IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON EMERGING TRENDS, ISSUES, AND CHALLENGES IN ENERGY-EFFICIENT CLOUD COMPUTING

Received June 4, 2017, accepted July 1, 2017, date of publication July 12, 2017, date of current version August 8, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2725318

Storage-Tag-Aware Scheduler for Hadoop Cluster

NAWAB MUHAMMAD FASEEH QURESHI[1], DONG RYEOL SHIN[1], (Member, IEEE), ISMA FARAH SIDDIQUI[2], AND BHAWANI SHANKAR CHOWDHRY[3], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 440-746, South Korea
[2]Department of Software Engineering, Mehran University of Engineering and Technology, Jamshoro 76062, Pakistan
[3]Faculty of Electrical Electronics and Computer Engineering, Mehran University of Engineering and Technology, Jamshoro 76062, Pakistan

Corresponding author: Dong Ryeol Shin (drshin@skku.edu)

This work was supported by Basic Science Research Program through the National Research Foundation of Korea through the Ministry of Education under Grant NRF-2017R1D1A1B03032855.

ABSTRACT Big data analytics has simplified the processing complexity of extremely large data sets through ecosystems, such as Hadoop, MapR, and Cloudera. Apache Hadoop is an open-source ecosystem that manages large data sets in a distributed environment. MapReduce is a programming model that processes massive amount of unstructured data sets over Hadoop cluster. Recently, Hadoop enhances its homogeneous storage function to heterogeneous storage and stores data sets into multiple storage media, i.e., SSD, RAM, and DISK. This development increases the performance of data block placement strategy and allows a client to store large data sets into multiple storage media efficiently than homogeneous storage. However, this evolution increases the consumption of computing capacity and memory usage over MapReduce job scheduling. The scheduler processes MapReduce job into homogeneous container having configuration of CPU, memory, DISK volume, and network I/O, and accesses, processes, and stores data sets over heterogeneous storage media. This produces a processing latency of locating and pairing source data set to MapReduce tasks and results an abnormal high consumption of computing capacity and memory usage in a Datanode. Similarly, when scheduler assigns MapReduce jobs to multiple Datanodes, the same processing latency can severely affect the performance of whole cluster. In this paper, we present Storage-Tag-Aware Scheduler (STAS) that reduces processing latency by scheduling MapReduce jobs into heterogeneous storage containers, i.e., SSD, DISK, and RAM container. STAS endorses job with a tag of storage media, such as Job_{SSD}, Job_{DISK}, and Job_{RAM} and parses them into heterogeneous shared-queues, which assign processing configuration to enlist jobs. STAS manager then schedules shared-queue jobs into heterogeneous MapReduce containers and generates an output into storage media of the cluster. The experimental evaluation shows that STAS optimizes the consumption of computing capacity and memory usage efficiently than available schedulers in a Hadoop cluster.

INDEX TERMS Hadoop, HDFS, scheduler, MapReduce, storage-tier.

I. INTRODUCTION

Big data analytics has opened a new gateway to think beyond the limits set by traditional data management systems. Nowadays, we find number of large dataset management systems such as Apache Hadoop [1], MapR [2] and Cloudera [3] in market. Hadoop [4] is an open-source dataset management system of Apache Software Foundation and supports large datasets processing in a distributed parallel environment. It consists of four core components; Hadoop-common, Yet Another Resource Negotiator (YARN) [5], Hadoop Distribution File System (HDFS) [6], and MapReduce [7]. Hadoop-common is a library of utilities, which manages I/O operations of the cluster. YARN is a resource manager that allocates resources and schedules MapReduce jobs in the cluster. HDFS is a file system that stores and processes large datasets into storage media. MapReduce is a YARN-based programming model that processes client's jobs over large datasets and generates output into storage media. In order to understand the integrated working of Hadoop components, let's assume that a client submits a MapReduce job into the ecosystem. YARN assigns job into default scheduler and allocates resource parameters i.e. memory, CPU, DISK volume and network I/O to container [8]. In the next step, scheduler processes MapReduce job into homogeneous con-

tainers and generates an output over storage media of HDFS. The file system then transforms output of MapReduce job into data blocks and performs data block placement function to store them into storage media of HDFS. The default size of a data block is 64 MB and can be re-configured to 128 and 256 MB [9].

MapReduce is a distributed programming model and consists of two main functions i.e. Map and Reduce. The map function executes map tasks such as filtering a value n from a file m over a dataset and generates a result r. The reduce function collects map output through combiner task $c\,(r)$ and merges them together to produce an output R into storage media of HDFS. In the distributed environment, map function executes map tasks into i number of nodes and generates a result $(n+i)$. The reduce function collects distributed map results through combiner task $c\,(n+i)$ and merges them together to produce an output $j\,(R)$ into storage media of HDFS [10].

Recently, Hadoop upgrades its homogeneous storage paradigm to heterogeneous storage and stores datasets into multiple storage media i.e. SSD, RAM and DISK [11]. This enhancement increases the performance of data block placement function and allows a client to store large datasets with an increased speed of $4x$ times using SSD and $12x$ times through RAM than DISK media [12] [13]. However, this development raises consumption of computing capacity and memory usage at job scheduling. The scheduler processes MapReduce job into a homogeneous container i.e. DISK media having resource parameters of CPU, memory, DISK volume and network I/O [14] and accesses, processes and stores dataset into heterogeneous storage media i.e. DISK, SSD and RAM. This generates asynchronous communication of accessing, processing and storing map and reduce tasks into mapper and reducer containers from heterogeneous storage media and produces a processing latency of locating and pairing path of datasets into homogeneous containers. As a result, MapReduce consumes an abnormally high consumption of computing capacity and memory usage in a Datanode [15]. Secondly, the scheduler processes jobs in a single shared-queue environment that assigns equal computing capacity to enlist jobs. It is now clearly observed that, computing capacity of homogeneous container differs from accessing, processing and storing capacity of heterogeneous storage media [16], therefore, MapReduce job observes random values of read/write blocks per second with same processing time of MapReduce job.

In order to elaborate the discussed problems, we process a single MapReduce job of 'Wordcount' through default scheduler having heterogeneous storage media i.e. RAM, SSD and DISK over a dataset of 128 MB. We observe that the job finishes processing in 109 seconds and writes an output file in 15, 32 and 61 second over RAM, SSD and DISK storage media. HDFS performs data block placement function in 13, 19 and 35 seconds and allows a client to read blocks in 2, 6 and 16 seconds over RAM, SSD and DISK

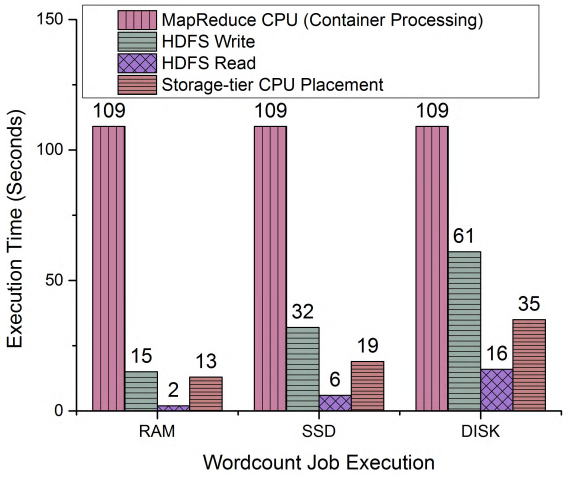

FIGURE 1. Wordcount job execution into storage media of Hadoop.

storage media. This huge difference of time in writing an output among multiple storage media reflects the impact of processing latency. Moreover, it affects data block placement strategy with storing latency as shown in Fig.1.

In order to resolve the above discussed issues, we present Storage-Tag-Aware Scheduler (STAS) that reduces processing latency by scheduling MapReduce jobs into heterogeneous storage container. STAS declares job with a tag of storage media such as Job_{SSD}, Job_{DISK} and Job_{RAM} and parses them into shared-queues i.e. $Queue_{SSD}$, $Queue_{DISK}$ and $Queue_{RAM}$ of heterogeneous queue container. STAS manager schedules tag jobs from shared-queues and processes into heterogeneous MapReduce containers i.e. mapper and reducer container of DISK, SSD and RAM and generates an output into storage media of HDFS. The significant contributions of STAS are to be noted as:

- A novel endorsement of MapReduce job with storage media tag.
- A simple parser to filter out tag jobs from a shared-queue.
- A functional enhancement of recognizing tag jobs at scheduler layer.
- Assignment of heterogeneous storage media volumes as a container for MapReduce processing
- Scheduling tag jobs through heterogeneous storage media container.

The remaining paper includes six sections, which describes the proposed approach in following manner. Section II provides a brief overview about Hadoop ecosystem and presents a motivation of processing latency problem. Section III includes related work of all such researchers, which contributed their schemes and proposed ideas to strengthen the functionality of Hadoop scheduler. Section IV discusses STAS in detail. Section V presents experimental evaluation of STAS individually and comparatively with other schedulers of Hadoop cluster. Section VI concludes STAS and highlights future work contribution.

II. OVERVIEW AND MOTIVATION

MapReduce (MRv2) is an open-source programming model that processes client's job over a dataset of Hadoop cluster. It is an integral part of YARN and works in collaboration with two components i.e. (i) Resource Manager and (ii) Application Master. Resource manager is a master component that arbitrates cluster resources and processes MapReduce jobs through scheduler and application manager. Application master negotiates job's resources with resource manager and informs Node managers about container configuration of MapReduce job. The node manager is a framework agent that manages and monitors resource usage i.e. CPU, memory, DISK volume and network I/O into a Datanode. It is also responsible to handle a container, which consists over a summary of memory, CPU, secondary storage volume and network I/O configuration set for MapReduce job processing. The map and reduce functions are two individual procedures and processes large datasets into two containers i.e. Mapper and Reducer container. The scheduler is responsible for allocating resources to running applications and scheduling tasks into containers. Thus, at first, it allocates resources to a MapReduce job and then schedules map tasks into mapper container and reduce tasks in reducer container as shown in Fig.2.

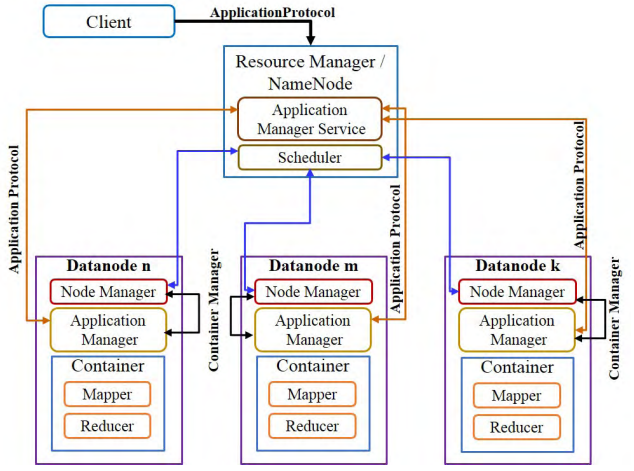

FIGURE 2. MapReduce job processing into Hadoop Cluster.

Hadoop ecosystem uses three default schedulers, (i) First-In First-Out (FIFO), (ii) Fair, and (iii) Capacity scheduler. FIFO [17] is a simple scheduling algorithm that processes single job at a time having an order of First-In First-Out. FIFO is not equipped with queue configuration and management. Therefore, it processes jobs with full allocation of resources in the cluster. Fair Scheduler (FS) [18] allocates resources to all jobs in such a way that each job gets an equal share of resources over time. This finishes short jobs in reasonable time and shares unallocated resources among multiple users of cluster. FS is not programmed to understand shared-queue management among multiple users. Therefore, it assigns equal resources to MapReduce containers and generates job starvation problem. Capacity Scheduler (CS) [19] processes jobs in shared-queues and allows multiple organizations to share resources in distributed and parallel environment. It executes MapReduce jobs in an operator-friendly environment and maximizes the throughput and utilization by guaranteeing an organization with minimum capacity allocation of resources into a cluster. The scheduler allocates homogeneous resource parameters i.e. CPU, memory, DISK volume and network I/O configuration to shared-queue organizations and performs accessing, processing and storing dataset into heterogeneous storage media i.e. SSD, DISK and RAM. This produces job processing latency of locating and pairing source dataset to MapReduce job and consumes an abnormally high CPU time and memory usage in a Datanode.

STAS resolves this problem through four adequate measures i.e. (i) enhances a container with features of heterogeneous storage media volume, (ii) assigns media compatible processor core percentile to MapReduce jobs, (iii) schedules jobs through heterogeneity-aware STAS manager, and (iv) re-programs Map and Reduce functions to process tag jobs.

III. RELATED WORK

Many researchers have presented their contributions for enhancing the performance of Fair and Capacity schedulers and optimized the consumption of resources and job execution time over Hadoop. We have observed a good number of research contributions explaining the scheduling techniques into heterogeneous network and node environment [20]. To simplify the discussion, we divide these schedulers into two categories i.e. (i) Network heterogeneity and (ii) Node heterogeneity. The network heterogeneity involves scheduling of MapReduce jobs with multiple operating systems and network protocols. Therefore, schedulers access, process and store datasets in an inter-operable environment and retains mobility and Quality of Service (QoS) among nodes of the cluster [21]. The node heterogeneity includes inter-node processing of MapReduce job with the help of multiple core processors and memory types [22]. The recent development of adopting heterogeneous storage media strengthens network and node heterogeneity performance [11]. Let's elaborate such related schedulers that supports the concept of heterogeneity in processing a MapReduce job.

Delay scheduler [23] is a functional enhancement into Fair scheduling algorithm and re-configures a job with custom waiting time. As a result, short job finishes earlier than long job and optimizes the consumption of resources in the cluster. The Energy-efficient scheduler [24] presents a compression scheme that encodes map task's result and decodes it back at reduce function. This strategy minimizes consumption of resources over generation of input split programs and decreases data transfer time of MapReduce job in a network. However, it produces processing latency in decoding large number of input split results at reduce

function and degrades the performance of cluster at heterogeneous storage media. Matchmaking scheduler [25] addresses a method of locating nearby nodes and grabbing their tasks for job processing. In this way, a cluster executes nearby MapReduce jobs at first priority and compute distant jobs in second schedule. This reduces network congestion and decreases consumption of resources over job search in the heterogeneous network. However, node heterogeneity environment performs media lookup functions and increases network congestion and resource consumption in searching and executing MapReduce job in distant nodes. Thus, it is useful for locality-aware scheduling in heterogeneous network. Late scheduler [26] presents an approach of examining map tasks and their processing priorities. After the assessment takes place, jobs get into their respective shared-queues of high, medium and low priority processing. The cluster processes MapReduce jobs in high-medium-low order over heterogeneous network environment. This approach delivers an effective MapReduce job processing with the accessibility of datasets through DISK storage media. However, it requires an additional configuration of accessing, processing and storing dataset over heterogeneous storage media. Therefore, late scheduler is not much beneficial in heterogeneous storage environment. DyScale scheduler [27] divides processor's core into virtual resource pool of fast and slow cores. The capacity strength of fast core is 70% higher than slow core and therefore, scheduler processes MapReduce jobs quickly over fast core. However, when the same configuration of virtual pool is assigned to access, process and store dataset into heterogeneous storage media, MapReduce job consumes an equal computing time into RAM, SSD and DISK storage media but differs into read/write and block placement time as shown in Fig.1. Tarazu scheduling technique [28] consists of a prediction model that collects reasons of delay in scheduling a MapReduce job. The prediction model works in two steps i.e. (i) training phase and (ii) evaluation phase. The training phase collects statistics for combiner task of reduce function over homogeneous storage media i.e. DISK volume. The evaluation phase simulates combiner task processing and removes delay reasons i.e. inactive node delay and empty result of input split programs. This approach is simulated into homogeneous storage environment and requires an additional modeling to address delay reasons in heterogeneous storage media. Adaptive scheduler [29] reduces resource consumption through sharing map container with other MapReduce and non-MapReduce applications. By default, scheduler constructs new container for MapReduce job and do not recommend to re-use a container having old dump of tasks. Therefore, the adaptive scheduling algorithm reduces resource consumption but increases security and malfunctioning issues of MapReduce job. BAR [30] scheduling algorithm processes MapReduce job in two steps. At first, it searches near by nodes through calculating node distance and assigns input split programs of map function over them. In the next step, it performs reduce function through a random node and reduces resource consumption and workload problems. This scheduling technique is designed for processing MapReduce job into homogeneous storage devices i.e. DISK.

There are few researchers who presented heterogeneous storage frameworks of Hadoop cluster. Their contributions are discussed in following paragraph.

K. R. Krish et al., present heterogeneous storage framework named hatS [31], which enhances functional layer of HDFS with heterogeneous storage media. HATS provides brief functional policies to store and retrieve datasets into multiple storage media i.e. SSD and DISK. The proposed framework was published prior to official release of Hadoop's heterogeneous storage frame, therefore, it is not applicable to be used at the moment. N. S. Islam et al., propose a hybrid framework named Triple-H [32] that uses multiple storage media i.e. RAM, SSD and DISK for storing and exchanging block replicas in Hadoop. Triple-H provides a dynamic data block placement and replica management strategy in node heterogeneity environment. This framework is also published prior to official release and hence, it is not used in current Hadoop built. There are several other data placement strategies available but our approach is focused over scheduling MapReduce job over heterogeneous cluster.

The discussed schedulers use homogeneous container i.e. DISK volume for processing map and reduce functions. In order to observe comparison with our proposed approach, we equipped above schedulers with functionality of heterogeneous containers i.e. RAM and SSD volume. We find that schedulers do not recognize SSD and RAM volumes and consider them as DISK volume. The reason of not identifying strength of heterogeneous container is associated with old programming of core percentile allocation and homogeneous job processing into heterogeneous storage media.

IV. STORAGE-TAG-AWARE SCHEDULER (STAS)

The proposed approach STAS is an add-on to existing capacity scheduler and presents a functional enhancement of heterogeneous containers in MapReduce job processing. Lets understand the role of STAS in processing a MapReduce job as follows:

When a client submits MapReduce job $Job_i = \left(Code, Tag_{(SSD,RAM,DISK)}\right)$ to Namenode, resource manager schedules job Job_i with resources $Job_R = MR_{Container}$ and execution parameters $Job_P = (Mapper_i, Reducer_i)$, where $Code$ represents client task, $Tag_{(SSD,RAM,DISK)}$ depicts storage media tag in a MapReduce job, $MR_{Container}$ includes a configuration of CPU, memory, (SSD,RAM,DISK) volume and network I/O, $Mapper_i$ represents number of map tasks and $Reducer_i$ shows number of reduce tasks. By default, YARN allocates equal quota of computing capacity to the enlist jobs of shared-queue [33]. Since, STAS introduces three containers of storage media, therefore, it re-configures shared-queue in three forms i.e. $Queue_{SSD}$, $Queue_{DISK}$ and $Queue_{RAM}$ and allocates a core percentile $Core_{\%} =$

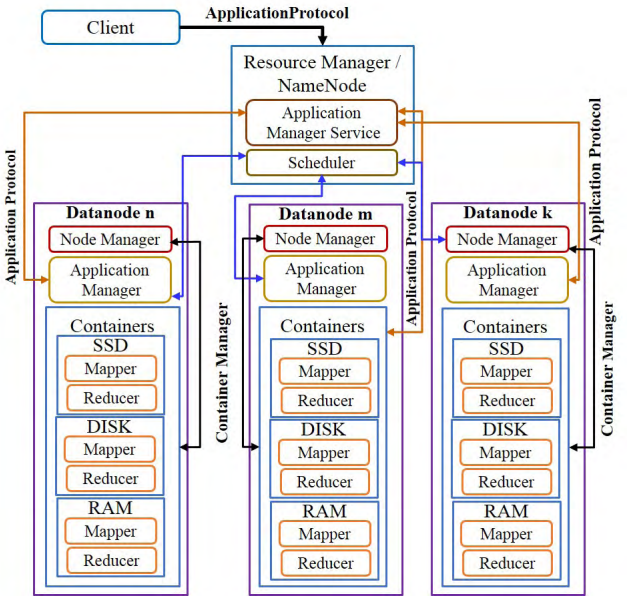

FIGURE 3. STAS job work-flow over Hadoop cluster.

{(30% *Core*, *SSD*), (50% *Core*, *DISK*), (20% *Core*, *RAM*)} to queues respectively. Heterogeneous queue container (HQC) manages I/O of shared-queues and parses job Job_i into them. STAS manager schedules shared-queue jobs into heterogeneous media containers and generates an output over storage media of HDFS as shown in Fig.3.

The functional work-flow of STAS can be understood by observing a complete MapReduce job cycle in Flow Chart environment. The tag job Job_i arrives at rate λ and consumes a parsing time QC_J to get into shared-queues i.e. $Queue_{SSD}$, $Queue_{DISK}$ and $Queue_{RAM}$ of HQC. STAS manager schedules a job Job_i from queue container and generates parallel map tasks to process map operation over heterogeneous map container in time T_{Map}. The scheduler then fetches map result-set from map container and generates parallel reduce tasks to process reduce operation over heterogeneous reduce container in time T_{Reduce}. The output of MapReduce job Job_i is stored into storage media of HDFS as shown in Fig.4.

In order to understand STAS briefly, let's assume that a client submits job Job_i at a submission rate of λ. The number of jobs can be obtained as,

$$Job_{QC} = \frac{Job_i}{\lambda_{Submission}} \tag{1}$$

Where $\lambda_{Submission}$ is service rate to submit a job and Job_{QC} is the number of jobs submitted at arrival rate λ.

After receiving tag jobs, STAS processes Job_i through four components written as:

1) Heterogeneous Queue Container (HQC),
2) STAS Manager,
3) Map processing, and
4) Reduce processing.

A. HETEROGENEOUS QUEUE CONTAINER (HQC)

A shared-queue is a type of local queue and consists of MapReduce jobs having configuration parameters i.e. queue capacity, memory allocation and core percentile allotment [34]. STAS re-programs a homogeneous shared-queue into three types of heterogeneous shared-queues i.e. $Queue_{SSD}$, $Queue_{DISK}$ and $Queue_{RAM}$ having respective memory and core percentile $Core_{\%}$ allotment. The memory allocation depends upon the capability limit of node and core percentile $Core_{\%}$ is distributed in such a way that each shared-queue uses a balanced CPU time for MapReduce job processing. Therefore, when a job Job_i gets into Heterogeneous Queue Container (HQC), it is identified by a storage media tag i.e. Job_{SSD}, Job_{DISK} and Job_{RAM} and parsed into respective heterogeneous shared-queues as shown in Algorithm-1.

The job Job_i in shared-queue $Queue_{SSD}$ can be represented as,

$$Job_{SSD} = \frac{Queue_{SSD}}{QC_{J_{SSD}}} \tag{2}$$

Similarly, the job Job_i in shared-queue $Queue_{DISK}$ can be represented as,

$$Job_{DISK} = \frac{Queue_{DISK}}{QC_{J_{DISK}}} \tag{3}$$

In the same way, the job Job_i in shared-queue $Queue_{RAM}$ can be represented as,

$$Job_{RAM} = \frac{Queue_{RAM}}{QC_{J_{RAM}}} \tag{4}$$

Algorithm 1 Heterogeneous Shared-Queue Parser of HQC

```
1:  procedure Redirect
2:      Job_i ← type of MapReduce job
3:      Q_1..3 ← HeterogeneousQueue
4:      top:
5:      if Job_i ==' SSD' then
6:          Queue_SSD ← Job_i
7:      else if Job_i ==' DISK' then
8:          Queue_DISK ← Job_i
9:      else if Job_i ==' RAM' then
10:         Queue_RAM ← Job_i
11:     Queue:
12:     if job_i = Empty then
13:         close;
14:     if Queue = Queue - 1 then insert Job_i
15:         Queue ← Job_i-1.
16:         goto Queue.
17:         close;
18:     goto top.
```

Algorithm-1 explains the procedure of identifying a tag in job Job_i and submitting it to respective share-queue. The purpose of using a tag in MapReduce job depends on the requirement of client, where organization specifically demands to

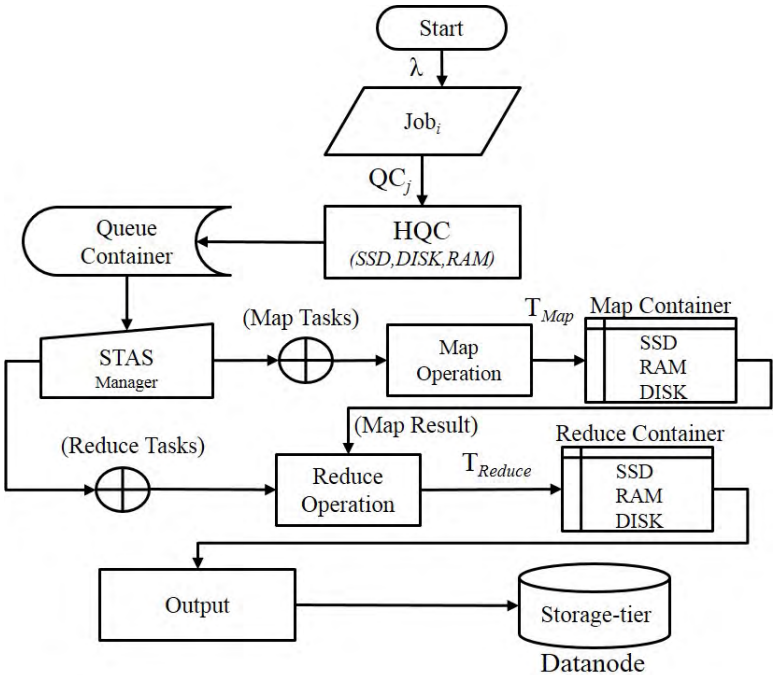

FIGURE 4. STAS Heterogeneous Job Processing Flow Chart.

process a MapReduce job over multiple heterogeneous storage media. By default, a MapReduce job executes into DISK container and this feature enhances client to process their jobs into heterogeneous storage container i.e. SSD and RAM.

The collection of tag jobs into HQC can be represented as,

$$HQC_{Jobs} = (Queue_{SSD}, Queue_{DISK}, Queue_{RAM}) \quad (5)$$

Where, HQC_{Job} is the heterogeneous queue container for storing shared-queues MapReduce jobs.

B. STAS MANAGER (SM)

STAS Manager is an add-on function to existing Capacity scheduler and schedules shared-queue jobs into heterogeneous container $MR_{Container}$. It adopts the same scheduling technique with a change in sub-scheduling methods, which adds tags into a MapReduce job and processes into heterogeneous storage containers. A tag job scheduling can be represented as,

$$Scheduler_{STAS} = \left\{ \begin{array}{c} Container_{DISK} \xleftarrow{schedule} Job_{DISK} \\ Container_{SSD} \xleftarrow{schedule} Job_{SSD} \\ Container_{RAM} \xleftarrow{schedule} Job_{RAM} \end{array} \right\} \quad (6)$$

By default, the scheduler is programmed to schedule map and reduce tasks into two containers i.e. mapper and reducer [35]. STAS adopts the same configuration and uses heterogeneous containers for map and reduce task processing as observed in Fig.4. Therefore, when a tag job schedules at mapper and reducer containers, STAS observes performance metrics and calculates (i) time to pick a job T_{Job_i}, (ii) time to service map function into mapper container T_{Map}, (iii) time to collect results from mapper container and drops at reducer container $T_{Switch(Job_i)}$ and (iv) time to service reduce function into reducer container T_{Reduce}. The total MapReduce job execution time can be obtained as,

$$T\eta_i = T_{Job_i} + T_{Map} + T_{Reduce} + T_{Switch(Job_i)} \quad (7)$$

The time T_{Map} of mapper container depends upon the usage of memory and core percentile. Therefore, it can be obtained as:

$$T_{Map} = Map_{memory(used)} + Map_{core\%(used)} \quad (8)$$

Similarly, the time T_{Reduce} of reducer container depends upon the usage of memory and core percentile. Therefore, it can be obtained as:

$$T_{Reduce} = Reduce_{memory(used)} + Reduce_{core\%(used)} \quad (9)$$

STAS manager uses two local shared-queues i.e. $Queue_{Map}$ and $Queue_{Reduce}$ for scheduling map and reduce functions into heterogeneous containers. The local shared-queues submits tasks to mapper and reducer container in FIFO order.

The computing capacity of job Job_i into mapper container can be obtained as,

$$Capacity_{Mapper} = Job_i \left(\frac{Queue_{Map[i]}}{Container_{Core\%}} \right) \quad (10)$$

Similarly, the memory consumption of job Job_i into mapper container can be obtained as,

$$Memory_{Mapper} = Job_i \left(\frac{Queue_{Map[i]}}{Container_{memory}} \right) \quad (11)$$

In the same way, the computing capacity of job Job_i into reducer container can be obtained as,

$$Capacity_{Reducer} = Job_i \left(\frac{Queue_{Reduce[i]}}{Container_{Core\%}} \right) \quad (12)$$

Similarly, the memory consumption of job Job_i into reducer container can be obtained as,

$$Memory_{Reducer} = Job_i \left(\frac{Queue_{Reducer[i]}}{Container_{memory}} \right) \quad (13)$$

where, $Queue_{Mapper[i]}$ and $Queue_{Reducer[i]}$ represents an index of map and reduce tasks of MapReduce job.

Thus, a single map and reduce task processing can be observed as,

$$Map_{Task[i]} = \left\{ \frac{(Capacity_{Mapper}, Memory_{Mapper})}{T_{Map}} \right\} \quad (14)$$

$$Reduce_{Task[i]} = \left\{ \frac{(Capacity_{Reducer}, Memory_{Reducer})}{T_{Reduce}} \right\} \quad (15)$$

The Algorithm-2 shows that STAS manager schedules map and reduce tasks into mapper and reducer containers and produces an output into storage media of HDFS.

Algorithm 2 STAS Manager

procedure Schedule HQC jobs
  $Job_i \leftarrow$ type of *MapReducejob*
  $T_{Job_i} \leftarrow$ Time to pick a Job_i
  $MR_{Container} \leftarrow$ Processing Ratio of Job_i
  $T_{Map} \leftarrow$ Time to complete Map processing
  $T_{Reduce} \leftarrow$ Time to complete Reduce processing
  $T_{Switch(Job_i)} \leftarrow$ Time to switch Map result to Reducer
  $Mapper_C \leftarrow$ Mapper container of $MR_{Container}$
  $Reducer_C \leftarrow$ Reducer container of $MR_{Container}$
  Pick Job_i *from HQC*:
  get $Job_{(Tag_{SSD}, Tag_{RAM}, Tag_{DISK})} \leftarrow HQC_{Container}$
  Assign $(Mapper_C, Reducer_C) \leftarrow MR_{Container}$
  Process $Mapper_C \leftarrow Job_i \left(Queue_{Map_{[i]}}\right)$
  Collect $(Mapper_C)_{Result}$
  Process $Reducer_C \leftarrow$
  $\left\{(Mapper_C)_{Result}, Queue_{Reduce_{[i]}}\right\}$
  Store $HDFS_{Storage-tier} \leftarrow Reducer_C$
  Map Function:
  if $Job_i\,(Map_{Task})\,.all =$ "*success*" **then**
    Serialize $Mapper_{C(Dataset)}$
    Return
  Reduce Function:
  if $(Reduce_{Task}) =$ "*success*" **then**
    Generate $Job_{(i)Resultset}$
    Return
  Return $T_{Job(i)}, T_{Map}, T_{Reduce}, T_{Switch(Job_i)}$

C. MAP PROCESSING

Map processing is a procedure to produce input split programs of job Job_i in a scheduling algorithm [36]. STAS re-programs this subroutine to understand heterogeneous input splits of job Job_i and enhances its functionality to fetch source datasets from heterogeneous storage media of HDFS.

Map processing works in two phases i.e. (i) Data File Locality and (ii) Input split formation.

1) DATA FILE LOCALITY (DFL)

Data file locality preserves original location of source dataset and shares block address to job Job_i. Therefore, a map processing instance when requests for the location of a dataset, Namenode responds with a metadata file having location of original dataset and replicas in Hadoop cluster. The number of Data file locations per Job_i can be obtained as:

$$DFL_n = \left\{ original \left(\frac{Dataset}{Location} \right), replica \left(\frac{Dataset}{Location} \right)_i \right\} \quad (16)$$

After receiving a data file location DFL_n, map processing generates split function of accessing dataset from heterogeneous storage media. The number of splits can be obtained as:

$$Splits_n = DFL_n \quad (17)$$

2) INPUT SPLIT TASKS

Map function produces input splits to computes client's job into heterogeneous mapper container. The function requires an input of three parameters i.e. job code, splits $Splits_n$ and number of mappers $Mapper_i$ and produces input split tasks of job Job_i. The number of input split tasks for heterogeneous mapper container can be obtained as,

$$InputSplit_n = (Code, Splits_n, Mapper_i) \quad (18)$$

D. REDUCE PROCESSING

Reduce processing is a procedure to execute combiner task and reduce function of job Job_i in a scheduling algorithm [37]. STAS re-programs combiner method to accept result of heterogeneous input splits and collects them using multiple threads $Thread_i$. The enhanced reduce function understands combiner result and processes them into heterogeneous reducer container.

The output of combiner method can be obtained as,

$$Combiner_{result} = node_i \left(Map_{InputSplit_n}, Thread_i\right) \quad (19)$$

Reduce processing compiles combiner method result $Combiner_{result}$ and generates an output into storage media of HDFS.

V. EXPERIMENTAL WORK

In order to evaluate STAS, we performed experimental executions over a Hadoop cluster configuration as mentioned in Table-1.

A. ENVIRONMENT

The hardware configuration includes Intel Xeon processor with 8 CPU units, 32GB processing memory and storage

TABLE 1. Hadoop cluster configuration.

Machine	Specifications	No. of VM	
Intel Xeon E5-2600 v2	8 CPUs, 32GB memory, 1T Disk and 128 GB SSD	3	1 Master Node, 2 Datanodes
Intel core i5	4 Core, 16GB memory, 1T Disk and 128 GB SSD	2	2 Datanodes
Hadoop	Hadoop-2.7.2 (stable)		
Virtual Machine Management	VirtualBox 5.0.16		

media i.e. 128GB Samsung SSD, 1TB Seagate hard disk drive and tmpfs [38] utility for RAM_DISK storage processing. Additionally, we have used Intel Core i5 processor with 4 cores, 16GB processing memory and storage media i.e. 128GB Samsung SSD, 500GB hard disk drive and tmpfs utility as RAM_DISK storage processing. The virtual environment includes Virtualbox 5.0.16 over 5 virtual machine configurations as elaborated in Table-2.

TABLE 2. Virtual machines configuration over Hadoop cluster.

Node	CPU	Memory	Disk	Configuration
Master Node	6	16 GB	HDD,SSD,RAM	Intel Xeon
Slave1	2	4GB	HDD,SSD,RAM	Intel Xeon
Slave2	2	4GB	HDD,SSD,RAM	Intel Core i5
Slave3	2	4GB	HDD,SSD,RAM	Intel Core i5
Slave4	2	4GB	HDD,SSD,RAM	Intel Core i5

B. EXPERIMENTAL DATASET

The dataset used to process experimental work includes:

1) 300 SSD tag Wordcount Jobs
2) 300 DISK tag Wordcount Jobs
3) 300 RAM tag Wordcount Jobs
4) 300 SSD tag Grep Jobs
5) 300 DISK tag Grep Jobs
6) 300 RAM tag Grep Jobs
7) 300 SSD tag Terasort Jobs
8) 300 DISK tag Terasort Jobs
9) 300 RAM tag Terasort Jobs
10) 100 GB SSD Container (50GB Mapper, 50GB Reducer)
11) 100 GB Disk Container (50GB Mapper, 50GB Reducer)
12) 50 GB Disk associated tmpfs utility RAM Container (25GB Mapper, 25GB Reducer)
13) 1 text file of size 1GB (source dataset)

C. EXPERIMENTAL RESULTS

The experiments conducted to evaluate STAS are as follows:

1) Heterogeneous Queue Container management,
2) STAS Manager,
3) HDFS file management,
4) Map processing evaluation,
5) Reduce processing evaluation and
6) Comparative analysis.

TABLE 3. Heterogeneous tag job parsing into shared-queues.

Heterogeneous Tag Jobs	Heterogeneous Queue Container (HQC)		
MapReduce (2700 jobs)	Queue SSD	Queue RAM	Queue DISK
Wordcount	300	300	300
Grep	300	300	300
Terasort	300	300	300

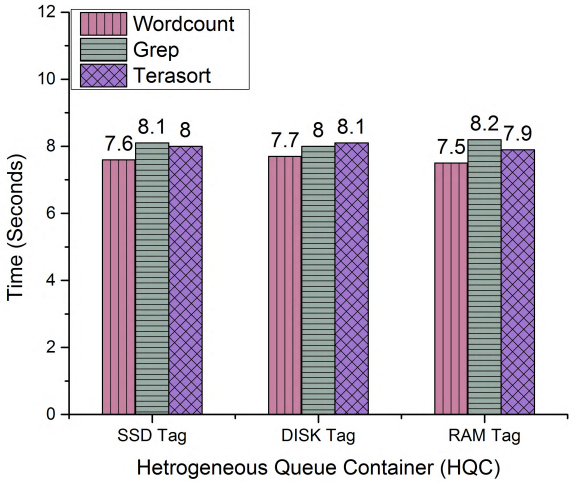

FIGURE 5. Tag Job Parsing into shared-queues of HQC.

1) HETEROGENEOUS QUEUE CONTAINER MANAGEMENT

Heterogeneous queue container is a shell that manages I/O operations of three buffer shared-queues for *DISK*, *SSD* and *RAM*. It contains environment variables and configuration methods to read, write and append shared-queue elements. The buffer shared-queues are formulated through queue management parameters i.e. queue capacity, maximum capacity, job limit, memory allocation and virtual core allocation. The shared-queue $Queue_{RAM}$ contains 20% of queue capacity with job limit set to 1000 and virtual core percentile of 20%. Similarly, shared-queue $Queue_{SSD}$ consists of 30% of queue capacity with job limit set to 1000 and virtual core percentile of 30%. In the same way, shared-queue $Queue_{DISK}$ contains 50% of queue capacity with job limit set to 1000 and virtual core percentile of 50%. Heterogeneous queue container receives a job Job_i and parses it to respective shared-queues buffers with a processing time in seconds. STAS submits 2700 MapReduce tag jobs into the container and evaluates number of Wordcount, Grep and Terasort tag jobs into heterogeneous shared-queue buffers as shown in Table-3. The tag jobs consume a parsing time of 7.6, 8.1 and 8 seconds into shared-queue $Queue_{SSD}$, 7.7, 8 and 8 seconds into shared-queue $Queue_{DISK}$ and 7.5, 8.2 and 7.9 seconds into shared-queue $Queue_{SSD}$ averagely as shown in Fig.5.

2) STAS MANAGER

STAS manager schedules MapReduce job Job_i in three steps as follows:

1) STAS configuration file,
2) HQC shared-queues parameters,

TABLE 4. STAS manager scheduling tag jobs into heterogeneous containers.

STAS Scheduler	SSD Container		RAM Container		DISK Container	
2700 Tag Jobs	Mapper	Reducer	Mapper	Reducer	Mapper	Reducer
Wordcount (900 Jobs)	112	29	96	26	147	34
Grep (900 Jobs)	118	25	93	31	151	32
Terasort (900 Jobs)	108	34	92	29	145	37

3) Messaging event handler,

Initially, STAS configuration file collects scheduling parameters i.e. resource percentile, number of tag jobs, node locality delay and shared-queue administrative configuration-set such as job submission, task processing and user limits. In the next step, STAS invokes *initScheduler* method to parse configuration file and adopts heterogeneous container shared-queues parameters i.e. capacity ($Queue_{SSD}$, $Queue_{DISK}$, $Queue_{RAM}$), maxCapacity(shared-queues), userLimit, maxTagJobs, ResourcePerShared-queue, minimumAllocationFactor, numContainers, state, ACL-Administrator-shared-queues and nodeLocalityDelay. The third step invokes a method *Dispatcher*, which handles event messaging in hierarchical order i.e. (i) add a node, (ii) activate containers, (iii) update resource manager, (iv) add a tag job from HQC, (v) assign job processing id, (vi) remove a job, (vii) refresh containers and (viii) report result.

The performance metrics of scheduling tag job Job_i are:

1) Optimal usage of memory and core percentile for scheduling of map tasks into heterogeneous mapper container,
2) Optimal usage of memory and core percentile for scheduling of reduce tasks into heterogeneous reducer container.

STAS schedules MapReduce tag job into two phases i.e. (i) map task scheduling and (ii)reduce task scheduling. The map task receives an input of tag job Job_i from shared-queues and schedules into matching tag container i.e. SSD, RAM and DISK. We observe that, STAS manager schedules 2700 tag jobs and effectively reduces 53.1% and 31.25% execution time through mapper containers of RAM and SSD media. Moreover, STAS manager schedules resultant map tasks and reduces 30.76% and 17.24% execution time through reducer containers of RAM and SSD media compared to default approach [39], as shown in Table-4.

3) HDFS TAG JOB MANAGEMENT

For this experiment, HDFS stores Amazon products dataset file into root directory [40]. The file system invokes *initBlock* method to transform source file with a size parameter of block equals to 256 MB. Thus, HDFS generates 4 blocks of the file into DISK volume and invokes replica method *initReplica* with storage media parameters i.e. (1, *SSD*), (1, *RAM*). This replicates original blocks of the source file into storage media of SSD and RAM. Therefore, when STAS processes a tag job Job_i, the file system receives input split tasks of tag job and executes map function over data block addresses into

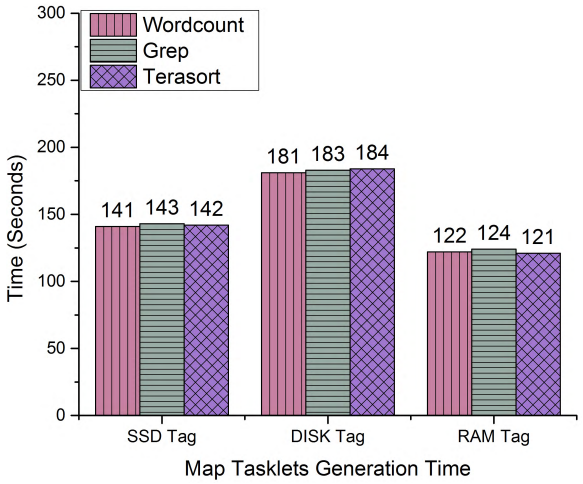

FIGURE 6. Generation of Input split tasks.

heterogeneous storage media. The input split tasks of map function consume page memory of heterogeneous storage media. Therefore, we observe that Wordcount tag job produces input split tasks in 141, 181, 122 seconds over SSD, DISK and RAM. Similarly, we evaluate that Grep tag job produces input split tasks in 143, 183, 124 seconds over SSD, DISK and RAM. In the same way, we find that Terasort tag job produces input split tasks in 142, 184, 121 seconds over SSD, DISK and RAM as shown in Fig.6. This reduces a processing capacity and memory usage of 48.36% and 28.41% at generating input split map tasks compared to homogeneous input split tasks of map function.

4) MAP PROCESSING EVALUATION

Map processing performs execution of input splits programs through a method *mapDaemon*. Namenode processes an equal number of *mapDaemon* methods to the count of source data blocks [41]. Therefore, we observe a set of four *mapDaemon* methods i.e. Map-1, Map-2, Map-3 and Map-4 for handling the execution of map processing. The *mapDaemon* method contains functional subroutines that require execution parameters i.e. *fetchBlockAddress*, *fetchStorageMedia*, *container*$(n, mediaType)$, maximum capacity and memory to process map function. Thus, *mapDaemon* method processes input split programs over heterogeneous mapper containers and generates an output of map tasks into heterogeneous storage-tier of HDFS.

STAS processes single Wordcount, Grep and Terasort input splits through *mapDaemon* methods and observes that

TABLE 5. Map processing result.

	Wordcount						Grep						Terasort					
	Core %			Memory			Core %			Memory			Core %			Memory		
Map Processing	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK
Map-1	52.61	71.83	32.87	9.2	8.7	9.1	53.62	71.09	31.41	9.6	8.4	9.7	52.84	72.42	32.72	9.8	8.8	9.9
Map-2	54.92	72.19	33.19	9.1	8.5	9.4	54.08	72.48	32.95	9.5	8.7	9.6	53.28	72.34	33.69	9.7	8.9	9.8
Map-3	53.54	71.14	32.26	9.2	8.8	9.4	52.89	71.27	31.79	9.2	8.9	9.4	53.77	71.59	31.49	9.8	8.7	9.9
Map-4	54.93	73.04	34.81	9.4	8.7	9.3	53.67	73.38	33.68	9.4	8.6	9.8	54.34	73.48	33.82	9.6	8.9	9.7

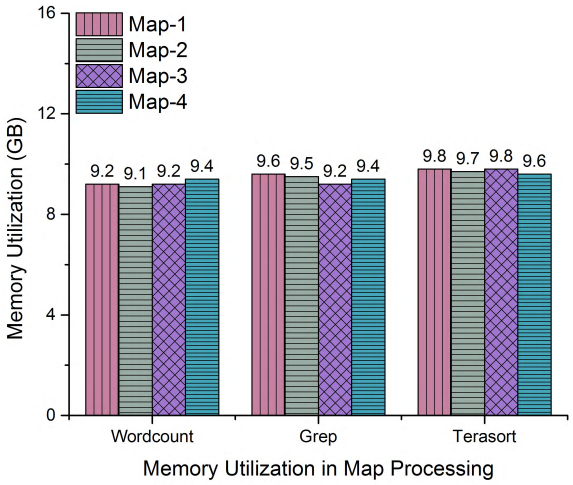

FIGURE 7. Memory usage into heterogeneous mapper container.

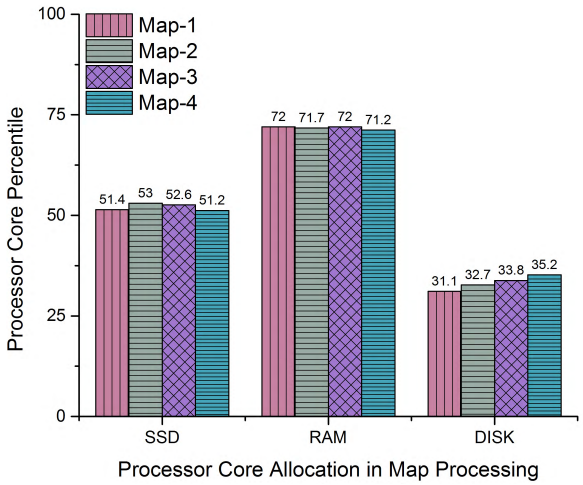

FIGURE 8. Computing capacity consumption into heterogeneous mapper container.

Map-1 consumes 9.2, 9.6 and 9.8 GB of memory, Map-2 uses 9.1, 9.5 and 9.7 GB of memory, Map-3 utilizes 9.2, 9.2 and 9.8 GB of memory and Map-4 consumes 9.4, 9.4 and 9.6 GB of memory over heterogeneous mapper container as shown in Fig.7. Furthermore, Map-1 consumes 51.4%, 72% and 31.1% of core capacity, Map-2 uses 53%, 71.7% and 32.7% of core capacity, Map-3 utilizes 52.6%, 72% and 33.8% of core capacity and Map-4 consumes 51.2%, 71.2% and 35.2% of core capacity over heterogeneous mapper containers as shown in Fig.8. This shows that input splits of single tag job Job_i consumes 54.23% and 37.52% less computing capacity and memory in RAM and SSD containers than DISK container at core percentile configuration of $Core_{\%}$.

In order to observe rigorous performance, STAS processes 2700 Wordcount, Grep and Terasort input splits through *mapDaemon* methods and records average performance of 10800 input split programs consuming computing capacity and memory usage over heterogeneous mapper containers as shown in Table-5. This depicts that 10800 input splits of 2700 tag jobs Job_i consumes 52.19% and 36.27% less computing capacity and memory in RAM and SSD containers than DISK container through core percentile configuration of $Core_{\%}$.

5) REDUCE PROCESSING EVALUATION

Reduce processing collects output of map input split programs through *combiner* function and summarizes the collection using *reduceDaemon* method [42]. Namenode assigns single instance of *reduceDaemon* method that generates n number of threads equivalent to count of input split programs. Thus, we observe single method *reduceDaemon* for executing multiple threads of combiner method and reduce processing. The *reduceDaemon* method consists of subroutines that requires a set of execution parameters i.e. *accumulatorLocator*, *threadPerAccumulator*, *combiner* (*thread*, *accumulator*), *container*(n, *mediaType*),

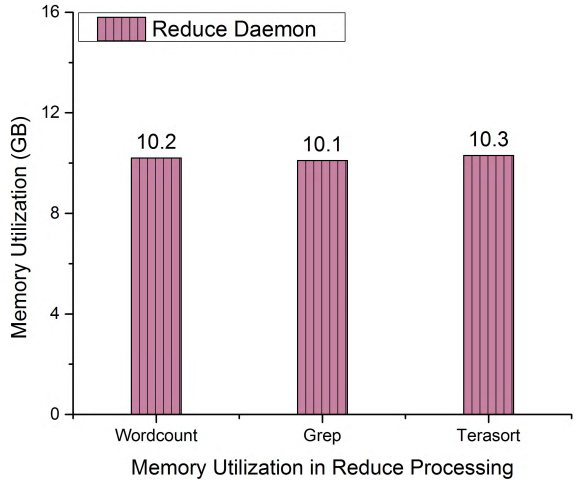

FIGURE 9. Memory usage into heterogeneous reducer container.

TABLE 6. Reduce processing result.

	Wordcount						Grep						Terasort					
	Core %			Memory			Core %			Memory			Core %			Memory		
Reduce Processing	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK
Reduce	59.61	80.95	35.37	10.1	10.3	10.2	59.73	81.38	35.18	10.3	10.4	10.2	59.95	80.47	35.16	10.4	10.5	10.3

TABLE 7. MapReduce tag job Job_i processing into Hadoop cluster.

$Jobs_n$	Job_i	$Bandwidth_{rate}$	$Mapper_{rate}$	$Mapper_{Container}$	$Reducer_{Container}$	$Memory_{Total}$	Map_{Task}	$Reduce_{Task}$	$HDFS_{Write}$	$Core_{\%}$
300	SSD (Wordcount)	40.94 Mbps	75.3MB/Each Job	8.7 GB	9 GB	9.7 GB	1200	300	11.4 GB	58.27%
300	SSD (Grep)	41.31 Mbps	75.9MB/Each Job	8.1 GB	8.4 GB	9.8 GB	1200	300	10.8 GB	59.41%
300	SSD (Terasort)	41.94 Mbps	76.4MB/Each Job	9.2 GB	9.5 GB	9.9 GB	1200	300	11.9 GB	58.93%
300	RAM (Wordcount)	43.82 Mbps	75.85MB/Each Job	8.1 GB	8.4 GB	9.8 GB	1200	300	10.8 GB	78.38%
300	RAM (Grep)	42.64 Mbps	76.25MB/Each Job	8.3 GB	8.7 GB	10.1 GB	1200	300	11.1 GB	78.29%
300	RAM (Terasort)	44.73 Mbps	76.4MB/Each Job	8.6 GB	8.9 GB	10.05 GB	1200	300	11.3 GB	78.42%
300	DISK (Wordcount)	45.29 Mbps	76.38MB/Each Job	8.9 GB	9.3 GB	9.9 GB	1200	300	11.7 GB	35.5%
300	DISK (Grep)	46.39 Mbps	78.96MB/Each Job	9.1 GB	9.4 GB	10.05 GB	1200	300	11.8 GB	35.82%
300	DISK (Terasort)	48.63 Mbps	79.5MB/Each Job	9.2 GB	9.6 GB	10.1 GB	1200	300	12.05 GB	35.95%

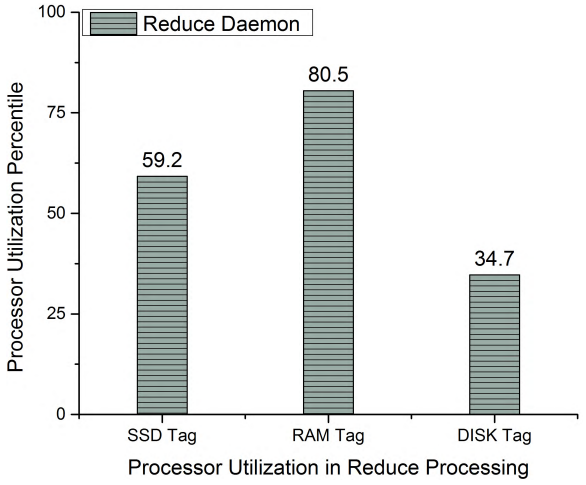

FIGURE 10. Computing capacity consumption into heterogeneous reducer container.

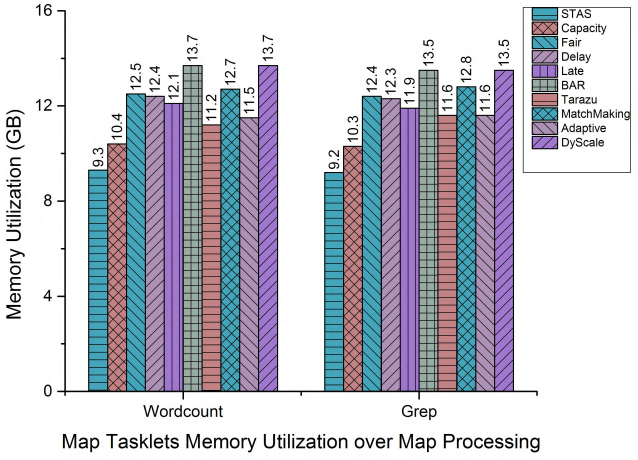

FIGURE 11. Memory usage into heterogeneous mapper container of Schedulers.

maximum capacity and memory to process reduce function. Thus, *reduceDaemon* method processes output of *mapDaemon* over heterogeneous reducer containers and generates an output of tag job Job_i into heterogeneous storage media of HDFS.

STAS schedules *mapDaemon* output of Wordcount, Grep and Terasort jobs and observes that *reduceDaemon* method consumes 10.2, 10.1 and 10.3 GB of memory over heterogeneous reducer container as shown in Fig.9. Moreover, *reduceDaemon* method consumes 59.2%, 80.5% and 34.7% of core capacity over heterogeneous reducer containers as shown in Fig.10. This shows that multiple threads of instance *reduceDaemon* consumes 54.23% and 37.52% less computing capacity and memory in RAM and SSD containers than DISK container at core percentile configuration of $Core_{\%}$.

In order to evaluate precise performance, STAS schedules *mapDaemon* output of 2700 Wordcount, Grep and Terasort jobs through *reduceDaemon* method and records an average performance of 10800 threads consuming computing capacity and memory usage over heterogeneous reducer containers as shown in Table-6. This shows that 10800 threads of reduce processing consumes 42.71% and 55.73% less computing capacity and memory in SSD and RAM containers than DISK container through core percentile configuration of $Core_{\%}$.

The map and reduce processing are parts of MapReduce paradigm. Therefore, we summarize evaluation reports and observe individual performance of MapReduce tag jobs Job_i as shown in Table-7.

6) COMPARATIVE ANALYSIS

We equip enlist schedulers i.e. Capacity, Fair, Delay, Late, BAR, Tarazu, Matchmaking, Adaptive, and DyScale with a configuration of executing jobs Job_i over heterogeneous MapReduce containers. The comparison metrics of scheduling tag job Job_i through schedulers are:

1) Optimal usage of memory and core percentile for scheduling input splits into heterogeneous mapper container,

TABLE 8. Resources consumption into heterogeneous mapper container of Schedulers.

	Wordcount						Grep						Terasort					
	Core %			Memory			Core %			Memory			Core %			Memory		
Scheduler	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK
STAS	51.7	72.8	32.5	9.1	9.2	9.05	52.4	72.4	32.9	9.4	9.2	9.1	51.9	72.6	32.6	9.4	9.1	9.5
Capacity	75.7	86.3	61.5	10.4	10.5	10.3	76.1	86.6	61.2	10.6	10.4	10.8	75.8	86.1	61.6	10.3	10.7	10.25
Fair	82.7	89.1	74.6	12.5	12.7	12.4	82.4	89.4	74.9	12.9	12.6	12.8	82.9	89.7	74.1	12.8	12.3	12.7
Delay	84.9	88.3	76.3	12.4	12.3	12.6	84.7	88.2	76.1	12.5	12.8	12.9	84.2	88.6	76.8	12.9	12.4	12.6
Late	84.8	89.5	77.4	12.3	12.1	12.7	84.4	89.4	77.8	12.5	12.9	12.7	84.2	89.2	77.3	12.1	12.8	12.3
BAR	86.3	91.4	81.5	13.8	13.9	13.4	86.6	91.2	81.9	13.6	13.3	13.1	86.2	91.8	81.8	13.9	13.4	13.6
Tarazu	77.4	86.9	65.2	11.4	11.2	11.5	77.5	86.7	66.5	11.9	11.3	11.2	77.8	86.6	65.9	11.4	11.8	11.28
Matchmaking	85.8	89.3	78.7	12.7	12.9	12.6	85.6	89.8	77.9	12.9	13.05	12.8	85.3	89.5	79.3	12.8	12.94	12.5
Adaptive	83.5	90.4	77.2	11.6	11.8	11.9	83.9	90.2	77.8	11.9	11.6	11.92	83.6	90.7	77.3	12.1	11.8	11.9
DyScale	87.2	91.3	84.8	13.8	13.6	13.9	87.6	91.7	85.3	14.1	13.7	14.02	87.5	91.6	80.04	14.3	14.1	13.9

TABLE 9. Resources consumption into heterogeneous reducer container of Schedulers.

	Wordcount						Grep						Terasort					
	Core %			Memory			Core %			Memory			Core %			Memory		
Scheduler	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK	SSD	RAM	DISK
STAS	54.2	74.3	34.2	10.5	10.7	10.3	54.7	74.2	34.7	10.8	10.6	11.1	54.3	74.7	34.4	10.9	10.7	10.5
Capacity	77.8	88.6	63.6	11.6	11.3	11.2	78.3	88.3	63.7	11.9	11.5	12.2	78.1	88.1	63.1	11.6	11.9	11.8
Fair	84.5	91.8	75.7	11.9	11.8	11.4	84.8	91.5	75.3	12.3	12.9	12.6	84.6	91.2	75.2	12.8	12.5	12.8
Delay	86.7	90.7	77.8	13.4	13.7	13.6	86.2	90.1	77.5	13.6	13.5	13.3	86.4	90.4	77.6	13.7	13.9	13.4
Late	86.2	91.4	78.5	13.7	13.4	13.6	86.9	92.4	78.2	13.9	13.3	13.5	86.4	91.9	78.6	13.6	13.2	13.5
BAR	88.4	93.5	82.7	14.6	14.8	14.3	88.1	93.7	82.6	14.4	14.8	14.6	88.5	93.8	82.3	14.5	14.7	14.2
Tarazu	79.2	88.3	67.1	11.6	11.7	11.4	79.3	88.4	67.4	11.8	11.6	11.2	79.5	88.7	67.8	11.9	11.7	11.4
Matchmaking	87.5	91.8	79.6	13.5	13.8	13.2	87.1	91.7	79.8	13.7	13.8	13.5	87.4	91.5	79.4	13.3	13.6	13.7
Adaptive	85.9	92.6	78.4	12.4	12.6	12.7	85.5	92.5	78.8	12.3	12.6	12.1	85.4	92.7	78.2	12.9	12.7	12.6
DyScale	89.3	93.5	85.8	13.9	13.2	13.6	89.4	93.8	85.3	13.8	14.2	13.9	89.5	93.6	85.5	14.9	14.4	14.2

2) Optimal usage of memory and core percentile for scheduling of reduce tasks into heterogeneous reducer container.

We schedule single Wordcount and Grep job through enlist schedulers and evaluate that *mapDaemon* method consumes 11.89%, 34.59%, 33.51%, 29.72%, 47.02%, 23.24%, 37.83%, 24.86%, and 47% less memory of STAS than Capacity, Fair, Delay, Late, BAR, Tarazu, Matchmaking, Adaptive and DyScale schedulers respectively, as shown in Fig.11. Furthermore, we found that the same Wordcount and Grep jobs consume 46.65%, 56.16%, 57.03%, 57.95%, 59.72%, 50%, 58.29%, 57.8%, and 61.2% less core percentile of STAS than Capacity, Fair, Delay, Late, BAR, Tarazu, Matchmaking, Adaptive, and DyScale schedulers respectively, as shown in Fig.12.

In order to observe accuracy in performance, schedulers process 2700 Wordcount, Grep and Terasort input splits through *mapDaemon* methods and records average performance of 10800 input split programs consuming computing capacity and memory usage over heterogeneous mapper containers as shown in Table-8. This shows that 10800 input splits of 2700 tag jobs Job_i consume 29.48% and 38.62% less computing capacity and memory through STAS than enlist schedulers having core percentile configuration of $Core_{\%}$.

We schedule *mapDaemon* output of Wordcount and Grep jobs through enlist schedulers and evaluate that *reduceDaemon* method consumes 13.79%, 16.74%, 30.04%, 29.06%, 39.9%, 16.25%, 31.52%, 26.1%, and 36.45% less memory of STAS than Capacity, Fair, Delay, Late, BAR, Tarazu, Matchmaking, Adaptive and DyScale schedulers

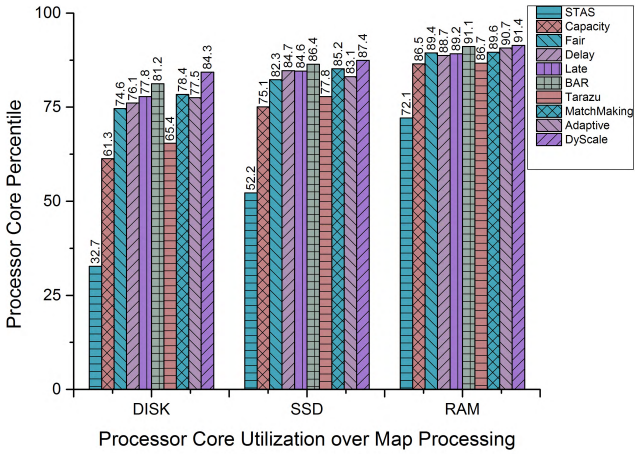

FIGURE 12. Computing capacity consumption into heterogeneous mapper container of Schedulers.

respectively, as shown in Fig.13. Moreover, we observe that the same *reduceDaemon* method over *mapDaemon* output of Wordcount and Grep jobs consume 21.76%, 29.19%, 30.09%, 30.68%, 32.58%, 24.14%, 31.12%, 30.59%, and 33.71% less core percentile of STAS than Capacity, Fair, Delay, Late, BAR, Tarazu, Matchmaking, Adaptive and DyScale schedulers respectively, as shown in Fig.14.

In order to evaluate accuracy in performance, schedulers process *mapDaemon* output of 2700 Wordcount, Grep and Terasort jobs through *reduceDaemon* method and record an average performance of 10800 threads consuming computing

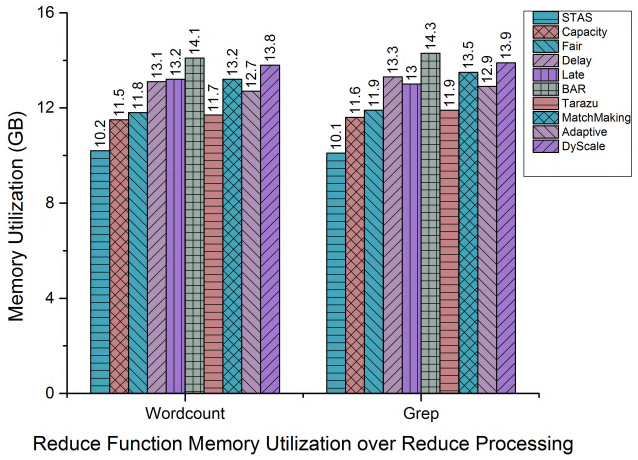

FIGURE 13. Memory usage into heterogeneous reducer container of Schedulers.

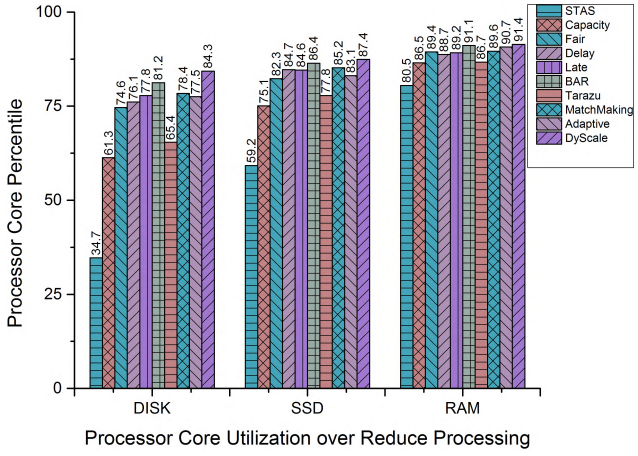

FIGURE 14. Computing capacity consumption into heterogeneous reducer container of Schedulers.

capacity and memory usage over heterogeneous reducer containers as shown in Table-9. This shows that 10800 threads of reduce processing consumes 31.41% and 43.79% less computing capacity and memory through STAS than enlist schedulers through core percentile configuration of $Core_{\%}$.

VI. CONCLUSION

This paper proposes a resource efficient scheduler that processes MapReduce jobs into heterogeneous storage containers. STAS introduces the use of tag jobs, heterogeneous shared-queues, heterogeneity-aware STAS manager and use of heterogeneous storage media as container volume. STAS reduces processing latency of locating and pairing dataset into heterogeneous storage media and optimizes the consumption of resources in Hadoop cluster. In future, we focus to work over scheduling issues of multi-homing nodes connected through mesh topology in Hadoop cluster.

REFERENCES

[1] (2014). *Welcome to Apache Hadoop!*, accessed on Dec. 8, 2016. [Online]. Available: http://hadoop.apache.org/
[2] M. Technologies. (2016). *Featured Customers*, accessed on Dec. 8, 2016. [Online]. Available: https://www.mapr.com/
[3] Cloudera. (2016). *The Modern Platform for Data Management and Analytics*, accessed on Dec. 8, 2016. [Online]. Available: http://www.cloudera.com/
[4] F. Bajaber, R. Elshawi, O. Batarfi, A. Altalhi, A. Barnawi, and S. Sakr, "Big data 2.0 processing systems: Taxonomy and open challenges," *J. Grid Comput.*, vol. 14, no. 3, pp. 379–405, Jun. 2016.
[5] (2015). *Apache Hadoop 2.7.1 Apache Hadoop NextGen MapReduce (YARN)*, accessed on Dec. 8, 2016. [Online]. Available: https://hadoop.apache.org/docs/r2.7.1/hadoop-yarn/hadoop-yarn-site/YARN.html
[6] (2016). *Apache Hadoop 2.7.3 HDFS Architecture*, accessed on Dec. 8, 2016. [Online]. Available: http://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
[7] M. Senthilkumar and P. Ilango, "A survey on job scheduling in big data," *J. Inst. Inf. Commun. Technol. Bulgarian Acad. Sci.*, vol. 16, no. 3, pp. 35–51, Jan. 2016.
[8] B. J. Mathiya and V. L. Desai, "Apache Hadoop yarn parameter configuration challenges and optimization," in *Proc. Int. Conf. Soft-Comput. Netw. Secur. (ICSNS)*, Feb. 2015, pp. 1–6.
[9] I. A. Hashem, N. B. Anuar, A. Gani, I. Yaqoob, F. Xia, and S. U. Khan, "MapReduce: Review and open challenges," *Scientometrics*, vol. 109, no. 1, pp. 389–422, 2016.
[10] S. Singh, G. Rakhi, and P. K. Mishra. (2017). "Review of Apriori based algorithms on MapReduce framework." [Online]. Available: https://arxiv.org/abs/1702.06284
[11] *Heterogeneous Storage*, accessed on Dec. 8, 2016. [Online]. Available: https://issues.apache.org/jira/browse/HDFS-2832
[12] N. M. F. Qureshi and D. R. Shin, "RDP : A storage-tier-aware robust data placement strategy for Hadoop in a Cloud-based heterogeneous environment," *KSII Trans. Internet Inf. Syst.*, vol. 10, no. 9, pp. 4063–4086, Sep. 2016.
[13] D. Park, K. Kang, J. Hong, and Y. Cho, "An efficient Hadoop data replication method design for heterogeneous clusters," in *Proc. 31st Annu. ACM Symp. Appl. Comput.-(SAC)*, 2016, pp. 2182–2184.
[14] J. Mace, P. Bodik, R. Fonseca, and M. Musuvathi, "Retro: Targeted resource management in multi-tenant distributed systems," in *Proc. NSDI*, 2015, pp. 589–603.
[15] S. Shaikh and D. Vora, "YARN versus MapReduce—A comparative study," in *Proc. 3rd Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, 2016, pp. 1294–1297.
[16] R. Gu *et al.*, "SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters," *J. Parallel Distrib. Comput.*, vol. 74, no. 3, pp. 2166–2179, Mar. 2014.
[17] R. S. Pakize, "A comprehensive view of Hadoop MapReduce scheduling algorithms," *Int. J. Comput. Netw. Commun. Secur.*, vol. 2, no. 9, pp. 308–317, 2014.
[18] (2016). *Apache Hadoop 2.7.2 Hadoop: Fair Scheduler*, accessed on Dec. 8, 2016. [Online]. Available: http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/FairScheduler.html
[19] (2016). *Apache Hadoop 2.7.2 Hadoop: Capacity Scheduler*, accessed on Dec. 8, 2016. [Online]. Available: http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html
[20] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 369–392, Feb. 2014.
[21] D. Cheng, J. Rao, Y. Guo, C. Jiang, and X. Zhou, "Improving performance of heterogeneous MapReduce clusters with adaptive task tuning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 774–786, Mar. 2017.
[22] N. S. Islam, M. Wasi-ur-Rahman, X. Lu, and D. K. Panda, "Efficient data access strategies for Hadoop and spark on HPC cluster with heterogeneous storage," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 223–232.
[23] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.
[24] X. Wang, Y. Wang, and H. Zhu, "Energy-efficient multi-job scheduling model for cloud computing and its genetic algorithm," *Math. Problems Eng.*, vol. 2012, Oct. 2012, Art. no. 589243.

[25] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new MapReduce scheduling technique," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2011, pp. 40–47.
[26] M. Zaharia *et al.*, "Improving MapReduce performance in heterogeneous environments," in *Proc. OSDI*, vol. 8. 2008, p. 4.
[27] F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni, "DyScale: A MapReduce job scheduler for heterogeneous multicore processors," *IEEE Trans. Cloud Comput.*, vol. 5, no. 2, pp. 317–330, Apr. 2017.
[28] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 1, p. 61, Apr. 2012.
[29] J. Polo, Y. Becerra, D. Carrera, J. Torres, E. Ayguade, and M. Steinder, "Adaptive MapReduce scheduling in shared environments," in *Proc. 14th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2014, pp. 61–70.
[30] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "BAR: An efficient data locality driven task scheduling algorithm for cloud computing," in *Proc. 11th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2011, pp. 295–304.
[31] K. R. Krish, A. Anwar, and A. R. Butt, "hatS: A heterogeneity-aware tiered storage for Hadoop," in *Proc. 14th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, Chicago, IL, USA, May 2014, pp. 502–511.
[32] N. S. Islam, X. Lu, M. Wasi-ur-Rahman, D. Shankar, and D. K. Panda, "Triple-H: A hybrid approach to accelerate HDFS on HPC clusters with heterogeneous storage architecture," in *Proc. 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2015, pp. 101–110.
[33] M. Wasi-ur-Rahman, X. Lu, N. S. Islam, R. Rajachandrasekar, and D. K. Panda, "High-performance design of YARN MapReduce on modern HPC clusters with Lustre and RDMA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2015, pp. 291–300.
[34] W. Hu *et al.*, "Multiple-job optimization in MapReduce for heterogeneous workloads," in *Proc. 6th Int. Conf. Semantics, Knowl. Grids*, Nov. 2010, pp. 135–140.
[35] D. Cheng, X. Zhou, P. Lama, J. Wu, and C. Jiang, "Cross-platform resource scheduling for spark and MapReduce on YARN," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1341–1353, 2017.
[36] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[37] R. Chaiken *et al.*, "SCOPE: Easy and efficient parallel processing of massive data sets," *VLDB Endowment*, vol. 1, no. 2, pp. 1265–1276, Aug. 2008.
[38] (2011). *An in-Place Memory RAM Mirror Tool*, accessed on Dec. 8, 2016. [Online]. Available: https://www.krenger.ch/blog/linux-ramdisk-with-tmpfs/
[39] K. R. Krish, M. S. Iqbal, and A. R. Butt, "VENU: Orchestrating SSDs in hadoop storage," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Oct. 2014, pp. 207–212.
[40] *Amazon Product Dataset*, accessed on Dec. 8, 2016. [Online]. Available: https://snap.stanford.edu/data/bigdata/amazon/amazon-meta.txt.gz
[41] Y. V. Lokeswari and S. G. Jacob, "A comparative study on parallel data mining Algorithms using Hadoop map reduce," in *Proc. 2nd Int. Conf. Inf. Commun. Technol. Competitive Strategies-(ICTCS)*, 2016, p. 143.
[42] A. B. Patel, M. Birla, and U. Nair, "Addressing big data problem using Hadoop and map reduce," in *Proc. Nirma Univ. Int. Conf. Eng. (NUiCONE)*, Dec. 2012, pp. 1–5.

NAWAB MUHAMMAD FASEEH QURESHI received the B.E. degree in software engineering and the M.E. degree in information technology from the Mehran University of Engineering and Technology, Pakistan, in 2006 and 2013, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Sungkyunkwan University, South Korea. His research interests include Big Data platform and cloud computing.

DONG RYEOL SHIN received the B.S. degree in electrical engineering from the Sungkyunkwan University in 1980, the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology in 1982, and the Ph.D. degree in electrical engineering from the Georgia Institute of Technology, USA, in 1992. From 1992 to 1994, he was with Samsung Data Systems, South Korea, where he was involved in the research of intelligent transportation systems. Since 1994, he has been with the Department of Computer Science and Engineering, Sungkyunkwan University, where he is currently a Full Professor with the Network Research Group. His current research interests lie in the areas of mobile network, ubiquitous computing, cloud computing, and bioinformatics. He is actively involved in the security of vehicular area networks, and the implementation and analysis of Big Data platform, applicable to 3-D image processing of robotic arms.

ISMA FARAH SIDDIQUI received the B.E. degree in software engineering and the M.E degree in information technology from the Mehran University of Engineering and Technology, Pakistan, in 2006 and 2008, respectively. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Hanyang University ERICA, South Korea, funded by Higher Education Commission, Pakistan. Since 2006, she has been with the Department of Software Engineering, Mehran University of Engineering and Technology, where she is currently designated as an Assistant Professor. Her research interests include software engineering, smart environment, semantic Web, IoT, and Big Data.

BHAWANI SHANKAR CHOWDHRY received the Ph.D. degree from the School of Electronics and Computer Science, University of Southampton, U.K., in 1990. He is currently a Full Professor and the Dean Faculty of Electrical Electronics and Computer Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan. He is having teaching and research experience of more than 30 years. He has the honor of being one of the editors of several books *Wireless Networks, Information Processing and Systems* (CCIS 20), *Emerging Trends and Applications in Information Communication Technologies* (CCIS 281), *Wireless Sensor Networks for Developing Countries* (CCIS 366), and *Communication Technologies, Information Security and Sustainable Development* (CCIS 414), published by Springer Verlag, Germany. He has also been serving as a Guest Editor of *Wireless Personal Communications*, which is a Springer International Journal. He has produced more than 13 Ph.D. degrees and supervised more than 50 M.Phil./master's Theses in the area of ICT. His list of research publication crosses to over 60 in national and international journals, IEEE and ACM proceedings. Also, he has Chaired Technical Sessions in USA, U.K., China, UAE, Italy, Sweden, Finland, Switzerland, Pakistan, Denmark, and Belgium. He is a member of various professional bodies including: the Chairman IEEE Karachi Section, Region10 Asia/Pacific, Fellow IEP, Fellow IEEEP, Senior Member, IEEE Inc., USA, SM ACM Inc., USA. He is a Lead Person at MUET of several EU funded Erasmus Mundus Program, including Mobility for Life, StrongTies, INTACT, and LEADERS. He has organized several International Conferences, including IMTIC08, IMTIC12, IMTIC13, IMTIC15, WSN4DC13, IEEE SCONEST, IEEE PSGWC13, and the Track Chair in Global Wireless Summit (GWS 2014), and the Chief Organizer of GCWOC'16 in Málaga, Spain.