Wireless Pers Commun (2017) 95:2709–2733
DOI 10.1007/s11277-017-3953-5

Performance Improvement of MapReduce for Heterogeneous Clusters Based on Efficient Locality and Replica Aware Scheduling (ELRAS) Strategy

J. V. Bibal Benifa[1] · **Dejey**[1]

Published online: 13 January 2017

Abstract MapReduce is a parallel programming model for processing the data-intensive applications in a cloud environment. The scheduler greatly influences the performance of MapReduce model while utilized in heterogeneous cluster environment. The dynamic nature of cluster environment and computing workloads affect the execution time and computational resource usage in the scheduling process. Further, data locality is essential for reducing total job execution time, cross-rack communication, and to improve the throughput. In the present work, a scheduling strategy named efficient locality and replica aware scheduling (ELRAS) integrated with an autonomous replication scheme (ARS) is proposed to enhance the data locality and performs consistently in the heterogeneous environment. ARS autonomously decides the data object to be replicated by considering its popularity and removes the replica as it is idle. The proposed approach is validated in a heterogeneous cluster environment with various realistic applications that are IO bound, CPU bound and mixed workloads. ELRAS improves the throughput by a factor about 2 as compared with the existing FIFO and it also yields near optimal data locality, reduce the execution time, and effective utilization of resources. The simplicity of ELRAS algorithm proves its feasibility to adopt for a wide range of applications.

Keywords MapReduce programming model · Data locality · Heterogeneous clusters · Virtualization

1 Introduction

Big data analysis is a process of handling complex data set that is generated by online social networks and scientific research [1, 2], using advanced tools to produce meaningful information in a cloud environment [3]. The huge volume of data are processed using a

✉ J. V. Bibal Benifa
benifa.john@gmail.com

Dejey
dejeytilak@gmail.com

[1] Department of Computer Science and Engineering, Anna University- Regional Campus, Tirunelveli, Tamil Nadu 627007, India

MapReduce programming model [4] that executes large number of parallel task on the cluster of physical machines (PM). MapReduce divides the job into multiple tasks that are executed in parallel manner on data elements placed in various PM in a cluster. Apache Hadoop is an open source software framework that exploits the MapReduce programming model for processing big data in a cloud environment such as Amazon, Oracle, Microsoft, and IBM [5].

A scheduling algorithm influences the performance of MapReduce framework in a computing environment as it manages the large cluster of hardware machines, controls the allocation of data and task on these hardware machines [6]. The scheduler is expected to satisfy various quality metrics such as data locality, fairness, throughput, response time, availability, Energy efficiency, CPU utilization, memory utilization, disk utilization, and security metrics [6]. Here, data locality is one of the important quality metrics and a critical factor that affects the performance of MapReduce clusters [7]. It reduces the cross-rack data movement and improves the throughput as a task is executed in a data-local manner. Fairness is providing equal opportunity to users by sharing the resources among them in a fair manner. For each user, the scheduler must ensure fairness by assigning the required number of computational resources. Throughput involves the number of request (jobs/tasks) completed over a specific period of time and it is achieved by improving the data locality and effective utilization of resources [8]. Effective resource utilization involves improving the resource usage and minimization of the system states being idle in a cluster.

The MapReduce scheduler plays a key role for meeting the quality metrics to confer a consistent performance in the heterogeneous environment. However, designing a scheduler to satisfy all the quality metrics is a challenging and complex process. For example, if a scheduling policy enforces strict data locality constraints then it has to compromise fairness since, a node having the data of jobs may not exist at the head of the queue based on fairness constraints [5]. In a MapReduce model, scheduling is a multi-objective optimization problem that is NP-hard in nature [9]. Further, a scheduler is also expected to be adaptive for allocating the resources based on availability of the system and capacity of PM in the cluster. Hence, an effective locality and replica aware scheduling (ELRAS) strategy is proposed based on the requirements with the following features:

- ELRAS strategy adapts itself for the heterogeneous clusters and achieves higher data locality rates during data intensive computations.
- A data placement strategy is formulated to compute the available space in the nodes dynamically and for positioning the data. Then, the task scheduling algorithm is designed to satisfy the data locality constraints and placing the task in the node that holds the data.
- An autonomous replication scheme (ARS) is proposed and it is used in combination with ELRAS to perform effective replication decision.
- ELRAS strategy satisfies the quality metrics such as execution time, data locality, resource utilization and throughput.

2 Background Research

2.1 Hadoop MapReduce Model

The Hadoop system architecture is illustrated in Fig. 1 and it is based on two important components: Firstly, the MapReduce programming model is utilized for parallel

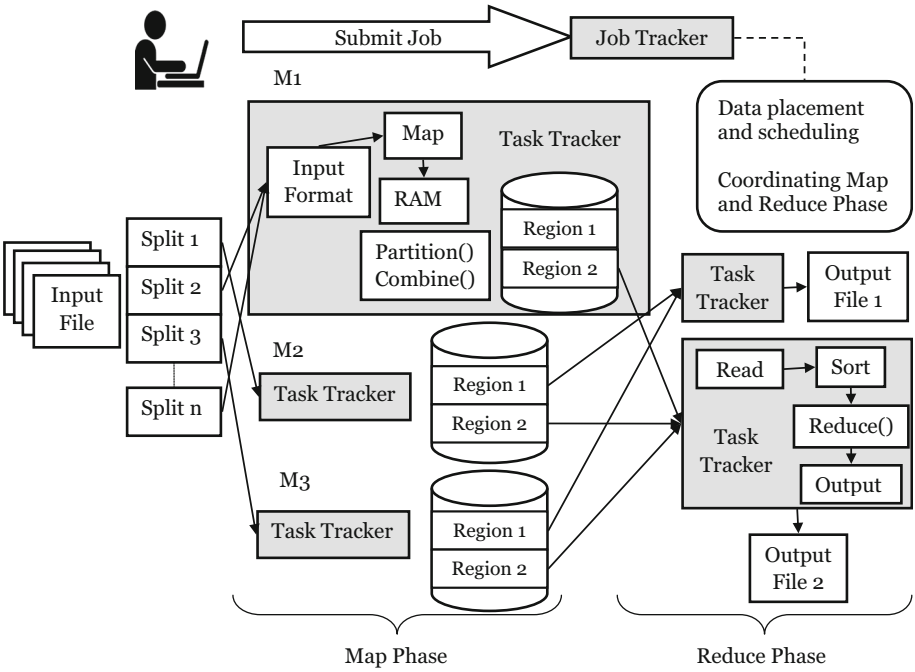


Fig. 1 The overview of Hadoop architecture

computations and then Hadoop Distributed File System for storing the data [10]. The MapReduce framework offers the execution environment for MapReduce jobs and a cluster of task trackers are employed for computations that is controlled by one centralized job tracker. All jobs are submitted to the job tracker and subsequently it is divided into multiple map and reduce tasks. The scheduler in the job tracker allocates the tasks to various task trackers for execution. Hadoop also implements the shuffle service using two phases such as shuffle and reduce tasks. Shuffle phase fetch the map outputs related with a reduce task from several nodes and combine them into one reduce input. An external merge sort algorithm is utilized as the intermediate data is too large to fit in memory.

Hadoop distributed file system (HDFS) has one name node to save the metadata and multiple data nodes and that are used for storing the data blocks. Hadoop is deployed on physical clusters in which typical deployment includes two master nodes (job tracker and name node) and several slave nodes to run the task tracker. Here, a slave node is divided into a fixed number of map and reduce slots, also a slot-based task scheduler is applied for scheduling. During the execution of a MapReduce job, the slave nodes periodically check the availability of task slots and requesting the master node for new tasks.

2.2 Related Work

MapReduce Scheduling is a multi-dimensional problem as it needs to optimize a set of quality metrics in a heterogeneous environment and workload constraints. The data placement, task scheduling and replication mechanism also influence the performance of the schedulers. Related work is presented by focusing the various dimensions of the scheduling strategies.

2.2.1 Scheduling Policies Based on Quality Metrics

Hadoop uses three basic scheduling policies namely FIFO, fair-share and capacity scheduler. The FIFO algorithm [11] gives preference to the workload that arrives earlier and here starvation problem occurs because of small jobs has to wait for longer jobs to complete execution. The fair-share (FS) [12] and capacity scheduler (CS) [13] equally distributes the computing resources among the users to ensure strict fairness. FS allocates one pool to each user in the system, and the resources are shared equally among all the pools. CS employs queues as an alternative method instead of pools, where each queue is allotted to a user and resources. In the recent literatures, several scheduling strategies are proposed by optimizing the basic schedulers or new scheduling strategies based on the combination of various quality metrics as an objective. Data locality is essential for data intensive computations to reduce the data transfer and for improving the throughput [14]. Further, the overall throughput can be improved by reducing the response time during execution. It is a complicated process to ensure a high throughput while adapting to varying workloads and fluctuating resource characteristics.

Earlier, delay scheduling [15] was proposed to deal with the conflict between locality and fairness needed for sharing the resources. According to the fairness constraints, a scheduled job couldn't initiate a local task and further it delays the execution and launches the other task. Recently, Purlieus architecture [16] is used for allocating resources while a MapReduce application is executed in a cloud environment. It also suggests that data placement and virtual machine placement greatly influence the execution time of MapReduce application. The classification and optimization scheduler for Hadoop (COSHH) [17] classifies jobs using a k-means clustering algorithm concerning to the recorded execution times. Then, it determines the allocation matrix for jobs using a linear programming approach by maximizing the resource utilization. A hybrid algorithm [18] maintains the throughput of dynamic MapReduce by various scheduling algorithms for different combinations of workload volumes. During low workload periods, it uses the FIFO policy to achieve lower completion times and FS as the system load is balanced. As the workload volume is high, it uses the COSHH approach to achieve significant fairness level and better average completion time (ACT).

The Quincy scheduling algorithm [19] maintains a balance between data locality and fairness for the incoming workload. Here, a bipartite graph is used to represent nodes, jobs, and tasks while a weight is assigned based on the amount of data transferred for the edge that connects the node and task. Then, a scheduling plan is devised by computing the minimum flow cost in the graph under the fairness constraint. In a baseline scheduler [20], the scheduling decisions are made without any global cluster information and the duplications are made to the idle nodes for load balancing and ensuring locality constraint.

2.2.2 Scheduling Policies Adapting to Heterogeneous Environment

The presence of large number of heterogeneous nodes causes various difficulties for effectively utilizing the resources such as CPU, network I/O, and disk I/O [21]. Reducing the task execution time also significantly improves the resource utilization [22]. The scheduling algorithm focusing resource utilization verifies the available bandwidth of different types of resources on a machine against the requirements of a job before scheduling to minimize busy waiting and resource utilization skewness.

Late scheduler [23] enhances the performance of MapReduce framework by reducing the overhead of speculative task in a heterogeneous environment and also it computes the completion time of an application speculatively. The triple-queue scheduler [24], first allocates the map tasks of a fresh job to an available slot then, using the execution data, the scheduler moves the job to a CPU-bound or a disk IO-bound queue. The scheduler continuously monitors the execution times of tasks to move jobs dynamically from one type of queue to another. This scheme is proven to achieve an overall improvement in throughput.

2.2.3 Replication Mechanism

Data replication is an important optimization methodology to handle large volumes of data thereby replicating the data at different locations and achieves higher data availability. Hierarchical cluster scheduling (HCS) algorithm is proposed for data intensive computations in a grid environment [25]. HCS algorithm is also used in combination with hierarchical replication scheme (HRS) to achieve improvement in the execution time when used for data intensive computations. Here, the author(s) had suggested that an effective replication scheme and a scheduling strategy reduce the overhead and job execution time irrespective of the environment (homogeneous or heterogeneous).

HRS [25] decides the number of replica based on the previously executed data set and it is used with HCS to attain optimal results. bandwidth hierarchy based replication (BHR) [26], [27] tries to increase the necessary data in the same region to fetch the replica easily since, the bandwidth within the region is high. Moreover, the regional popularity of files is computed and the best replica for a job is identified through BHR. Least recently used (LRU) replication mechanism replicates all the files and it removes the file that has been used recently [28]. Some of the related works from the recent literatures are listed in Table 1.

3 Problem Analysis and System Design

Data locality improves the performance of MapReduce framework in heterogeneous clusters by placing the computations in nodes that hold the data blocks. Heterogeneity has an impact on Hadoop scheduler because in a cluster, the node parameter always fluctuates. Hence, the solutions proposed for improving the data placement, task scheduling and replication mechanism is summarised as follows:

- An adaptive scheduling algorithm named ELRAS that identifies the capacity of node in a heterogeneous cluster environment is proposed. A three layer mapping relationship is used to acquire cluster configuration and node statistics information dynamically. Further, a method for tracking the data objects is integrated with the scheduler.
- An ARS is proposed to determine the data objects to be replicated and the location for placing the replica.
- The environment and task are heterogeneous because hardware parameters and workloads always vary with respect to time. Hence, the cluster setup for evaluation is configured with heterogeneous nodes and the workload characteristics.

Table 1 Comparison among the existing scheduling policies in MapReduce Framework

Method/references	Performance metrics evaluated	Test bed and environment	Performance objective and limitation of the approach	Limitations addressed by proposed method
Delay scheduling [15]	Data locality and fairness	Private cluster and Amazon EC2 Facebook workload Heterogeneous	Scheduling is performed in a data local manner by slightly relaxing the fairness Maximum data locality achieved is only about 94% for 100 Map tasks	In the proposed ELRAS, data locality is achieved up to 99% for 100 Map tasks
Resource-aware adaptive scheduling (RAAS) [29]	Resource utilization and completion time	Private cluster Gridmix benchmark Homogeneous	Task scheduling is done based on the availability of resources and focused for improving the resource utilization In RAAS, maximum resource utilization is attained about 45% for mixed workload case	ELRAS offers maximum resource utilization (up to 92%) for mixed workload case
COSHH [17]	Data locality and mean completion time	Private cluster Gridmix benchmark Heterogeneous	Task is classified and scheduled in the cluster as data local manner In COSHH, data locality is achieved around 98% for 100 Map tasks and time consumed for execution is high because of the classification process	Data locality is achieved about 99% for 100 Map tasks. Further, total job completion time of ELRAS is less as compared to COSHH
Locality-aware reduce task scheduling (LARTS) [30]	Data locality	Private cluster Word count and sort benchmark Homogeneous	Scheduling is done in a data local manner by avoiding scheduling delay, network traffic and Completion time In LARTS, data locality is observed only about 60% for 11 Map tasks	In ELRAS, maximum data locality is attained (up to 99%) for 11 Map tasks
History-based auto-tuning (HAT) [31]	Completion time and scalability	Private cluster Sort and word count application Homogeneous	The objective of HAT is to reduce the execution time For 10 jobs, HAT consumes 8000 s as the execution time	In ELRAS, total job execution time is comparatively less than the HAT (i.e.), for 10 jobs, it requires only 3500 s

Table 1 continued

Method/references	Performance metrics evaluated	Test bed and environment	Performance objective and limitation of the approach	Limitations addressed by proposed method
Self-adaptive MapReduce scheduling (SAMR) [32]	Reduce runtime and scalability	Private cluster Sort and Word Count application Heterogeneous	The aim of SAMR is to minimize the execution time by focusing on scalability and without data locality constraints The execution time is decreased merely about 17% as compared to FIFO	ELRAS considers data locality and it is highly scalable in a dynamic cluster environment. Here, execution time is decreased up to 76% while compared with FIFO
Maestro [33]	Network traffic, data locality and run time	Private cluster and Grid5000 Gridmix benchmark Heterogeneous (or) homogeneous	It maximizes the computations in a data local manner by reducing the network traffic and execution time In Maestro, 95% data locality only achieved for the given 100 Map tasks	Because of the higher data locality achieved in ELRAS, cross-rack communication and execution time are kept in minimum
Context aware scheduler for Hadoop (CASH) [34]	Execution time and throughput	Private cluster Terasort benchmark Heterogeneous	The intention of CASH is to reduce the execution time and to improve the throughput The execution time is reduced only about 20% - 36% as comparing with FIFO. Moreover, data is simply balanced across the nodes in the cluster	In ELRAS, an efficient data placement strategy is presented and execution time is reduced up to 76% while comparing with FIFO
Quincy [19]	Data locality, throughput and fairness	Private cluster DatabaseJoin, Sort, PageRank, PrimeSmall and PrimeLarge Heterogeneous	Scheduling is done by focusing data locality and improving the throughput Throughput achieved is only about 40% while compared to native Hadoop and Data locality is found to be around 50% because of fairness constraints	In ELRAS, maximum Throughput is achieved (up to 89%) that is greater than the native Hadoop
Hybrid algorithm [18]	ACT and fairness	MRSIM Yahoo and facebook trace Heterogeneous	Hybrid scheduler focuses on the reduction of ACT and fairness improvement It considers only heterogeneous environment	The completion time of ELRAS is less than the Hybrid scheduler in a real environment. It supports both heterogeneous and homogeneous environments

Table 1 continued

Method/references	Performance metrics evaluated	Test bed and environment	Performance objective and limitation of the approach	Limitations addressed by proposed method
Cost-effective Scheduling across multiple heterogeneous MapReduce clusters (ChEsS) [35]	Execution time and data locality	Private cluster PUMA benchmark Heterogeneous	ChEsS is a Pareto based scheduling framework for assigning Jobs to the cluster Pareto frontier method used in ChEsS is an expensive computational process. The search space grows exponentially with the number of jobs and the cluster that leads to higher execution time	ELRAS is a simple and cost effective algorithm that reduces the execution time, improves throughput and resource utilization

3.1 Proposed Work

3.1.1 Overview of Proposed Work

The architecture of the ELRAS strategy is presented in Fig. 2 and sequence of operations is described in Fig. 3.

- In Fig. 2, the JobTracker holds the node statistics table (NST) that includes entries such as rack id, physical machine id, data object id, replica object id, task id of running task, CPU utilization rate, unused storage space (S_i) and status flag.
- Scheduler is located in JobTracker to make data placement decisions, task scheduling decision, replica node decision and to collect the results from the slave nodes.
- The job queue maintains the list of jobs and task that is ready for execution. TaskTracker (slave node/data node) executes the task and returns the results. Further, each slave node executes the ARS for deciding the object for replication and issue replication request to the JobTracker.

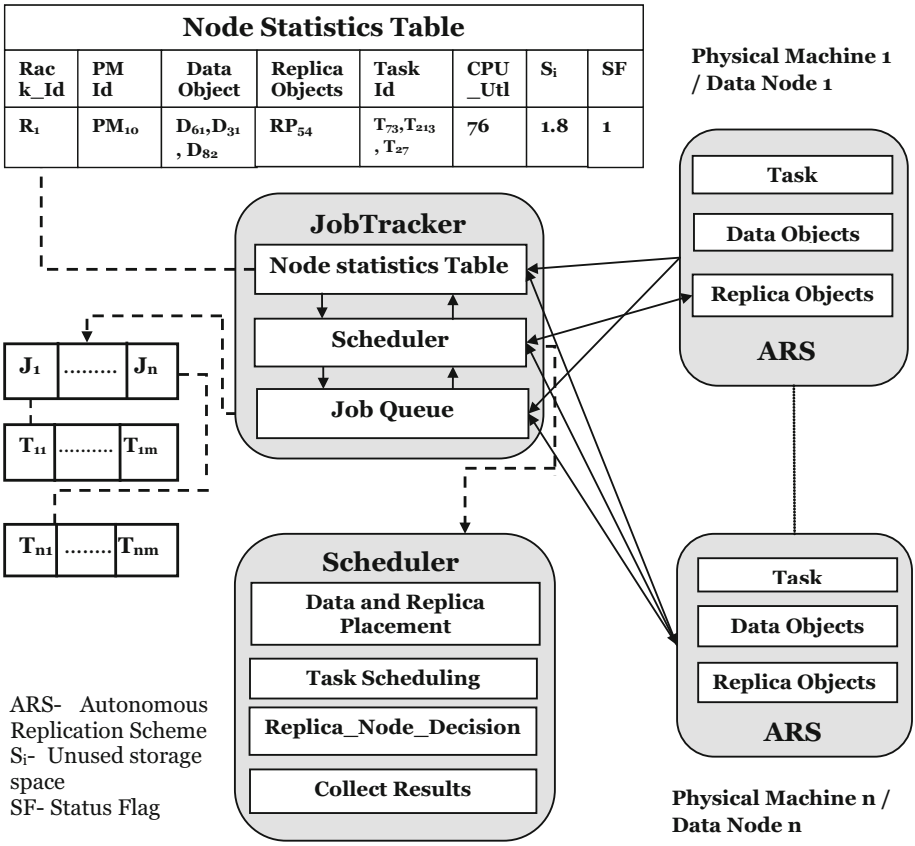


Fig. 2 Architecture of ELRAS strategy

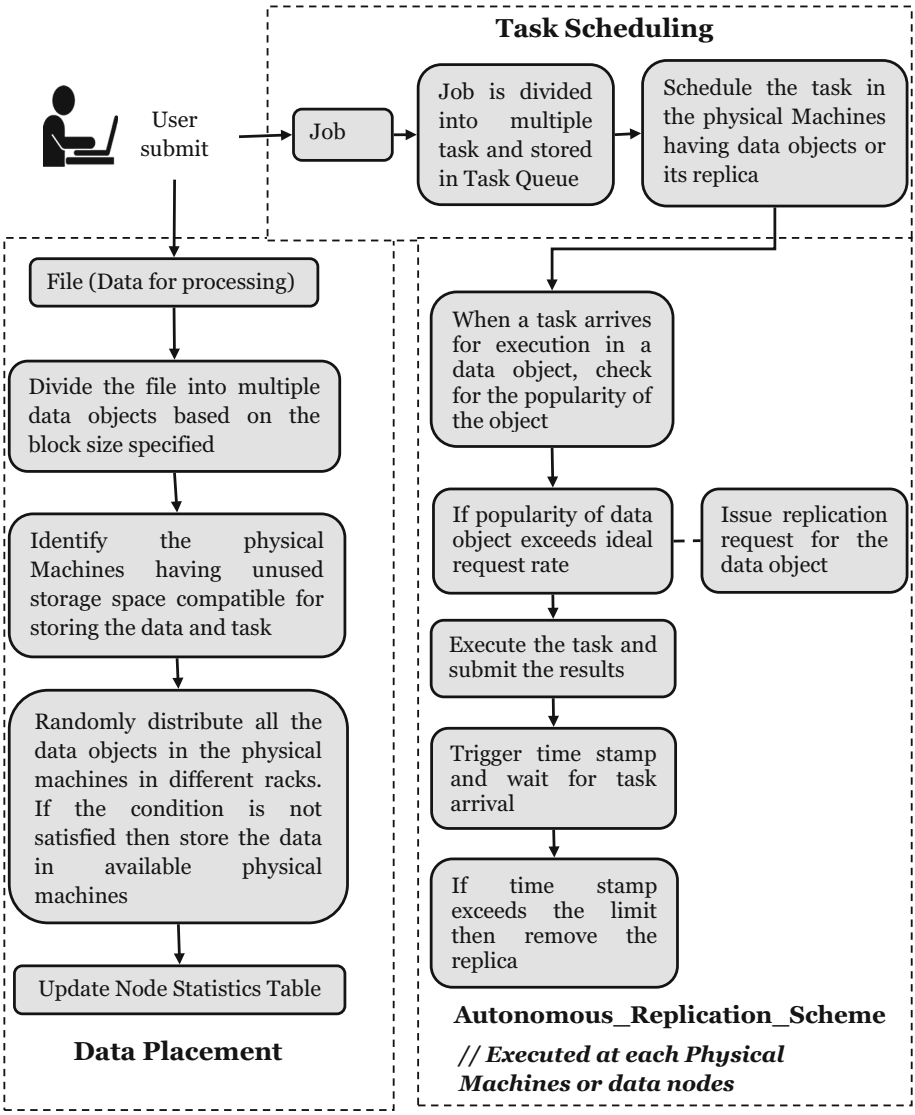


Fig. 3 Sequence of flow of operations in the ELRAS architecture

3.1.2 Efficient Locality and Replica Aware Scheduling Strategy (ELRAS)

(1) Creation and Modification of Node Statistics Table (NST)

The nodes in a cluster are organized according to a three layer mapping relationship (rack layer—physical machine layer—virtual machine layer) as illustrated in Fig. 4. When new PM are added to the network, the parameters such as Rack Id, Node_Id of the physical Machine, identity of currently running task, data objects stored, CPU utilization rate, status of the physical machine (1-for overloaded), and available free space (S_i) are added to the NST. The NST is located in the JobTracker and it frequently updates the information as a

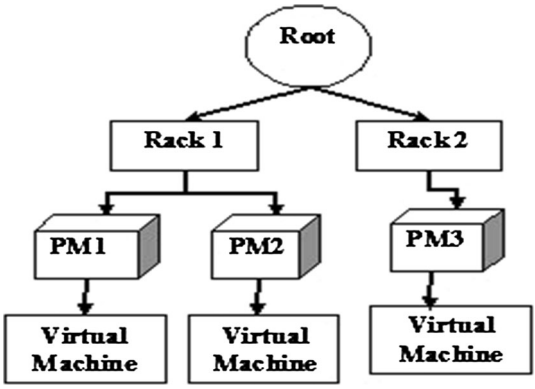


Fig. 4 Mapping relationship between different layers

Table 2 Example for node statistics table

Node statistics table							
Rack_Id	PM_Id	Data objects (D_{ij})	Replica objects (RP_{ij})	Running Task_Id	CPU_Utilization	Unused storage space (S_i)	Status flag
R_1	PM_{10}	D_{61}, D_{31}, D_{82}	RP_{54}	T_{73}, T_{213}, T_{27}	76	1.8	1
R_7	PM_7,	D_{42}, D_{71}, D_{62}		T_{15}	43	1.9	0
	PM_9	D_{93}	RP_{82}	T_{68}, T_{91}, T_{314}	58	2.5	0

new physical machine (PM_i) is added or whenever an update request (UR_i) arises from the existing PM. Algorithm 1 describes the dynamic creation and modification of NST and Table 2 is an illustrative example for NST.

Algorithm 1: Dynamic Creation and Modification of NodeStatisticsTable
Input: Incoming Physical Machines, UpdateRequest for entries
Output: NodeStatisticsTable Updated
Create entries in the Table for each Physical Machine PMi
Node_Id= PM_i //Physical Machine Id
CPU_Util= 0 // Cpu Utilization
Task_Id=0 // Running Task Id
Data Object Dij= 0 // Data Chunk j of File i
Si=0, StatusFlag=0 // Unused Storage Space S_i of PM_i and status flag}
For each new Physical Machine PM_i **do**
  Update_NodeStatisticsTable(UR_i)
End **For**
For each Update Request UR_i **do**
  Insert_ NodeStatisticsTable (Node_Id, CPU_Util, Task_id, DataObject D_{ij}, S_i, StatusFlag)
End **For**

(2) Initial Data Placement and Replica Placement

Once a file is submitted by a user for processing, it is divided into multiple blocks based on the size specified by the user or fixed default size. The number of data objects depends on the file size and the specified block size. Each data objects are placed in the available free space of the PM, starting from the first rack and the available capacity is identified through the NST. For each data placement decisions, the NST is updated and when replication request comes for a data object, the replica is placed in a node wherever the primary copy of the data object does not exist.

The proposed algorithm solves two key issues associated with data locality, namely (1) identifying the PM compatible for a data set (data placement), (2) placing the virtual machines in the identified node. The tasks are placed in the form of virtual machines (VM) in the computing nodes. For example, the data object placement strategy in a compatible free slot is displayed in Fig. 5. At this point, PM_5 consists of free space and the data objects A3, B3 and C3 are placed in PM_5. Algorithm 2 describes the process involved in the data placement. Here, γD_{ij} is the computing and storage space required for placing data and task. Where,γD_{ij} is identified from the ACT divided by the number of task slot.

Algorithm 2: Data and Replica Placement
Input: File, NodeStatisticsTable
Output: Data placement in Physical Machines { PM_1,.... PM_n}
F_i=0, D_{ij}, sf=0, bs=0, ω_i=0 // {File Id, Data Chunk Id, File Size, block size specified and number of data chunks for file Fi}//
For each File F_i **do**
  $sf \longleftarrow Sizeof(F_i)$ // Obtain from File Information
  $bs \longleftarrow Set\ by\ User$ // {Block Size Default-64MB}
  $\omega_i = sf / bs$ // Compute number of blocks or data chunks for a File Fi
  While (ω_i!=0)
    Data_Placement();
    ω_i ++;
  End **While**
End **For**
Data_Placement(D_{ij} or RP_{ij})
For each Rack Ri in NodeStatisticsTable **do**
  For each Physical Machine PM_i **do**
    Search for the nodes with unused storage space S_i
    If ($S_i > \gamma D_{ij}$)
      Then Place the data chunk D_{ij} or Replica RP_{ij} in PM_i with unused storage space S_i
      Update_NodeStatisticsTable(UR_i)
    End **If**
  End **For**
End **For**

(3) Task Scheduling in Physical Machines

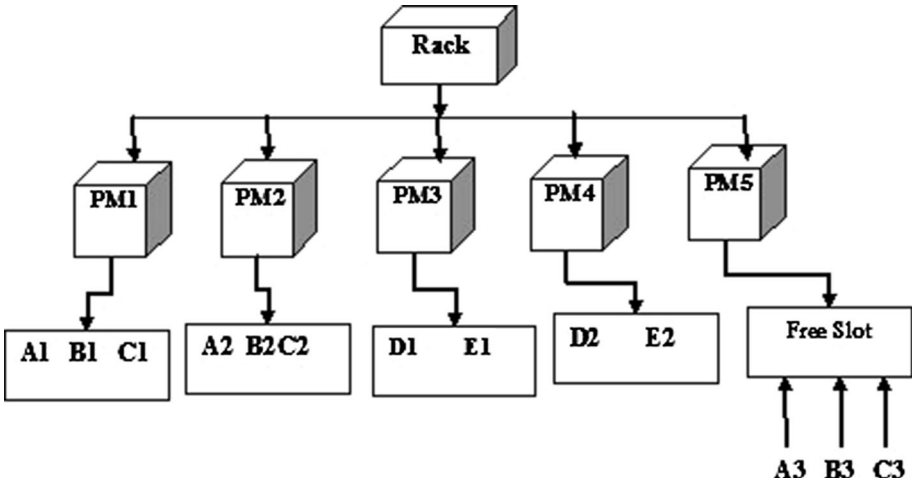


Fig. 5 Data placement strategy

In the task scheduling process, the user submits a job for execution and each job is placed in a job queue. It is further divided into sub-tasks and placed in the task queue and a Task_Id is allocated. Then, the tasks are scheduled in the PM that are not overloaded and hold the data objects or replica. Overloaded status is identified from the NST where the status flag of the corresponding overloaded node is set to one. Algorithm 3 describes the process involved in the task scheduling to the corresponding PM in clusters.

Algorithm 3: Task Scheduling in Physical Machines PM_i
Input: NodeStatisticsTable, Job $\{J_1,....J_m\}$
Output: Task Scheduled to Physical Machines PM_i
For each Job J_i in JobQueue **do**
  Split the Job J_i into multiple Task and store into TaskQueue
  Assign Task Id T_i to each Task in TaskQueue
  For each Task T_i in JobQueue **do**
    Scan NodeSatisticsTable()
    Determine the list of nodes holding D_{ij} or RP_{ij} // Data Chunk or Replica of data chunk
    Randomly Schedule the Task in the Node where D_{ij} or RP_{ij} exists
    While (PM_i flag is '1')
      Set the Task as 'Wait' in the JobQueue
      Until PM_i holding D_{ij} is set to '0'
    End **While**
  End **For**
End **For**

(4) Replica Node Decision

The algorithm 4 describes the mechanism for identifying a PM to place the replica object. If NST receives a replication request of a data object, then a node is selected based on the availability with a constraint. Initially, the replica object and data objects will be scheduled in PM at different racks, and if it is not possible then an alternate PM in the same rack is

identified. Subsequently, if a node is not available for placing the replica, then the task waiting for execution is set to wait status until the PM holding the data object is freed from overloaded status.

Algorithm 4: Replica_Node_Decision
Input: Request for Replication of D_{ij}
Ouput: D_{ij} Replicated to the physical Machines PM_i
For each Rack R_i in NodeStatistics Table **do**
  If PM_i in R_i holds the data chunk D_{ij}
    Then Proceed to next Rack R_{i+1}
    Identify the Physical Machine where $S_i > bs$
    Replicate the data chunk D_{ij}
    TaskScheduling()
  Else
    Scan R_i and identify where PM_i doesnot hold D_{ij}
    Replicate the data object D_{ij}
    TaskScheduling()
  End **If**
End **For**

(5) Autonomous Replication Scheme (ARS)

Assumption 1 Replicating the data objects only when it is necessary, consistently reduces the overhead.

Assumption 2 It is possible to replicate the data objects possess high hit rates by computing its popularity.

The algorithm 5 (ARS) is executed at each node and it decides to issue the replication request for a data object based on the load processing capacity of the PM. When the task arrival for the data object is too high, it decides further to replicate. The popularity of the data object $\lambda(D_{ij})$ is computed by adding the request arrival rate. When it exceeds the ideal request arrival rate of the physical machines ($\lambda(PM_i)$), then the replication decision is made. As the decision for replication is made, a request is issued to NST and the task in the queue with the wait status is triggered for scheduling. The ideal request rate of a PM_i is expressed as

$$\lambda(PM_i) = \frac{C_i}{\sum_{k=1}^{M} C_k}\lambda \tag{1}$$

where, C is the computing capacity of the node and λ is the average arrival rate. The popularity of the data object D_{ij} is computed as follows using Eq. (2),

$$\lambda(D_{ij}) = \sum_{j=1}^{N} \lambda_{ij} \tag{2}$$

In a PM, once a replica is placed the time stamp is triggered automatically and it checks for the arrival of task within the limit of time stamp. As soon as the time stamp exceeds the limit, the replica is removed automatically.

Algorithm 5: Autonomous_Replication_Scheme
Result: Replica Removal or Issuing Replication request
// Executed at each Physical Machine PM_i
For each Physical Machine PM_i in Rack R_i **do**
While (T_i arrives for D_{ij} or RP_{ij})
  Compute the popularity of the data chunk D_{ij}
  $\lambda(D_{ij}) = \sum_{j=1}^{N} \lambda_{ij}$ // where λ_{ij} is the request rate of the data chunk D_{ij}
  If $\lambda(D_{ij}) >= \lambda(PM_i)$ // $\gamma(PM_i)$ is the ideal request rate of the physical Machine PM_i
    Set Flag ($1 \longleftarrow PM_i$) // PM_i Overloaded
    Update NodeStatisticsTable(UR_i)
    Issue_ReplicationRequest(D_{ij})
  Else If((D_{ij} or RP_{ij})and T_i exists)
    StartExecution()
    SubmitResults()
    Remove the 'overload' flag if exists
    Poll TaskQueue
    Trigger TimeStamp TS;
    While(TS=0)
      Remove dataChunk D_{ij} or RP_{ij}
      Update_NodeStatisticsTable(UR_i)
    End While
  End **If**
End **While**
End **For**

4 Testbed and Evaluation

4.1 Experimental Setup

In the experimental work, a heterogeneous cluster is built in the Amazon EC2 environment and the corresponding node configuration is presented in Table 3. The Hadoop 1.0.0 is used for all the experiments and the block size of file system is configured as 64 MB. The ELRAS algorithm is implemented and its performance is evaluated in a heterogeneous environment with 28 nodes and different configurations. Here, the scheduler in the Hadoop package is modified with ELRAS for the evaluation of proposed approach.

4.2 Benchmark used for Analysis

HiBench [36] is a benchmarking suite and its workloads such as WordCount, TeraSort, Grep and K-Means clustering are used to test the performance of proposed framework in the Hadoop environment. The WordCount benchmark reads the input text files and computes the occurrences of each word from the input file. The grep command is employed for matching the patterns in plain-text datasets [36]. The TeraSort benchmark is used to sort the huge dataset as quick as possible and it involves the following processes: (1) Generating an input data set by TeraGen module, (2) Executing the task on the input dataset, and (3) Validating the result using TeraValidate module. The k-Means clustering is an iterative approach that is implemented as a series of MapReduce rounds. It partitions 'n'

objects into k-clusters where each object belongs to the cluster with the nearest distance. The input to the algorithm is an initial set of cluster centres and a set of objects represented as d-dimensional vectors.

4.3 Results and Discussion

4.3.1 Heterogeneous Cluster

To evaluate the performance of ELRAS approach using Hadoop, a default scheduler is modified with the proposed method in the Hadoop package. The proposed method is evaluated by using workloads presented in Table 4. Firstly, to analyse the efficiency of the schedulers in a lightly-loaded system, a sample workload with ten jobs is evaluated. Then, multiple experiments are conducted using various workloads by increasing the total number of jobs in the workload to investigate the performance variation.

The ELRAS algorithm is compared with various scheduling policies for different performance metrics. The existing competent algorithms such as Maestro, HCS, Apollo, Baseline, and delay schedulers are compared with ELRAS for validating the results and it is discussed in the subsequent sections. In addition, the existing replication schemes such as LRU, BHR, and HRS are compared with the proposed ARS for comparing the performance variations. The heterogeneity in workloads is proved by testing the algorithms for mixed workloads (i.e.), combination IO bound and CPU bound workloads. It is observed that the algorithm performs smoothly for mixed workloads and the results are presented with respect to the total job execution time, cross rack communication, computational resource usage, data locality, and throughput.

4.3.2 Total Job Execution Time

The total job execution time is the combination of queuing time, access time and execution time. It also includes the time required for moving a file to a desired location until the execution process completes. Figure 6 presents the total execution time of a job for an input data size 100 GB and 120 jobs. The replication schemes such as LRU, BHR, HRS and the proposed ARS are used in combination with ELRAS and hierarchical cluster scheduling (HCS) strategy. ELRAS scheduling strategy with the ARS provides excellent results as compared with other strategies.

In the proposed ELRAS scheduling strategy, while it is accompanied with BHR and HRS replication mechanisms a small deviation in the total execution time is observed. If the HCS + ARS combination is compared with HCS + HRS strategy, then it is noticed that no improvement in the total job execution time. Moreover, the file transmission time is the major factor to influence the total job execution time. Here, accurate scheduling and data placement approach in ELRAS effectively reduces the file transmission time. Thus, the presented results indicate the vital role of scheduling strategy and replication scheme in MapReduce model to reduce the total job execution time.

Figure 7a presents the total job execution time of the ELRAS scheduler as it is used with LRU, BHR, HRS, and ARS for multiple numbers of jobs (ranging from 10 to 120). It is observed that, ELRAS + ARS combination has less job execution time because of the replication scheme employed. The results indicate that replication is also one of the major factors affecting the scheduler in a fluctuating environment. Figure 7b presents the various existing scheduling schemes such as HRS + HCS, Apollo, Maestro, and Baseline compared with the ELRAS + ARS scheduler in terms of total job execution time. Here,

Table 3 Amazon EC2 configuration

Node type	# Nodes	CPU type	RAM (GB)	#m, r slots
Type 1	8	3. 40 GHz, 4 cores	4 GB	4, 2
Type 2	6	3. 40 GHz, 4 cores	8 GB	4, 2
Type 3	10	3.20 GHz, 4 cores	4 GB	4, 2
Type 4	4	2.00 GHz, 4 cores	4 GB	4, 2

ELRAS + ARS gives best performance as it is compared with existing scheduling policies. Figure 8 illustrates the performance of ELRAS scheduler in the heterogeneous and the homogenous environment and the observation shows that the proposed method is adaptable to fluctuating environment. For the heterogeneous environment, the configuration shown in Table 3 is used and for the homogenous environment Type 1 nodes shown in Table 3 (number of nodes—28) is selected.

4.3.3 Computing Resource Usage

The computing resource usage is the percentage usage of all computing elements exists in the cluster. Scheduling the task to a node at which no required data exists that leads to increase in the access latency because of redundant data transfer. Figure 9a, b illustrate the computing resource usage for the mixed workload with 100 and 50 GB data size. ELRAS with ARS gives the maximum computational usage about 92% and while ELRAS is combined with HRS the computational usage is only about 82%. The maximum computational usage increases about 1% than the previous case as HCS strategy is combined with the ARS scheme. Figure 10 displays the computational resource usage for (100 GB dataset and 120 jobs) various existing schedulers that are compared with the ELRAS strategy. The results indicate that for a mixed workload category, ELRAS + ARS can be exploited to achieve a better performance gain in terms of computational resource usage.

4.3.4 Cross-Rack Communication

The number of cross-rack communications for each scheduling and replication mechanism is illustrated in Fig. 11. As the ELRAS + ARS scheme is used in combination, the number of cross-rack communications is reduced because of excellent data locality. The data object is replicated as the node holding the data object is in overloaded state and there is no space for scheduling the task. ELRAS approach balances the load properly and reduces communication overhead to the maximum thereby scheduling the task as well as data effectively.

Table 4 Application characteristics of workload in terms of data size

Sl. no	Application	Data size (GB)	# Jobs
1	Word count	10, 50, 100	10–120
2	TeraSort	10, 50, 100	10–120
3	Grep	10, 50, 100	10–120
4	K-means clustering	10, 50, 100	10–120

4.3.5 Data Locality

Data locality is placing the task in the nodes where the input data resides. Locality is crucial for performance in large clusters because of network bisection bandwidth becomes a bottleneck [15]. Running the task in the same node where the data exists (data locality/node locality) is efficient and if it is not possible then running the task in the same rack (rack locality) is adopted. Figure 12 illustrates the data locality and rack locality of multiple map task as the experimentation are done with the mixed workload. Locality is achieved by replicating the hot files obviously that leads to queuing delay. Hence, for each Job separate task queue is maintained and the tasks are handled in parallel aspect.

Networks in large clusters are organized in a hierarchical fashion as the nodes are grouped under series of racks. Each rack or switch can hold a maximum of 80 nodes at the lowest level [19] and the bandwidth inside a rack is higher than bandwidth between the racks. Table 5 shows the data locality and the rack locality of various Map task while executing ELRAS compared to delay scheduling. The delay scheduling yields 99% data locality and 100% rack locality for 10 Map task. Then, as the number of Map task is increased to 100, the locality rate decreases. When ELRAS is investigated for data locality, it achieves 99% data locality and 100% rack locality for 10 to 100 Map task. As the number of Map task is increased above 100, the data locality and rack locality decreases and it is negligible. Figure 12 illustrates the node and rack locality from 3 to 1000 Map task for various scheduling schemes. Different strategies such as FIFO, delay scheduling, and ELRAS are compared and the overall performance statistics are presented in Fig. 13.

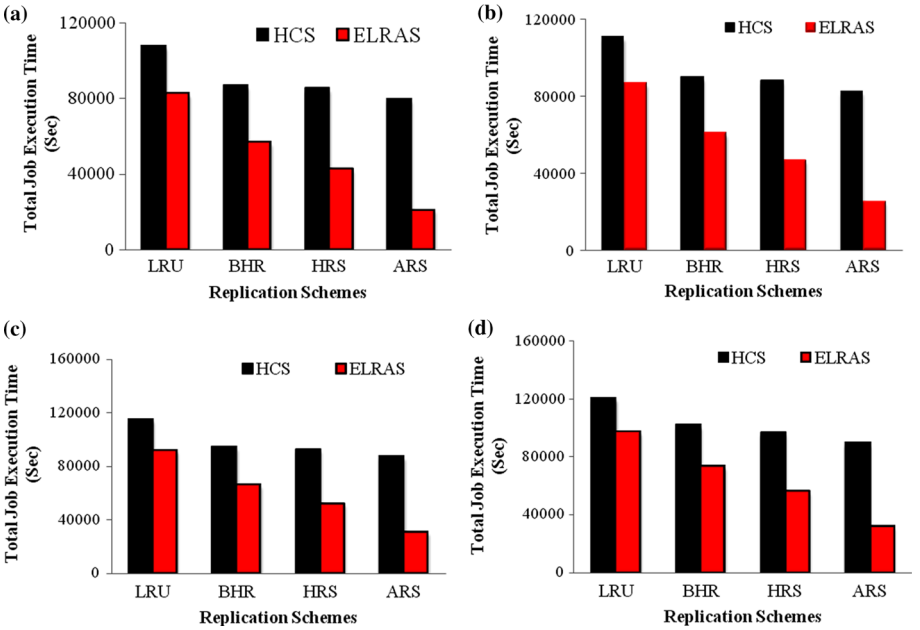


Fig. 6 Total job execution time of ELRAS and HCS with different replication schemes for workload. **a** Word count, **b** TeraSort, **c** Grep, **d** K-means

Figure 13 displays the data locality rate for a mixed workload case represented in Table 6. When the number of jobs is less, the data locality rate is below 30% for FIFO and FS schedulers. Delay scheduling performs better than FIFO and FS by achieving a locality rate of 72%. Nevertheless, the ELRAS outperforms delay scheduling by yielding an optimal locality rate about 82% for a minimum workload.

4.3.6 Throughput

The throughput of existing schedulers HCS + HRS is less because of data locality problem and interdependence between Map and reduce tasks. As the strict fairness constraint is enforced the data-locality becomes degraded [8]. In ELRAS + ARS, the load across the cluster is completely balanced that ensures high throughput since, there is no task waiting in the queue. Figure 14 shows the throughput for the time period about 10 min while running a single workload and mixed workload cases. ELRAS achieves highest throughput rate of handling 120 jobs within 10 min in a 100 GB dataset. In addition, ELRAS achieves competitive performance in throughput as compared with Delay scheduling.

From the experiments conducted and based on the observations the highlights of ELRAS are summarized as follows

- The proposed work (ELRAS) suggests that enhancing data locality and reducing the cross-rack communication significantly decrease the total job execution time of data intensive computations.
- A tracking method integrated identifies the data objects that enable efficient scheduling and reduction in cross-rack communication. Further, ARS integrated with ELRAS decides the data object for replication and adaptively removes the replica when it is idle.
- The computing resource usage rate is high for ELRAS strategy while compared with the existing scheduling and replication approaches.
- The algorithm is designed for heterogeneous cluster environment as the node characteristics are varying and dynamic in nature. The presented results show that the total job execution time for varying workload characteristics is reduced as compared to other scheduling schemes.

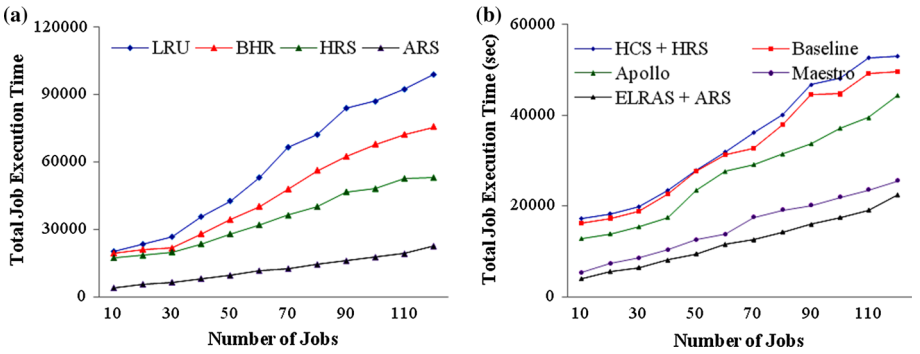


Fig. 7 Total job execution time for mixed workload and 100 GB dataset. **a** Replication Schemes + ELRAS strategy, **b** different scheduling schemes

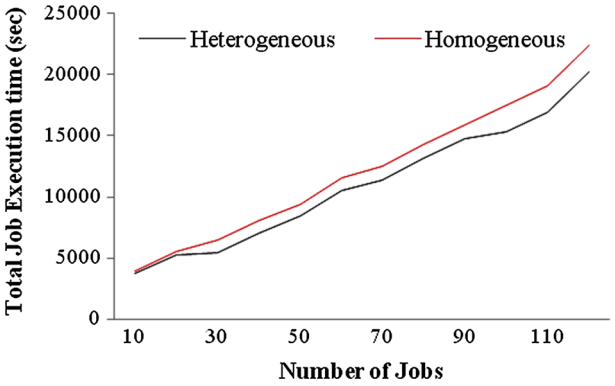


Fig. 8 Total job execution time for mixed workload and 100 GB dataset of ELRAS strategy in heterogeneous and homogenous environment

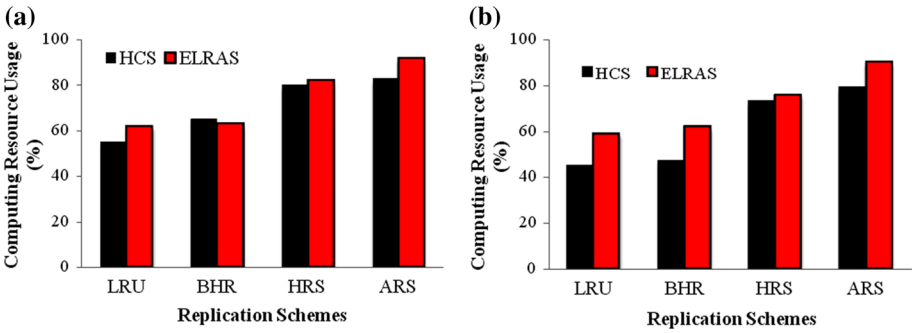
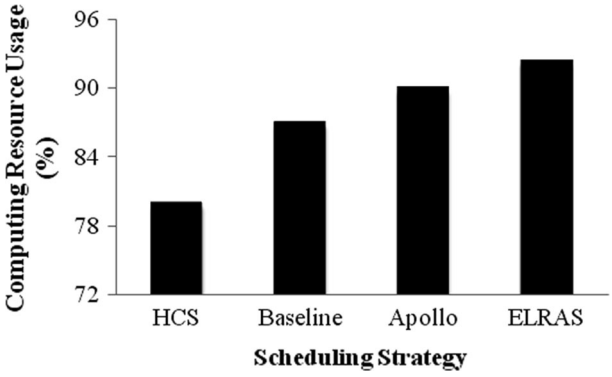


Fig. 9 Computational resource usage of ELRAS compared with HCS for different replication schemes. **a** 100 GB data and 120 jobs, **b** 50 GB data and 50 jobs

Fig. 10 Computational resource usage for different scheduling strategy for mixed workload (data size—100 GB and #jobs—120)



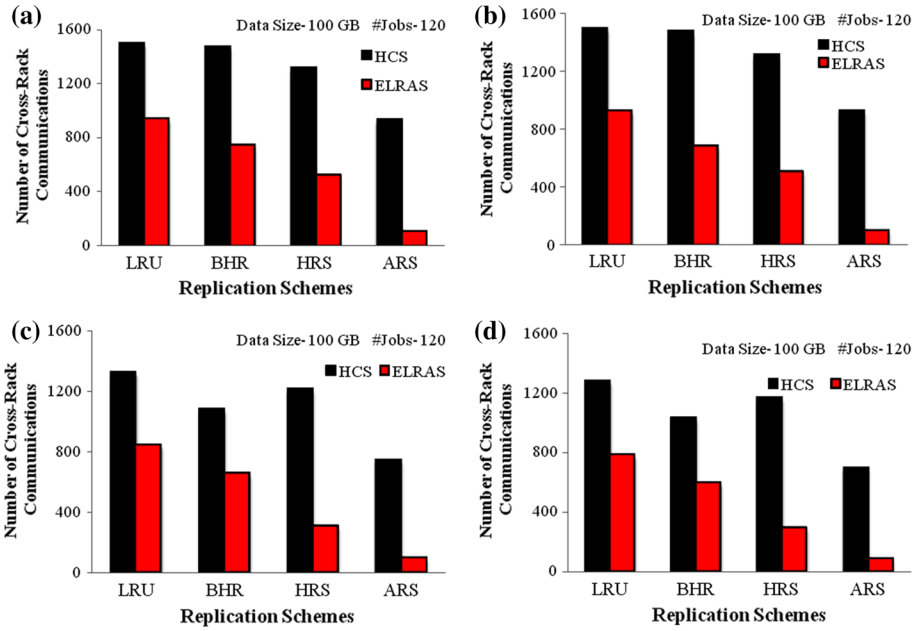


Fig. 11 Number of cross-rack communications of ELRAS compared with HCS. **a** Word count, **b** TeraSort, **c** Grep, **d** K-means

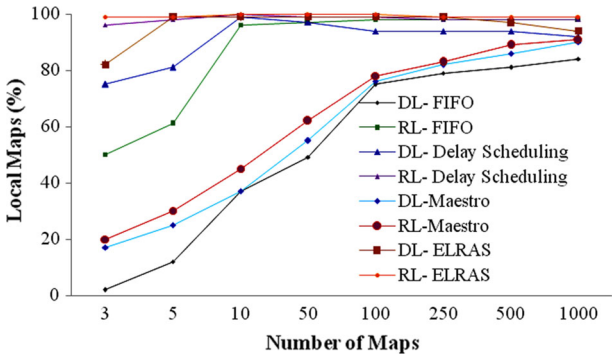


Fig. 12 Data-locality of different schedulers

Table 5 Performance comparison of delay scheduling and ELRAS for node and rack locality

Job size	Node or rack locality with delay scheduling	Node or rack locality with ELRAS
3 Maps	75%/96%	82%/99%
10 Maps	99%/100%	99%/100%
50 Maps	97%/99%	99%/100%
100 Maps	94%/99%	99%/100%
1000 Maps	92%/94%	94%/99%

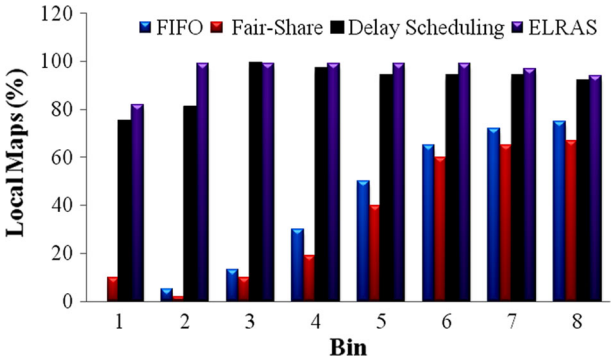


Fig. 13 Data locality for each bin (Table 5) under mixed workload

Table 6 Number of Jobs in mixed workload used for data locality measurement

Sl. no	# Maps	# Jobs in BenchMark
1	1	1
2	10	10
3	20	15
4	50–100	30
5	101–250	50
6	251–500	70
7	501–1000	100
8	1001–1500	120

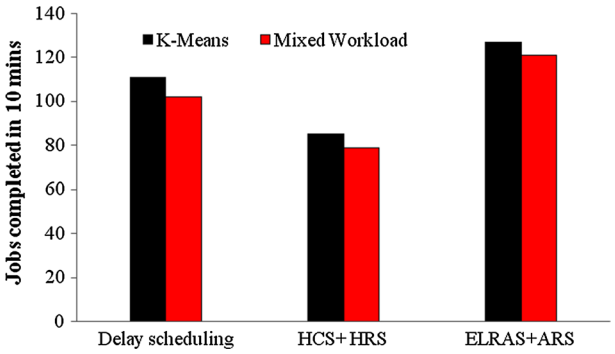


Fig. 14 Throughput comparison for K-means and mixed workload

5 Conclusions

The ELRAS algorithm presented in this article is simply adapted for MapReduce applications in Heterogeneous cluster environment. It focuses on locality and replica aware scheduling and demonstrates that the data locality improves the throughput and reduces the cross-rack communications. This algorithm is flexible to adapt for the dynamic environment while the new nodes are added or removed. An ARS is used to make decision on replication and replicas are tracked effectively using the NST. The various performance metrics such as total job execution time, computing resource usage, number of cross-rack communication and throughput are studied individually for the combination of ELRA-S + ARS in a heterogeneous cluster environment. It is also concluded that the data placement method, scheduling strategy and replication scheme play a vital role for improving the performance metrics. The results proved the efficiency of the algorithm for heterogeneous clusters and workloads. As a future work, the algorithm can be integrated with Auto-Scaling applications that are used in the commercial cloud environments.

Acknowledgements The author(s) greatly acknowledge the support of Department of Computer Science and Engineering, Anna University—Regional Campus, Tirunelveli, India for providing the computing facilities to complete this research work successfully.

References

1. Wang, W., Zhu, K., & Ying, L. (2016). MapTask scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Transactions on Networking, 24*(1), 190–203.
2. Alsmirat, M. A., Jararweh, Y., Obaidat, I., & Gupta, B. B. (2016). Internet of surveillance: A cloud supported large-scale wireless surveillance system. *Journal of Supercomputing*. doi:10.1007/s11227-016-1857-x.
3. Gou, Z., Yamaguchi, S., & Gupta, B. B. (2016). Analysis of various security issues and challenges in cloud computing environment: A survey. In *Handbook of research on modern cryptographic solutions for computer and cyber security* (pp. 393–419, Chapter 17). IGI Global. doi:10.4018/978-1-5225-0105-3.ch017.
4. Dean, J., & Ghemawat, S. (2008). MapReduce simplified data processing on large clusters. *Communications of the ACM, 51*(1), 107–113. **(50th anniversary issue)**.
5. Tripathi, S., Gupta, B. B., Almomani, A., Mishra, A., & Veluru, S. (2013). Hadoop based defense solution to handle distributed denial of service (DDoS) attacks. *Journal of Information Security, 4,* 150–164.
6. Tiwari, N., Sarkar, S., Bellur, U., & Indrawan, M. (2015). Classification framework of MapReduce scheduling algorithms. *Journal of ACM Computing Surveys*, *47*(3), 49.
7. Sun, M., Zhuang, H., Zhou, X., Lu, K., & Li, C. (2014). HPSO: Prefetching based scheduling to improve data locality for MapReduce clusters. In *Algorithms and architectures for parallel processing: 14th International conference*, China (Vol. 8631, pp. 82–95).
8. Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S., & Stoica, I. (2009). *Job scheduling for multi-user MapReduce clusters*. University of California, Berkeley, Technical Report No. UCB/EECS-2009-55.
9. Fischer, M. J., Su, X., & Yin, Y. (2010). Assigning tasks for efficiency in Hadoop: Extended abstract. In *Proceedings of the twenty-second annual ACM symposium on parallelism in algorithms and architectures*, Greece (pp. 30–39).
10. Hadoop Distributed File System. Accesed Oct 30, 2016, from https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
11. Lim, N., Majumdar, S., & Smith, P. A. (2015). A constraint programming based Hadoop scheduler for handling MapReduce jobs with deadlines on clouds. In *Proceedings of the 6th ACM/SPEC international conference on performance engineering*, Texas, USA (pp. 111–122).
12. Accesed Oct 30, 2016, from https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html.

13. Accesed Oct 30, 2016, from https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html.
14. Zhang, X., Feng, Y., Feng, S., Fan, J., & Ming, Z. (2011). An effective data locality aware task scheduling method for MapReduce framework in heterogeneous environments. In *International conference on cloud and service computing*.
15. Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S., & Stoica, I. (2010). Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *European conference on computer systems*, Paris (pp. 265–278).
16. Palanisamy, B., Singh, A., Liu, L., & Jain, B. (2011). Purlieus: Locality-aware resource allocation for MapReduce in a cloud. In *Proceedings of international conference for high performance computing, networking, storage and analysis*, New York, USA.
17. Rasooli, A., & Down, D. G. (2014). COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems. *Future Generation Computer Systems, 36,* 1–15.
18. Rasooli, A., & Down, D. G. (2012). A hybrid scheduling approach for scalable heterogeneous Hadoop systems. In *Proceedings of the 2012 SC companion: high performance computing, networking storage and analysis*, Washington DC (pp. 1284–1291).
19. Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., & Goldberg, A. (2009). Quincy: Fair scheduling for distributed computing clusters. In *Symposium on operating systems principles* (pp. 261–276).
20. Morton, K., Balazinska, M., & Grossman, D. (2010). ParaTimer: A progress indicator for MapReduce DAGs. In *Proceedings of the ACM SIGMOD international conference on management of data* (pp. 507–518). ACM.
21. Hanif, M., & Lee, C. (2016). An efficient key partitioning scheme for heterogeneous MapReduce clusters. In *18th International conference on advanced communication technology (ICACT)*, IEEE, INSPEC Accession Number: 15823957.
22. Mao, Y., Zhong, H., & Wang, L. (2015). A fine-grained and dynamic MapReduce task scheduling scheme for the heterogeneous cloud environment. In *14th International symposium on distributed computing and applications for business engineering and science*.
23. Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., & Stoica, I. (2009). Improving MapReduce performance in heterogeneous environments. In *USENIX symposium on operating systems design and implementation* (pp. 29–42).
24. Tian, C., Zhou, H., He, Y., and Zha, L. (2009). A dynamic MapReduce scheduler for heterogeneous workloads. In *Eighth international conference on grid and cooperative computing*, INSPEC Accession Number: 1090627.
25. Chang, R. S., Chang, J. S., & Lin, S. Y. (2007). Job scheduling and data replication on data grids. *Future Generation Computer Systems, 23,* 846–860.
26. Foster, I., & Ranganathan, K. (2002). Decoupling computation and data scheduling in distributed data-intensive applications. In *Proceedings of the 11th IEEE international symposium on high performance distributed computing, HPDC-11*. IEEE, CS Press, Edinburgh, UK (pp. 352–358).
27. Park, S. M., Kim, J. H., Go, Y. B., & Yoon, W. S. (2003). Dynamic grid replication strategy based on internet hierarchy. In *International workshop on grid and cooperative computing, Lecture note in computer science* (Vol. 1001, pp. 1324–1331).
28. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., & Tuecke, S. (2000). The data grid: Towards an architecture for distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications, 23,* 187–200.
29. Polo, J., Castillo, C., Carrera, D., Becerra, Y., Whalley, I., Steinder, M., Torres, J., & Ayguade, E. (2011). Resource-aware adaptive scheduling for MapReduce clusters. In *ACM/IFIP/USENIX international conference on distributed systems platforms and open distributed processing* (pp. 187–207).
30. Hammoud, M., & Sakr, M. F. (2011). Locality-aware reduce task scheduling for MapReduce. In *IEEE third international conference on cloud computing technology and science (CloudCom)* (pp. 570–576).
31. Chen, Q., Guo, M., Deng, Q., Zheng, L., Guo, S., & Shen, Y. (2011). HAT: History-based auto-tuning MapReduce in heterogeneous environments. *The Journal of Supercomputing, 64*(3), 1038–1054.
32. Chen, Q., Zhang, D., Guo, M., Deng, Q., & Guo, S. (2010). SAMR: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In *IEEE 10th international conference on computer and information technology (CIT)*, Bradford (pp. 2736–2743).
33. Ibrahim, S., Jin, H., Lu, L., He, B., Antoniu, G., & Wu, S. (2012). Maestro: Replica-aware map scheduling for MapReduce. In *12th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid)*. doi:10.1109/CCGrid.2012.122.

34. Kumar, K. A., Konishetty, V. K., Voruganti, K., & Rao, G. V. P. CASH: Context aware scheduler for Hadoop. In *Proceedings of the international conference on advances in computing, communications and informatics*, Chennai, India (pp. 52–61).
35. Zacheilas, N., & Kalogeraki, V. (2016). ChEsS: Cost-effective scheduling across multiple heterogeneous MapReduce clusters. In *IEEE international conference on autonomic computing (ICAC)* (pp. 65–74).
36. Huang, S., Huang, J., Liu, Y., Yi, L., & Dai, J. (2010). The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *IEEE 26th international conference on data engineering workshops (ICDEW)*, Long Beach, CA (pp. 41–51).

J. V. Bibal Benifa is a Research Scholar at Anna University—Regional Campus, Tirunelveli. She has received her master's in Software Engineering from Anna University in India. She obtained her bachelor's degree in Computer Science and Engineering from Anna University in India. Her research interests are virtualization and distributed data processing in cloud environment.

Dejey received her B.E. and M.E. degrees in Computer Science and Engineering from Manonmaniam Sundaranar University, Tirunelveli, India, in 2003 and 2005, respectively. Later, she was with the Department of Computer Science and Engineering, Manonmaniam Sundaranar University, Tirunelveli, India, as a Junior Research Fellow under the UGC Research Grant. She completed her Ph.D. in Computer Science and Engineering in 2011. She has been with the Department of Computer Science and Engineering, Anna University Regional Campus—Tirunelveli, as an Assistant Professor since 2010 and as the Head of the Department from 2011 to 2015. She is a member of IEEE, IE (INDIA) and ISTE. Her research interests include image and signal processing, watermarking, information hiding and multimedia security.