	Time taken to initialize the Job (J)
Map	Dependent	M_{avg}	Average duration of Map tasks (M).
	Dependent	$M_{\max}$	Maximum duration of Map tasks (M).
	Dependent	M_{ratio}	Ratio of Output and Input Map size (M)
Shuffle	Independent	$S_{\max}^1$	Maximum duration of first Shuffle (S).
	Independent	S_{avg}	Average duration of typical Shuffle (S).
	Independent	$S_{\max}$	Maximum duration of typical Shuffle (S).
Reduce	Dependent	R_{avg}	Average duration of Reduce tasks (R).
	Dependent	$R_{\max}$	Maximum duration of Reduce tasks (R).
	Dependent	R_{ratio}	Ratio of Output and Input Reduce size (R)

The job lifecycle in MapReduce is split into multiple phases, they are, initialization, map, shuffle, and reduce phases. The sort operation falls into the shuffle phase of the job lifecycle. Map and reduce phase are dependent with respect to configured resources like vCore (v) and memory.

$$M_{avg}^{j} = \frac{\sum_{i}^{M^j} m_i^j}{\sum_{i}^{M^j} v_i^j} \quad (4)$$

$$R_{avg}^{j} = \frac{\sum_{i}^{R^j} r_i^j}{\sum_{i}^{R^j} v_i^j} \quad (5)$$

The Eqs. (4) and (5), can calculate average duration of map tasks for job *j*. Similarly other variables related to Map and Reduce tasks in the Table 1 are calculated.

4.2 Parallelizer

It calculates the parallelism of map and reduce tasks, which can quantify the proposed job model. When a job (J) is submitted to the job queue from the users, the Resource Manager (RM) allocate the recourse based on the available resources and resource configuration provided by users for map and reduce tasks. The parallelism estimator can calculate the upper bound of the given job *j* (T_{up}^{j}) by using the Make-span Theorem [7]. Refer Eqs. (6) and (7) for detail.

$$T_{up}^{j} = \frac{T_M}{P_M^j} + \frac{T_R}{P_R^j} + Q \quad (6)$$

$$\frac{T_M}{P_M^j} + \frac{T_R}{P_R^j} = C \quad (7)$$

Where,

$$T_M = (M^j - 1) \times M_{avg}$$

$$T_R = (R^j - 1) \times (S_{avg} + R_{avg})$$

$$Q = T_{init} + M_{\max} + R_{\max} + S_{\max}^{1} + (S_{\max} - S_{avg})$$

$$C = D^j - Q$$

We can calculate P_M^j and P_R^j by using Lagrange's method. The parallelism of map and reduce tasks for a given job *j*, is shown Eqs. (8) and (9).

$$P_M^j = \frac{\sqrt{T_M} \times \left(\sqrt{T_M} + \sqrt{T_R}\right)}{C} \quad (8)$$

$$P_R^j = \frac{\sqrt{T_R} \times \left(\sqrt{T_M} + \sqrt{T_R}\right)}{C} \quad (9)$$

4.3 SLA-Aware scheduler

The SLA-Aware Scheduler uses parallel estimator to assign the map and reduce tasks of a given job J. The Worker nodes send heartbeat message to the ResourceManager to acknowledge its presence and work progress information, when the resource request was placed by ApplicationMaster (AM) to RM for providing the list of free containers to perform map and reduce operations. We use Earliest Deadline First (EDF) algorithm for optimal dynamic scheduling in cloud environment. For details refer, Algorithm 1.

Algorithm 1: SLA-Aware Resource Allocation

Input:
jobQueue: job queue.
freeContainers: number of empty containers
Output: ***assignedTasks***: assign task lists
Precondition: Job j is added into the job Queue (*jobQueue*).
ResourceSchedule(jobQueue, freeContainers)
Sort the *jobQueue* in the order of earliest deadline from job model (equ 1)
while (freeContainers> 0)
foreachfreeContainer fc in freeContainers
foreach*job* j in *jobQueue*
if $(M_c^j < P_M^j)$ then
allocate fc to the AM for running the higher priority map task;
assignedTasks.add($m_{c,i}^j$);
elseif
$(M_f^j > completed.maps) \&\& (R_c^j < P_R^j)$ then
allocate fc to the AM for running reduce task;
assignedTasks.add($r_{c,i}^j$);
endif
end for
end for
end while
return assignedTasks;

Table 2. Parallelism for Tera-Sort and Inverted Index

Deadline(s)	Tera-Sort			Inverted Index		
	SAGS	[15] Approach	ARIA	SAGS	[15] approach	ARIA
240	(50, 64)	(56,68)	(48,62)	(57, 43)	(63, 50)	(43,34)
300	(39, 47)	(43, 52)	(38,47)	(40, 32)	(48,38)	(32, 25)
360	(27,38)	(34,44)	(30,41)	(31, 23)	(34,27)	(25, 20)
420	(21,30)	(28,38)	(25,33)	(23, 17)	(27, 21)	(21,17)

The mapred.reduce.slowstart.completed.maps (aka completed.maps) is a threshold (range from 0 to 1), it tells when to start the reducer tasks. This range represents the percentage of overall map tasks get completed or finished. M_c^j and, R_c^j represents running map and reduce tasks of job *j*. Similarly, M_f^j and, R_f^j represents finished map and reduce tasks of job *j*.

4.4 Performance monitor

The Performance Monitor resides in AM and keeps track of the task progress from each NodeManager (NM).The NM watches the task status of its containers present in the respective worker node. The performance monitor can calculate the execution time of the remaining reduce tasks (u_r^j).

$$u_r^j = \frac{(D^j - T)}{\left\lceil \frac{R_r^j + U_r^j}{P_R^j} \right\rceil} - S_{avg} \tag{10}$$

Where,

R_r^j : Currently Running reduce tasks in job *j*

U_r^j : Yet to start reduce tasks in job *j*

T – At the current time of Operation

4.5 Frequency tracker

The Frequency Tracker resides in Application master to collect overall frequency information from worker nodes (W). Frequency Tuner is the service which runs in each worker node and tunes or varies the frequencies of the cores in the container. In the recent version of Linux kernel are preloaded with DVFS support where users can dynamically change the frequency of the processors using RAPL (Running Average Power Limit) interface. The DVFS-enabled processors can run in multiple frequencies under different power supply. Our approach monitors the reducer dataset size (input dataset to each reducer).Based on the workload, finds the frequency and number of vCores [8, 9] can be used. If, we have more workload and deadline is less (D^j – T), then our approach spawns more vCores based on the dataset size and also increase the operating frequency. By tuning, the right combination of frequency and number of vCores, we can complete the task before the given deadline D^j. Due to heterogeneity, we have to find the right frequency for appropriate hardware. Some processor may not operate on specific frequencies. In addition, we can also change the frequency of vCore present in the same container, where a container may have more than one vCore.

5. Evaluation

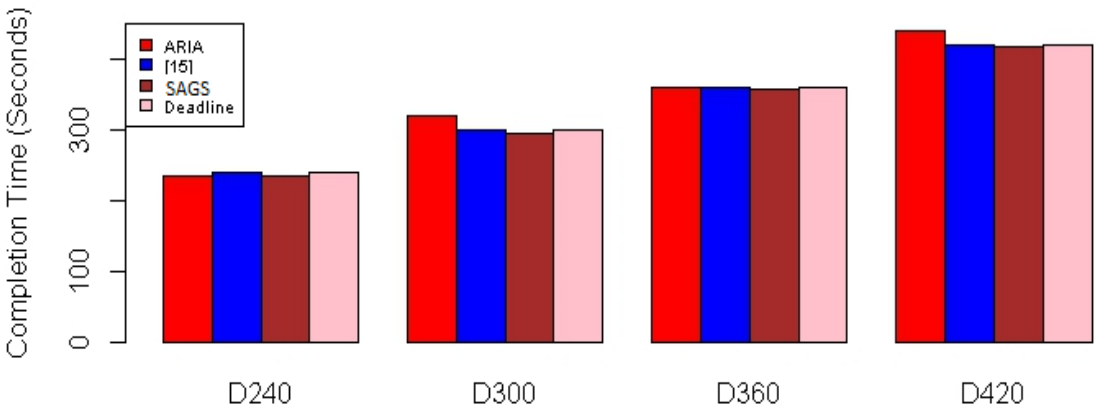
This section discuss about the experimental setting, benchmark datasets used to evaluate SAGS and compared the performance of our proposed system with Hadoop 2.7.1.

5.1 Benchmark dataset

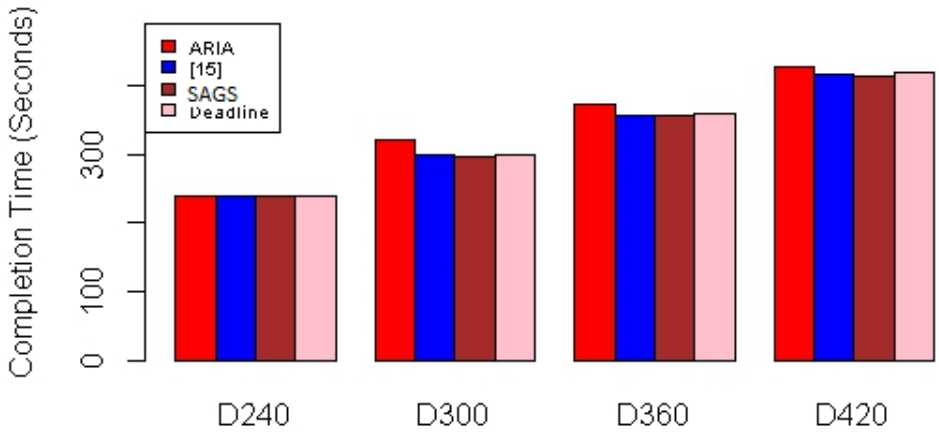
We used PUMA Hadoop benchmark dataset [10]. Refer Table 2, for our experimental datasets.
The benchmark applications used in our evaluation are as follows:

Tera-Sort : This application read the data from HDFS using map function and sort happens internally by using MapReduce inbuilt sort operation. And the Reduce function just store the results (<key, value> pairs) back in HDFS.

InvertedIndex : This application is like Wordcount application. It takes a list of documents as input and generated <word, docId> pairs.



(a)



(b)

Figure. 3 Comparison of execution time of ARIA, [15] and SAGS by using Tera-Sort and Inverted Index: (a) completion time of Tera-Sort and (b) completion time of Inverted Index

5.3 Results and discussion

We calculate the parallelism of jobs based on SAGS, [15], normal deadline and ARIA [7] approaches, where the deadline based approach is the baseline. For instance, Tera-Sort experiment with a deadline of 240 seconds consume (48, 62), (50, 64), and (56, 68) of (#map containers, #reduce containers) respectively.

Fig. 3 (a), illustrates the completion time of Tera-Sort benchmark experiment. In this experiment, we have Tera-sort job with different deadlines - 240, 300, 360, and 420 seconds. Fig. 3 (b), illustrates the execution time of Inverted Index experiment with different deadlines. ARIA [7] violates the deadline SLA in both experiments which are due to its average bound used in the parallelism. Both [15] and SAGS designed for the worst case bound parallelism. During the higher priority jobs or less deadline remaining time the tasks are executed in the higher frequency with more power consumption. The frequency and power are fine tuned to meet the deadline SLA. In our approach, we use dynamic vCore in containers based on the load at reducers which can minimize the skew [9]. More specifically, SAGS can tune frequencies of multiple vCores present in the specific containers, where the intermediate data is high. By this approach, we can improve the performance of the container without our increasing the vCore and achieve the deadline in time than other approaches.

The deadline based baseline approach have deadline which act as SLA. SAGS perform better than deadline based approach which never misses a deadline for the given workload. In some cases, ARIA is violates the given deadline. SAGS performs better than deadline, ARIA and [15] approaches. When the workload increases SAGS consumes 20% lesser map and reduce containers (or resources) than other approaches.

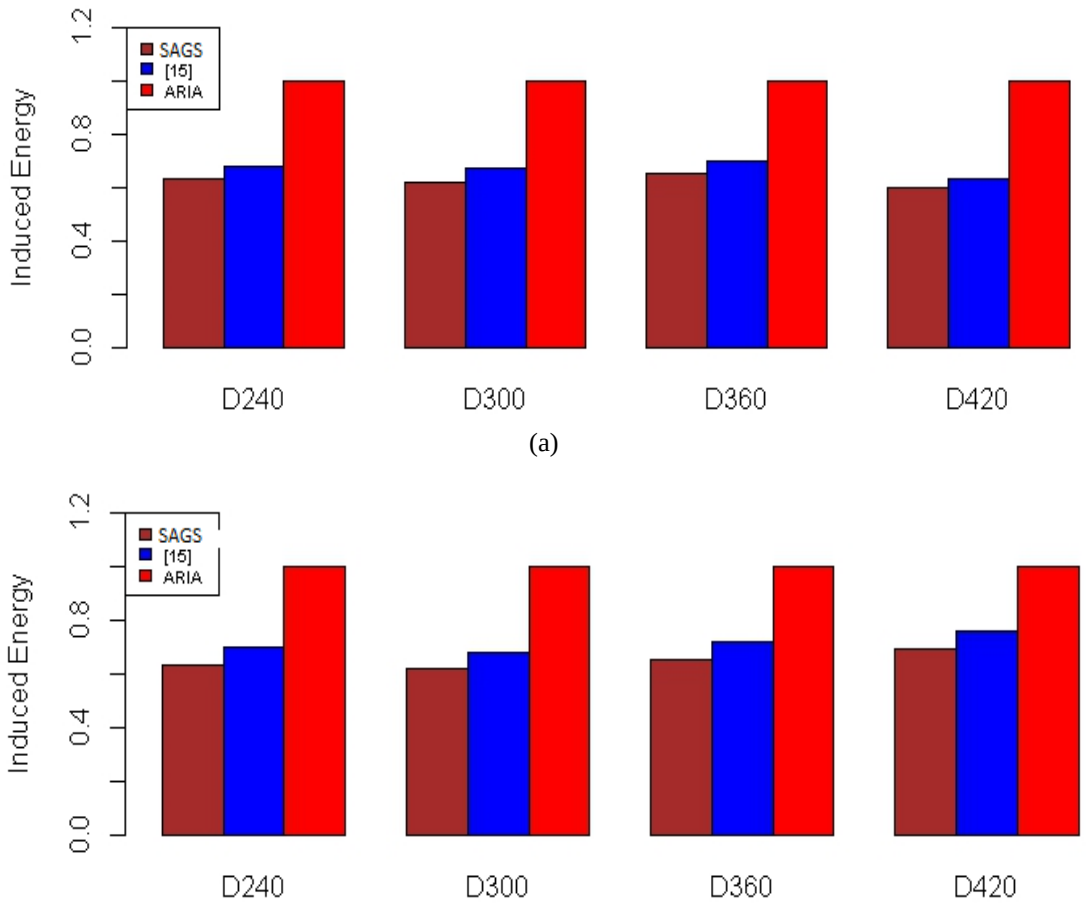


Figure. 4 Comparison of energy consumption of ARIA, [15] and SAGS by using Tera-Sort and Inverted Index: (a) induced energy of Tera-Sort and (b) induced energy of Inverted Index

Our approach, consume less power than [7, 9, 15] approach because, they have not considered the heterogeneity in the cloud environment (refer Figs. 4 (a) and (b)). Overall, the deadline is 100% achieved by SAGS and energy consumption is relatively better than the previous approaches. The energy and achieving the deadline are important in large scale data processing applications. Based on the results, SAGS perform well in both ARIA and [15]. In case of Induced energy, ARIA is considered as a baseline experiment. On Average, SAGS consume 40% lesser energy compare to ARIA and 8% lesser than [15] approach.

6. Related work

Originally, DVFS has been designed for a specific processor [21]. DVFS is a technique used for tuning the operating frequency of a processor which vary the power consumption of processors. In general, more the operating frequency better the CPU performance that can also increase the power consumption. The resource management considers VMs consolidation and cluster re-configuration where the hot or highly used VMs grouped, and under-utilized VMs job aggregated to fewer and rest are decommissioned.

Job scheduling, load balancing, and workload management is an emerging research area in the cloud-based MapReduce environment. The initial version of Hadoop has simple FIFO scheduler which is very efficient for single user and job processing environment. Capacity scheduler [12], was introduced to address the problem of multiple users sharing the same Hadoop environment. To provide fairness between users, the Fair Scheduler [13], was proposed which allocates the resources evenly across multiple users and maximize the data locality.

Jockey [14], is the best fit for a single job with guaranteed latency, it may not fit for the multiple and shared the job at same environment. ARIA [7], estimate and allocate the appropriate resources for the map and reduce tasks of a given job which has not considered the SLA-based deadline and energy cost. Ping et al. [15], proposed the SLA-based deadline aware energy efficient scheduling approach

for the Hadoop environment. But, this approach does not address the heterogeneity issues in the cloud environment which is an important factor to be considered during the design.

The operational expenditure (OPEX) keeps on increasing over the time, where the power consumption is an important factor to be considered. Power consumption has a dual effect - cost and environmental effect. It emits larger Cox gasses which heat up the greenhouse gasses and increases the global warming. To address, the power and energy related problem in an eco-friendly manner, a new flavour of research called green computing, starts evolving. Many energy-aware or optimization approaches have been developed to reduce the energy consumption in HPC clusters or cloud data centers. Originally, DVFS technique has been designed for a specific processor [16]. T. Wirstz et al. [17], proposed three DVFS scheduling polices for MapReduce framework. The results show the sign of improvement in energy efficiency over the proper utilization of DVFS scheduling. Modern DVFS processor is usually equipped with per-core DVFS technique, where each core can operate at multiple frequencies under different supply voltages. Kim et. al, [18], shows the dependency between heterogeneous workload characteristics Vs per-core DVFS. The effective utilization of per-core DVFS can outperform global DVFS. But, how to control and quantify the multiple cores in a DVFS cluster is a challenging problem.

Shyamala L et al. [19] proposed an energy aware resource management technique for job scheduling in data center. This approach categorizes the job in to three different types and assigned based on preemption policy with the earliest available time of the resource (VM) which is attached to a host. In the proposed approach, Hadoop jobs are not considered for preemption. AEGEUS++ proposed an opportunistic frequency tuning algorithm for energy optimization in Hadoop cluster which focus on homogenous environment [20]. AEGEUS++ focus on optimizing the make-span and energy utilization problem.

Our work differs from the previous studies. In this work, we consider both resource provision and energy conservation in heterogeneous cloud-based MapReduce environment. Heterogenous environment is most common in the cloud-based deployment.

Conclusion

In this paper, we present SAGS, energy efficient deadline-based SLA-aware scheduler in the heterogeneous cloud environment. Our approach uses per-core DVFS based technique to tune the frequency based on the given workload of a map or reduce tasks. We evaluated SAGS by using different benchmark datasets and workload (or job). Based on our observation, SAGS outperform the previously proposed solutions. When the workload increases SAGS consumes 20% lesser map and reduce containers (or resources) than other approaches. It also never misses the SLA deadline. On Average, SAGS consume 40% lesser energy compare to ARIA and 8% lesser than [15] approach.

This work can be extended by considering the per-core frequency techniques and GPU based energy optimization in the bigdata processing applications.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large Clusters", *Communications of the ACM*, Vol. 51, No. 1, pp.107-113, 2008.

[2] J. Markoff and S. Lohr, "Intel's huge bet turns iffy", In: New York Times Technology, Section 3, p. 1, 2002.

[3] W.V. Heddeghem, S.Lambert, B.Lannoo, D. Colle, M.Pickavet, and P.Demeester, "Trends in worldwide ICT electricity consumption from 2007 to 2012", *Computer Communications*, Vol. 50, pp.64-76, 2014.

[4] A. Greenberg, J. Hamilton, D.A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks", *ACM SIGCOMM computer communication review*, Vol. 39, No. 1, pp. 68-73, 2009.

[5] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy", In: *Proc. of USENIX Association Conference*, pp. 449-471, 1994.

[6] V.Vinodkumar, A.C. Murthy, C. Douglas, and S. Agarwal, "Apache Hadoop YARN:Yet Another Resource Negotiator", In: *Proc. of the 4th Annual Symposium on Cloud Computing*, 2013.

[7] A. Verma, L. Cherkasova, and R.H.Campbell, "ARIA: automatic resource inference and allocation for MapReduce environments", In: *Proc. of the 8th ACM international conference on Autonomic Computing*, pp. 235–244, 2011.

[8] Z. Liu, Q. Zhang, R. Boutaba, Y.Liu, and B. Wang, "OPTIMA: on-line partitioning skew mitigation for MapReduce with resource adjustment", *Journal of Network and Systems Management*, Vol. 24, No. 4, pp.859-883, 2016.

[9] V. Kumaresan and R.Baskaran, "Aegeus: An online partition skew mitigation algorithm for

mapreduce", In: *Proc. of the International conference on Informatics and Analytics*, 2016.
[10] F. Ahmad, S. Lee, M. Thottethodi, and T.N. Vijaykumar, "Puma: Purdue mapreduce benchmarks suite", 2012. (https://core.ac.uk/download/pdf/10238137.pdf last accessed at 05-march-2017).
[11] Apache Hadoop. *https://hadoop.apache.org.*
[12] Apache. Capacity Scheduler: *http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html*.
[13] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling", In: *Proc. of the 5th European Conference on Computer Systems*, pp. 265–278, 2010.
[14] A.D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: guaranteed job latency in data parallel clusters", In: *Proc. of the 7th ACM European conference on Computer Systems*, pp. 99–112, 2012.
[15] P. Li, L. Ju, Z. Jia, and Z. Sun, "SLA-aware energy-efficient scheduling scheme for Hadoop YARN", In: *Proc. of the 2015 IEEE 17th International conference on High Performance Computing and Communications, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pp. 623-628, 2015.
[16] W. Kim, D. Shin, H.S.Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems", In: *Proc. of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 219–228, 2002.
[17] T.Wirtz and R.Ge, "Improving mapreduce energy efficiency for computation intensive workloads", *In: Proc. of the 2011 International Green Computing Conference and Workshops*, pp. 1–8, 2011.
[18] W. Kim, M.S. Gupta, G-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators", In: *Proc. of the IEEE 14th International Symposium on High-Performance Computer Architecture*, pp. 123–134, 2008.
[19] S. Loganathan, R.D. Saravanan, and S. Mukherjee, "Energy aware resource management and job scheduling in Cloud Datacenter", *International Journal of Intelligent Engineering and Systems*, Vol. 10, No. 4, pp.175-184, 2017.
[20] V. Kumaresan, R. Baskaran, and P. Dhavachelvan, "AEGEUS++: an energy-aware online partition skew mitigation algorithm for MapReduce in cloud", In: *Proc. of the IEEE International conference on Cluster Computing*, pp.1-18, 2017.
[21] W. Kim, D. Shin, H-S. Yun, J. Kim, and S.L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems", *In: Proc. of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 219–228, 2002.