iaes Institute of Advanced Engineering and Science

International Journal of Cloud Computing and Services Science (IJ-CLOSER)
Vol. 3, No. 4,August 2014, pp. 233– 251
ISSN: 2089-3337

EDF Scheduling Technique for Private Cloud Environment using Hadoop MapReduce

B. Sathish Babu[1], Brinda[2], and P. Venkataram[3]
[1]Professor,Dept. of Computer Science and Engineering,Siddaganga Institute of Technology,Tumkur,Karnataka,India-572103
[2]M.Tech Scholar,Dept. of Computer Science and Engineering,Siddaganga Institute of Technology,Tumkur,Karnataka,India-572103
[3]Professor,Dept. of Electrical Communication Engineering,Indian Institute of Science, Bangalore,Karnataka,India-560012

Article Info

Article history:
Received June 26, 2014
Revised
Accepted

Keyword:
Private cloud
Deadlines
Earliest Deadline First scheduling
Hadoop MapReduce framework

Corresponding Author:
Brinda
M.Tech Scholar,Dept. of CSE, SIT, Tumkur
Karnataka,India - 572103
+91 9164775837
brinda012@gmail.com

Abstract

Job scheduling is a key issue, especially in private cloud environment where resources are limited. The importance of job scheduling increases more when an application oriented constraints such as time has to be considered, where user jobs have a deadline to meet. Two-level job scheduling scheme using EDF MapReduce framework in private cloud environment is implemented in proposed work. A first-level scheduler, the "Job Scheduler" determines the order of execution of each incoming jobs and the second-level scheduler, the "Hadoop MapReduce Server" performs the actual Map and Reduce tasks. The efficiency of proposed Earliest Deadline First is illustrated over three other scheduling techniques: FIFO scheduling, Shortest Job First scheduling and Priority Based scheduling. Experimental result shows that EDF MapReduce scheduling technique leads to the lowest executing time and lowest average waiting time for job sets as compared to other three scheduling techniques mentioned above. In EDF MapReduce scheduling, almost each job in each job sets completes their execution within their respective deadlines; hence it has almost no deadline miss or sometimes at the high load time it leads to very less deadline miss as compared to FIFO scheduling, Shortest Job First scheduling and Priority Based scheduling.

1. INTRODUCTION

Cloud computing has a variety of characteristics such as virtualization, dynamic provisioning, shared infrastructure, multitenancy, network access, etc. It has added an extra level of virtualization in whole task allocation domain which comes with the advantage of being easily scalable, but also has the downside of requiring a systematic task scheduling approach.

Scheduling an application in grid computing requires finding a subset of resources for the application. However, in cloud each scheduling request is job which is composed of set of tasks

for which resources have to be found first, then virtual machines have to be placed on them and finally job is scheduled on virtual machines. Hence to allocate resources to each job efficiently, scheduling plays important role in cloud computing [1].The importance of job scheduling increases more when an application oriented constraints such as time has to be considered, where user jobs have a deadline to meet. Priority is applied for scheduling jobs with deadlines and the scheduler assigns jobs to resources according to the priorities.

Based on the policy for assigning priority, real-time scheduling is classified into two types: fixed priority strategy and dynamic priority strategy [2]. In fixed priority scheduling all instances of one task have the same priority. The most influential algorithm under fixed priority assignment is Rate Monotonic (RM) algorithm proposed by Liu [3]. In RM algorithm, the priority of one task depends on its releasing rate. The higher the rate is, the higher the priority is. Dynamic priority assignment is more efficient than the fixed one, since it can fully utilize the processor for scheduling task. The priorities change with time, varying from one request to another or even during the same request. The most used algorithm under dynamic priority assignment is Earliest Deadline First (EDF) [4]. EDF assigns priorities to tasks inversely proportional to the deadlines of the active jobs. Whenever a scheduling event occurs (job finishes, new job released, etc.) the queue will be searched for the jobs closest to its deadline. This job is the next to be scheduled for execution [5]. In EDF it is possible to pre-emptively schedule a given collection of independent jobs such that all the jobs meet their deadlines.

Task scheduling is a key issue, especially in private cloud environment where resources are limited. Very limited research has been done so far in scheduling in private cloud, except some generic algorithm using tool OpenNebula [6]. The main motivation for using EDF task scheduling scheme in private cloud is the increasing requirements of massive data processing which in turn resulted in large tasks with deadlines. There are mainly two requirements to be considered. First requirement is that all tasks have to be guaranteed to complete their execution before their deadlines. Second is admission control, i.e., during the runtime of the system whenever new tasks with deadline arrives, the system must recalculate schedulability allowing new task to enter into system.

The rest of the paper is organized as follows: Section 2 introduces some technical background necessary to understand Eucalyptus cloud and Hadoop MapReduce framework. Section 3 presents the related work on EDF scheduling technique and MapReduce. The proposed scheme, EDF Scheduling technique for Private Cloud Environment using Hadoop MapReduce is discussed in section 4. Section 5 presents the experimental setup and section 6 presents results that support the contribution of the proposed work. Finally, Section 7 concludes the paper.

2. TECHNICAL BACKGROUND

2.1. Eucalyptus Cloud

Eucalyptus is an open source software that helps in creating and managing a private or even publicly accessible cloud. It also provides an EC2 (Elastic Cloud Computing) compatible and S3 (Simple Storage Service) compatible cloud platform. Since Eucalyptus makes use of AWS (Amazon Web Services) compatible API's, the client written for AWS can be used with Eucalyptus also. There are five high-level components each with its own web-service interface that comprise a Eucalyptus system [8]. CLC is the cloud controller which virtualizes the underlying resources (server, storage and network). The cluster controller (CC's) form the frontend for each cluster de-

fined in the cloud. NC's are the Node Controller which are the machines on which virtual machine instance executes. The Storage Controller (SC) provides block storage service (Similar to Amazon EB's) and walrus component allow users to store persistence data which is similar to the Amazon S3 in functionality.A Eucalyptus Machine Image (EMI) is a special type of pre-packaged operating system and application software that Eucalyptus uses to create a virtual machine instance.

2.2. Hadoop MapReduce Framework

Hadoop is an open source MapReduce runtime provided by the Apache Software Foundation. It uses the Hadoop Distributed File System (HDFS) as shared storage, enabling data to be shared among distributed processes using files. The Hadoop runtime consists of two types of processes: JobTracker and TaskTracker [7]. The JobTracker partitions the input data into splits using a splitting method defined by the programmer, populates a local task-queue based on the number of obtained splits, and distributes work to the TaskTrackers. Each TaskTracker controls the execution of tasks on a node. It receives a split descriptor from the JobTracker, divides the split data into records (through the 'RecordReader' component), and generates a new worker process that actually processes all the records in the split. Such worker process will run a Map task and Reduce task.

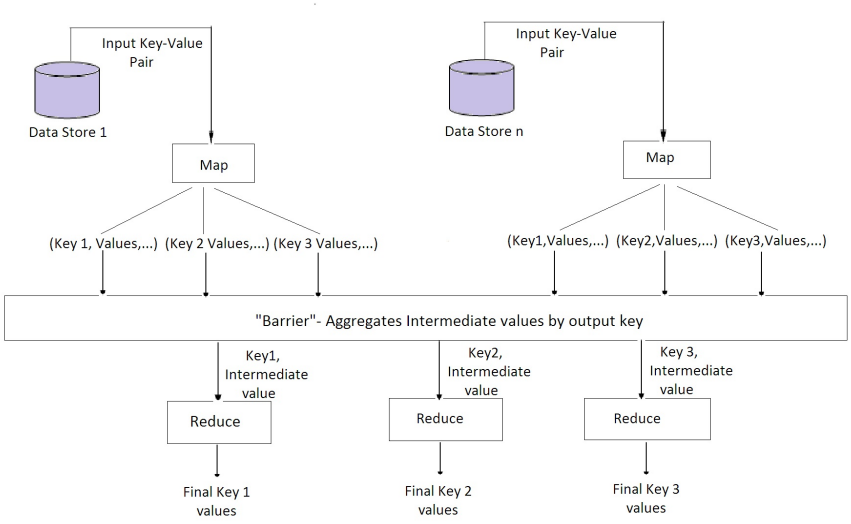

Figure 1. Data flow in MapReduce [19]

In a Map Reduce program, the Map task and Reduce task is implemented by two functions: map(), which processes fragments of input data to produce intermediate results, and reduce(), which combines the intermediate results to produce the final output. Each map input is a key-value pair (with types defined by the programmer) that identifies a piece of work. The output of each map is an intermediate result also expressed as a key-value pair (also defined by the programmer). The reduce input is composed of all the intermediate values identified by the same key; therefore, the reduce function can combine them to form the final result. Figure 1, illustrates MapReduce computation.

3. RELATED WORK

Past works have proposed heuristic driven approaches for scheduling workflow applications [9]. These heuristics cannot be applied in cloud computing environments because heuristic have bounded number of resources. Therefore, dynamic scheduling of tasks in Cloud has gained popularity in recent times. Survey on Scheduling issues in cloud computing [10] presents the comprehensive way of different type of scheduling algorithms in cloud computing environment. A partitioned earliest-deadline first symbolic schedulability analysis of dataflow graphs that minimizes the buffering requirements is discussed in [11]. The constructions of an abstract affine schedule of the graph that excludes overflow and underflow exceptions and minimizes the buffering requirements are discussed in this paper.

The problem of secondary job scheduling (jobs with low priorities) with deadlines under time-varying resource capacity is considered in [12]. For the overloaded system, an online scheduling algorithm V-Dover is proposed with asymptotically optimal competitive ratio when a certain admissibility condition holds. It is further shown that, in the absence of the admissibility condition, no online scheduling algorithm exists with a positive competitive ratio. Simulation results are presented to illustrate the performance advantage of the proposed V-Dover algorithm. [13] Proposes a new approach of semi partitioning, and design an EDF-based algorithm based on the proposed semi-partitioning technique. Tasks are never split as long as they can be partitioned. Thus, it completely succeeds the property of the traditional partitioning technique. According to the simulation results, the presented semi-partitioning approach improves schedulable multiprocessor utilization by 10% to 30% over the traditional partitioning approach.

A new MapReduce scheduling technique to enhance map task's data locality has been proposed in paper [14]. This technique is integrated into Hadoop default FIFO scheduler and Hadoop fair scheduler. To evaluate this technique, comparison is done with MapReduce scheduling algorithms with and without proposed technique. Experimental results show that this technique often leads to the highest data locality rate and the lowest response time for map tasks. Furthermore, unlike the delay algorithm, it does not require an intricate parameter tuning process. Hadoop's scheduler has a defect of data locality in resource assignment. In [15], a locality-aware scheduling algorithm (LaSA) for Hadoop-MapReduce scheduler has been implemented. Firstly, a mathematical model of weight of data interference in Hadoop scheduler is proposed. Secondly, the LaSA algorithm to use weight of data interference to provide data locality-aware resource assignment in Hadoop scheduler is introduced. Result shows reduction in network traffic and also it increases performance in data-intensive HPC systems.

In [16] the authors propose a fair scheduling implementation to manage data intensive and interactive MapReduce applications executed on very large clusters. The main concern of this scheduling policy is to give equal shares to each user. For the heterogeneous environment SAMR: a Self-Adaptive MapReduce scheduling algorithm is proposed in [17] which calculates progress of tasks dynamically and adapts to the continuously varying environment automatically. When a job is committed, SAMR splits the job into lots of ne-grained map and reduce tasks, then assigns them to a series of nodes. Meanwhile, it reads historical information which stored on every node and updated after every execution. Then, SAMR adjusts time weight of each stage of map and reduce tasks according to the historical information respectively. Experimental results show that SAMR signicantly decreases the time of execution up to 25% compared with Hadoop's scheduler and up to 14% compared with LATE scheduler.

4. PROPOSED WORK

This section focuses on Design goals, Job performance estimation, System model and Working of EDF Scheduler using Hadoop MapReduce.

4.1. Design Goals

Hadoop's default scheduler runs the jobs in FIFO order. When user jobs have deadlines to meet, most of the jobs miss their deadline which is the main disadvantage with FIFO scheduling. Shortest Job First scheduling also have a drawback of poor response time for large jobs in presence of short jobs even if large jobs have deadlines to be considered. Priority Based scheduling also have drawbacks if the lowest priority jobs have deadline requirements. Hence, the first goal of the job scheduling using EDF mechanism presented in this paper is to enable a MapReduce runtime to dynamically allocate resources in a cluster of machines based on the deadlines associated with each job. Second goal is to make sure that each job with deadline is scheduled efficiently so that they complete their execution within their respective deadlines.

The scheduler introduced in this paper uses the deadlines specified by the cloud clients. This technique targets the dynamic environment, in which new jobs can be submitted at any time in any order and in which MapReduce workload of one server share physical resources with other MapReduce servers.

4.2. Job Performance Estimation

For a given set of jobs *M* to be run on a MapReduce server, each job $m \in \mathbf{M}$ $i.e., M = \{m_1, m_2, ., m_k\}$ is composed of a set of tasks. Each task, k_i^m takes time t_i^m to complete its execution [18]. The job *m* is a set of $\{t_1, t_2, ..., t_k\}$.

The set of tasks for a given job *m* can be partitioned into tasks already completed (C_m), tasks not yet started (U_m) and tasks currently running (R_m) i.e., $(C_m) \cup (U_m) \cup (R_m) = M$. Set of tasks of job *m* already completed by TaskTracker *t* is denoted by $C_{m,t}$.

Let β_m be the mean completion task length observed for any running job *m*:

$$\beta_m = \frac{\sum_{i \subset C_m} t_i^m}{|C_m|} \tag{3.1}$$

Let β_m^t be the mean completion time for any task belonging to a job *m*:

$$\beta_m^t = \frac{\sum_{i \subset C_{m,t}} t_i^m}{|C_m,t|} \tag{3.2}$$

When implementing scheduler in Hadoop both β_m and β_m^t are to be considered. However, in the work presented in this paper, only β_m is considered and all are assumed to be equal. The main reason which motivated this decision is the design goal to make EDF task scheduler simple and

Table 1. MapReduce EDF Scheduler Definitions

Module	Functions
Job Scheduler	Responsible for assigning priorities to the tasks according to their deadline and assigning tasks to the Hadoop MapReduce Scheduler.
Hadoop MapReduce Server	Performs map and reduce operations and return the results to the job Scheduler.
Ordered Queue	Jobs are arranged on the basis of priorities assigned by job Scheduler and placed in Ordered Queue.
Map	The main map function, it takes input data from the job Scheduler and produces a list of key-value pairs.
Reduce	The main Reduce function, it is given an iterator which will iterate over all the key-value pairs generated by map tasks.

therefore TaskTracker of each Hadoop MapReduce server is considered identical. Another reason is that the task scheduling occurs in dynamic environment thus task completion time for each task changes frequently. Considering each TaskTracker's identical provides little help in scheduling jobs in dynamic changing cloud environment.

4.3. System model

Two-level job scheduling scheme using EDF MapReduce framework in private cloud environment is proposed in this paper. A first-level scheduler, the Job Scheduler determines the order of execution of applications. The second-level scheduler, the Hadoop MapReduce Server performs the actual Map and Reduce tasks. The architecture of EDF Scheduler using MapReduce framework is shown in figure 2.

A first-level scheduler, the job Scheduler works as a master node of Hadoop cluster, which distributes job scheduling work to a number of slave nodes. Hadoop MapReduce servers are the slave nodes. Jobs are assigned by Job Scheduler in response to the heartbeats (status message) received from each Hadoop MapReduce servers. Heartbeats are triggered by slaves TaskTracker and which includes number of free slots available, which gives the Job Scheduler the data necessary to schedule jobs. Table 1, provide modules and their corresponding functions, used in MapReduce EDF Scheduler framework.

4.3.1. Hadoop MapReduce Server

The main responsibility of the Hadoop MapReduce server is to process the map and reduce tasks and return the results to the Job Scheduler. It consists of Shared File System component, Task

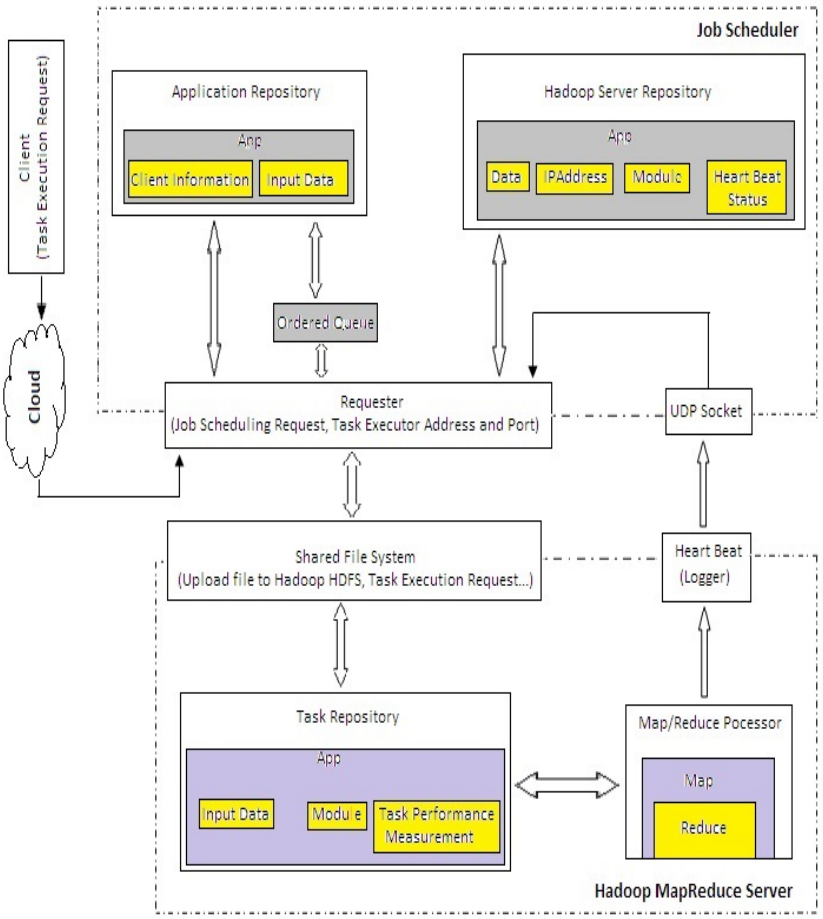

Figure 2. MapReduce EDF Scheduling Architecture.

Repository component, Heartbeat (Logger) component and Map/Reduce Processor component.

The Shared File System is used for interactions with Job Scheduler, to request job and download and upload data. Hadoop MapReduce server stores result locally in its Task Repository component includes the input data, contents of different modules and task performance measurements generated for each task. The Heartbeat component is used to maintain local statistics regarding free slots, processing times of the jobs etc. When job completes, these logger information are sent to Job Scheduler. Finally, Map/Reduce processor component implements the map and reduce tasks. Once the Hadoop MapReduce server completes the jobs, the results are uploaded to the Job Scheduler and stored in Hadoop Server Repository of Job Scheduler.

4.3.2. Job Scheduler

The main function of Job Scheduler is to assign priorities to the jobs according to their deadline and allocate them to the Hadoop MapReduce server when ever it is available to run a job. It consists of Requester component, Application Repository component, Ordered Queue component, Hadoop Server Repository component and UDP Socket.

Table 2. Different Packet Types and its functionality

TYPE	Sender	Receiver	Functionality
Job_Packet	Clients	Job Scheduler	Senders IP Addr; File name to be scheduled; Deadline.
Join_Packet	Hadoop MapReduce Server	Job Scheduler	List of Hadoop Servers IP Addr; List of files uploaded in HDFS.
Scheduling_Packet	Job Scheduler	Hadoop MapReduce Server	Hadoop Servers IP Addr; File name to be scheduled.
Heart_Beat_Packet	Hadoop MapReduce Server	Job Scheduler	Status either *"Idle"* or *"Busy"*.
File_Transfer_Request	Job Scheduler	Hadoop MapReduce Server	Receive the file from the *Busy* Hadoop Server.
File_Transfer_Start	Job Scheduler	Hadoop MapReduce Server	Send the file to the *Idle* Hadoop Server.

The Requester component receives the Task Execution request from the cloud clients and stores client information locally in the Application Repository component. The Job Scheduler assign priority to each dynamically arriving job based on the deadline. Job with least deadline is given higher priority. Jobs are organized in Ordered Queue component based on their priority.UDP Socket is used to listen incoming logs from each Hadoop MapReduce server and after every few seconds log details are send to Requester component. Hadoop Server Repository stores the entire map and reduce results. When Hadoop MapReduce server is free the job with highest priority is assigned to it through Requester component.

4.4. Working

The proposed work focuses on aperiodically arriving jobs in dynamic private cloud environment, where the main objective is the completion of jobs within their respective deadlines. For creating the job, the client provide the files to be scheduled along with the respective deadlines and other optional parameters which are passed in client utility function defined in the client module. There are six different packets considered in the proposed work: Job_Packet, Join_Packet, Scheduling_Packet, Heart_Beat_Packet, File_Transfer_Request and File_Transfer_Start. The functionality of each of these packets is listed in Table.2. Client sends the Job_Packet to Job Scheduler which mainly include the files to be scheduled with respective deadlines.

Once the job is submitted and the Job Scheduler receives the Job_Packet, it assign priorities to each job according to their deadlines and insert the jobs in the ordered queue according to

Table 3. Pseducode for Hadoop MapReduce Server

```
1. Input:Scheduling_Packet;File_Transfer_Start;
   File_Transfer_Request
2. Output:Join_Packet;Heart_Beat_Packet
3. begin
4.    Filesystem.get(conf) // To configure HDFS
5.    Upload Input Data files to HDFS
6.    Send Join_Packet to Job Scheduler
7.    while(true)
8.      if Scheduling_Packet
9.           Start job Execution Process
10.         Set Status_map=="Busy"
11.         Send Heart_Beat_Packet
12.         if Job Finished
13.           Set Status_map=="Idle"
14.           Send Heart_Beat_Packet
15.           Send the Job Completion Time
16.         endif
17.       elseif File_Transfer
18.           Receive the file from the Busy
              Hadoop Server
19.       else File_Transfer_Start
20.           Send the file to the Idle Hadoop
              Server
21.           endelseif
22.       else
23.         Wait for Scheduling_Packet
24.       endif
25.    endwhile
26. end
```

the priorities. The ordered queue changes dynamically on arrival of each Job_Packet. The pseducode for Job Scheduler is shown in Table 4. The main function of first-level scheduler, the Job Scheduler is to assign the jobs to different Hadoop MapReduce Server. First, the Job Scheduler read Heart_Beat_Packet from each Hadoop Server, if the status of Hadoop Server which has the requested file to be scheduled is *Idle* then Job Scheduler sends the Scheduling_Packet to the Hadoop MapReduce Server and if the status is *Busy* then Job Scheduler sends File_Transfer_Request to the *Busy* Hadoop MapReduce Server and File_Transfer_start to the *Idle* Hadoop MapReduce Server then sends the Scheduling_Packet to the *Idle* Hadoop MapReduce Server. After receiving the job completion time from Hadoop MapReduce Server, Job Scheduler sets the job status as completed.

To make the implementation simple, each Hadoop MapReduce Server node upload the Input Data file in HDFS and sends the detail to the Job Scheduler by sending the Join_Packet. The Input Data is the data file the Hadoop MapReduce server loads in HDFS and also user requests into the system. The pseducode for Hadoop MapReduce Server is shown in Table 3. Once the Scheduling_Packet is sent from Job Scheduler, the Hadoop MapReduce Server first creates a directory structure for it consisting of folder for map result, reduce result and final results. The original input data file is stored in a seperate folder so that it can be used for subsequent jobs without having to be re-uploaded. Hadoop Server waits for the Scheduling_Packet from the Job Sched-

Table 4. Pseducode for Job Scheduler

```
1. Input:Join_Packet;Heart_Beat_Packet;Job_Packet
2. Output:Scheduling_Packet;File_Transfer_Start;
        File_Transfer_Request
3. begin
4.    while(true)
5.      if Join_Packet
6.        Add Hadoop MapReduce Server IP;File
          lists in Hadoop Server Repository.
7.      elseif Job_Packet
8.        Assign Priority
9.        Add Job to Ordered Queue
10.       Read Heart_Beat_Packet from each
          Hadoop Server
11.        if status_map=="Idle"
12.          Send Scheduling_Packet
13.        else
14.          Send File_Transfer_Request to Busy
            Hadoop Server
15.          Send File_Transfer_Start to Idle Hadoop
            Server
16.          Send Scheduling_Packet
17.      if Job Completed
18.        Set Job_status=="completed"
19.        endelseif
20.      endif
21.   endwhile
22. end
```

uler. Once Scheduling_Packet is received, it sets its status as *Busy* and send the Heart_Beat_Packet to Job Scheduler. Each Hadoop MapReduce Server can receive two File_Transfer request from the Job Scheduler: File_Transfer_Request and File_Transfer_Start. If Hadoop MapReduce Server receives File_Transfer_Request it receive the file from the *Busy* Hadoop Server and if it receives File_Transfer_Start then it send the file to the *Idle* Hadoop Server. Once the file is copied from *Busy* Hadoop Server to *Idel* Hadoop Server, job execution beings. Each Hadoop MapReduce Server beings the job execution by downloading the input data from its HDFS and after this it performs the map task. Once map results are uploaded, it creates the reduce inputs by concatenating the map results. The resultant data received from the reduce operatios are stored as the final results. After the job completion, Hadoop Server set its status as *Idle* and again send the Heart_Beat_Packet to Job Scheduler along with the job completion time.

5. EXPERIMENTAL SETUP

The proposed MapReduce EDF Scheduling scheme is designed to operate in Eucalyptus private and hybrid cloud environment. Figure 3, describes the basic eucalyptus cloud setup with two servers (CC Server and NC Server) and client machine. The CC Server is the Cloud Controller in this setup, running on 64-bit CentOS machine. It also serves the functionality of Cluster Controller, Walrus and Storage Controller. NC Server is running on 64-bit CentOS machine and it

is Node Controller of Eucalyptus. Instances of the virtual machine run on the NC Server. Table 5 presents the sample configuration for CC Server and NC Server.

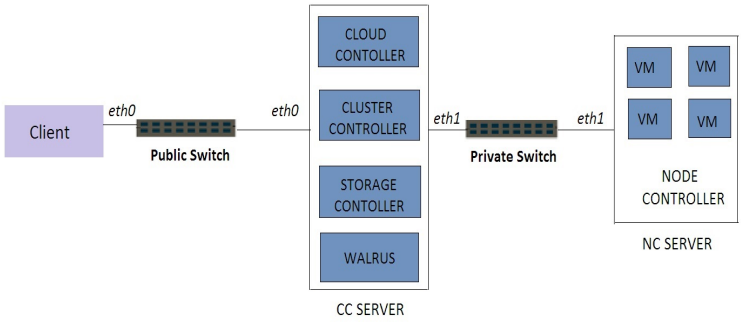

Figure 3. Basic Eucalyptus Cloud Setup.

CC Server requires two network interfaces:*eth0* and *eth1*. *eth0* is the public network interface for enterprise connections with the client and *eth1* is the private network interface for private Eucalyptus connections with Node Contollers. The client runs on Windows 64-bit version so that Firefox or other browsers can be used to access the Eucalyptus web interface.Job Scheduler module is implemented on CC Server. Hadoop MapReduce Server is implemented on other i3 Intel processor with 500GB HDD and 2GB RAM on 32-bit Windows version.

Table 5. Sample Configuration for CC Server and NC Server

	CC Server	**NC Server**
Functionality	CLC,CC,SC and Walrus	NC
No. of NIC's	*eth0*-public Network *eth1*-private Network	*eth1*-private N/W
IP Addresses	*eth0*- 192.168.196.179 *eth1*- 192.168.196.180	*eth1*:192.168.196.175
Gateway IP	192.168.196.179	192.168.196.179

5.1. Application and Scheduling Techniques

The effectiveness of Earliest Deadline First scheduling in private cloud environment can be evaluated in terms of completion time goals by all the submitted jobs. A real time application, *Gmail* application is implemented to evaluate the proposed Earliest Deadline First using Hadoop MapReduce framework. Different sets of mailing jobs along with the deadlines are submitted by cloud users to Job Scheduler. Job Scheduler running on the Cloud Controller of Eucalyptus divides the job into multiple Hadoop MapReduce Server. Each Hadoop Task Tracker is configured to run a maximum of one task in parallel (one slot for Map tasks and one for Reduce tasks). For each job the mappers emit *key,Value* where the Key is the filename and the Value is the number of different

file. The reduce tasks then separate the input records according to the value received from mapper and send emails with the attached file for which the request is sent in each job. The pseducode for Map and Reduce function is shown in Table 6 and Table 7 respectively. Each job is configured with random completion time which indicate its deadline.

Table 6. Pseducode for Map Function

```
1. Input: Scheduling_Packet; Input Data.
2. Output: key1; Value1
3. begin
4.    while(true)
5.      Read each Scheduling_Packet for unique
        file name
6.      Count unique file name
7.      Generate
8.        key1= unique file name
9.        Value1= count of unique file name
10.   endwhile
11. end
```

The efficiency of proposed Earliest Deadline First is illustrated over three other scheduling techniques: FIFO, Shortest Job First and Priority Based. In FIFO Scheduling, job leaves the queue at each Hadoop MapReduce server in the order in which they arrive: job coming in first is handled first, job coming in next waits until the first is finished. Shortest Job First Scheduler assigns the job to the Hadoop Server depending upon the length of each job. In the *Gmail* application the length of each job is calculated by the number of ToEmailAddresses. Moving short job before long job decreases the waiting time of short job but meanwhile also increases the waiting time of long job. In Priority Based Scheduling, all the jobs are mainly divided into three types of jobs: Very Strict jobs, Tight jobs and Soft jobs.Very Strict jobs have highest priority. Next priority is given to the Tight jobs then last priority is given to the Soft jobs. Jobs are scheduled on the basis of these three priorities.The following experiments are conducted which illustrates the efficiency of Earliest Deadline First in private cloud scenario when jobs deadline have to be considered.

Table 7. Pseducode for Reduce Function

```
1. Input: key1; Value1
2. Output: key2; Value2
3. begin
4.    while(true)
5.      Read key1; Value1 for each Scheduling_
         Packet
6.      email.sendMail(...)
7.      Generating
8.        key2= attached file name
9.        Value2= Either Yes or No
       (depending on deadline is missed or not)
10.   endwhile
11. end
```

Table 8. Hadoop MapReduce Server's Readings with Different Set of Jobs

MapReduce Readings	Job 10	Job 20	Job 30	Job 40	Job 50
HDFS Bytes Read	2065	4155	6218	8308	10177
HDFS Bytes Written	835	1669	3472	3337	4166
Local Bytes Read	1099	2099	3169	4239	5388
Local Bytes Written	2974	5848	8792	11736	14754
Map Output Bytes	854	1707	2561	3416	4269
Map Input Bytes	804	1607	2411	3213	4019

6. RESULT EVALUATION AND ANALYSIS

The job sets considered for the experiments is composed of five different set of jobs that share resources during their execution. Each set of jobs have different number of job scheduling requests varying from 10-50 jobs. The scenario is realistic in terms of sending the mail with attached file. Totally, four files are considered in the evaluation of proposed work. The size of each file is of 1Kb. Table 8 represents the Hadoop MapReduce Server's readings with Different Set of Jobs.

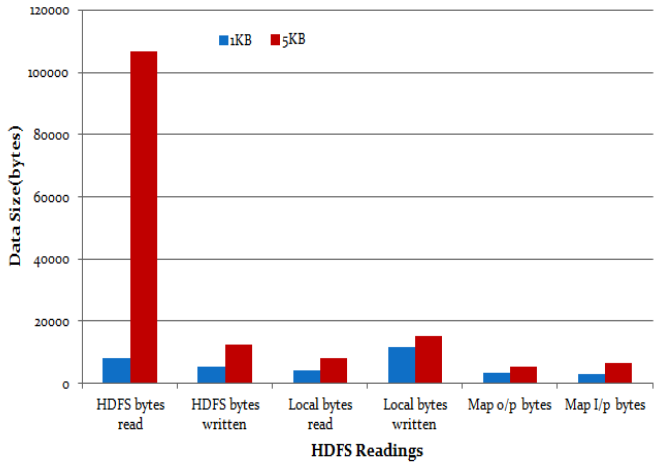

Figure 4. Comparison of Hadoop MapReduce Server's readings with 1Kb and 5Kb Files

Figure 4 represents the variations of Hadoop MapReduce Server's readings when the size of each file is increased to 5Kb. Y-axis represents the bytes reading. It can be seen from the figure that HDFS bytes read and HDFS bytes written for 5Kb file is very high as compared to 1Kb file. All the other Hadoop MapReduce Server's reading for 1Kb file is lesser as compared to 5Kb file. Following experiments are conducted with 1Kb files to evaluate the proposed Earliest Deadline First scheduling in private cloud.

6.1. Experiment One: Performance of Earliest Deadline First

To study the behaviour of Earliest Deadline First, in the first experiment different set of jobs are submitted to different number of Hadoop MapReduce Server's.

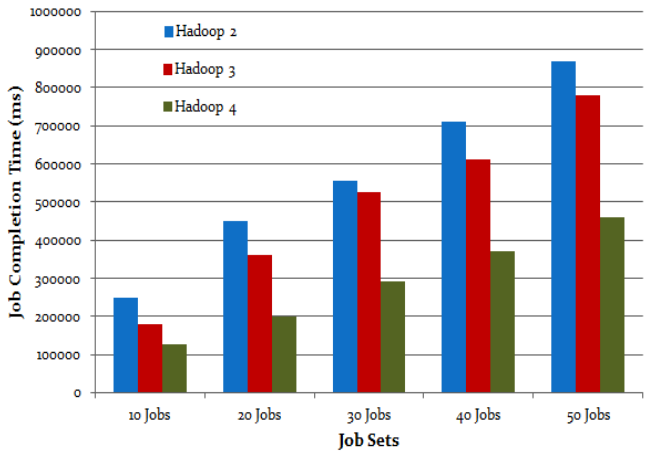

Figure 5. Job Completion Time for Different Job Sets in Different Hadoop MapReduce Server

Figure 5 shows the result of scheduling five different job sets varying from 10-50 jobs in three different Hadoop MapReduce Server's using EDF Scheduler. Y-axis represents the time in Milliseconds (ms).The job completion time for each job sets gradually decreases when the number of Hadoop Server is increased. For example, it can be see that time to schedule 50 job set in two Hadoop Server is 870000 ms whereas it decreases to 780000 ms when it is scheduled in three Hadoop Server and further it decreases to 460000ms when scheduled in four Hadoop Server.

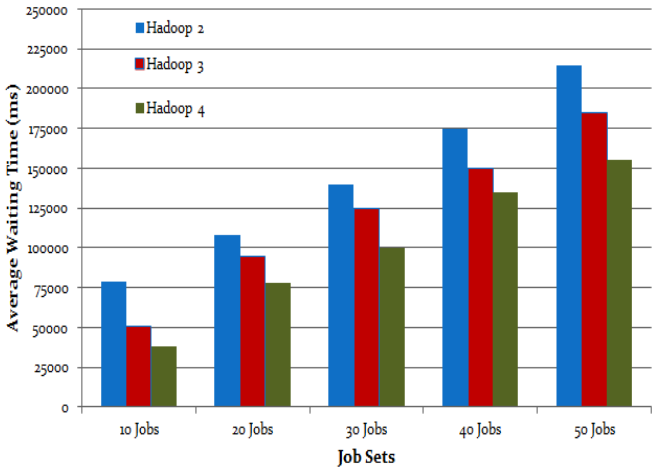

Figure 6. Average Waiting Time for Different Job Sets in Different Hadoop MapReduce Server

Figure 6 shows the average waiting time for each job sets when scheduled in three different Hadoop MapReduce Server's respectively. Here the results clearly shows that the average waiting time of each job set gradually decreases with the increase in the count of Hadoop MapReduce Server. The average waiting time for 50 job sets in two Hadoop Server is 215000 ms whereas it decreases to 185000 ms in three Hadoop Server and further it decreases to 155000 ms in four Hadoop Server.

Figure 7 shows the average Throughput of EDF scheduling approach in different Hadoop MapReduce Server's. Here the Throughput is the number of jobs scheduled per minute. With

the increase in number of Hadoop MapReduce Server's the EDF Throughput also increases. The throughput for one Hadoop Server is 2 whereas it increases to almost 4 for two Hadoop Server. For three Hadoop Server its nearer to 5.5 and further it increases to 7 for four Hadoop Server.

The first experiment results that the proposed Earliest Deadline First scheduler with Hadoop MapReduce gives better performance for the large set of jobs in private cloud environment. The performance of EDF scheduling is further compared with three other scheduling algorithms which are illustrated in the next experiments.

6.2. Experiment Two: Comparison for Deadline Miss

The aim of the second experiment is to evaluate the effectiveness of Earliest Deadline First scheduling in private cloud over FIFO, Shortest Job First and Priority Based scheduling. This experiment is conducted with two Hadoop MapReduce Servers to illustrate the number of jobs which miss its deadline when scheduled using proposed EDF scheduler and also by other three scheduling algorithm. Figure 8 illustrates the comparison of number of jobs which missed its deadline when scheduled with EDF, FIFO, Shortest Job First and Priority Based Scheduler. It can be clearly seen that for job sets 10 and job sets 20 there is no job which misses its deadline, for job sets 30 and job sets 40 there is only one job which misses its deadline and for job set 50 three job miss its deadline. The other three scheduling approach has high deadline missing rate as compared to proposed Earliest Deadline First scheduling.

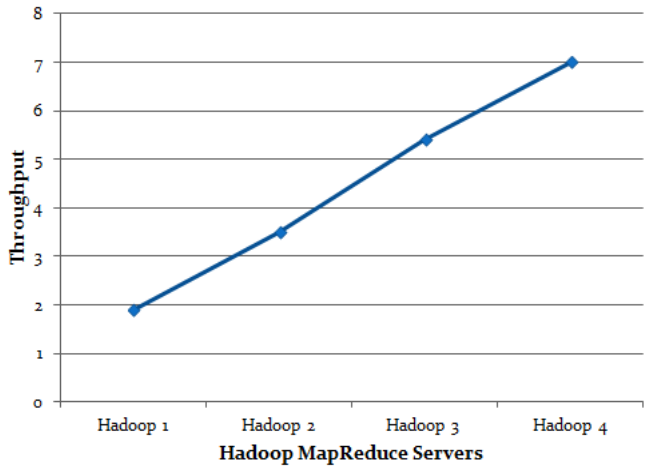

Figure 7. Average Throughput for Different Hadoop MapReduce Server

The results of second experiment shows that the proposed EDF MapReduce scheduling technique leads to almost no deadline miss or very less deadline miss during very high load on Hadoop MapReduce Server.

6.3. Experiment Three: Comparison for Job Completion Time and Average Waiting Time

The third experiment is conducted to illustrate the job completion time and average waiting time for different job sets. Different job sets are scheduled in two Hadoop MapReduce Server

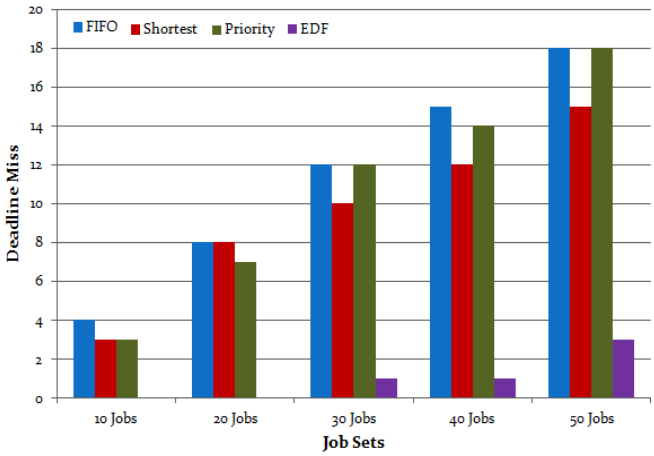

Figure 8. Comparison for Deadline Miss

using EDF and other three scheduling approach. The Figure 9 shows that the job completion time for scheduling jobs using proposed EDF is much lesser as compared to other three scheduling approaches. For scheduling 50 jobs using FIFO 850000 ms was required, when same 50 jobs are scheduled using Shortest Job First scheduling it took 890000 ms whereas for Priority Based scheduling 950000 ms was required. But when same 50 set of jobs were scheduled using proposed EDF it took 755000 ms to successfully get scheduled.

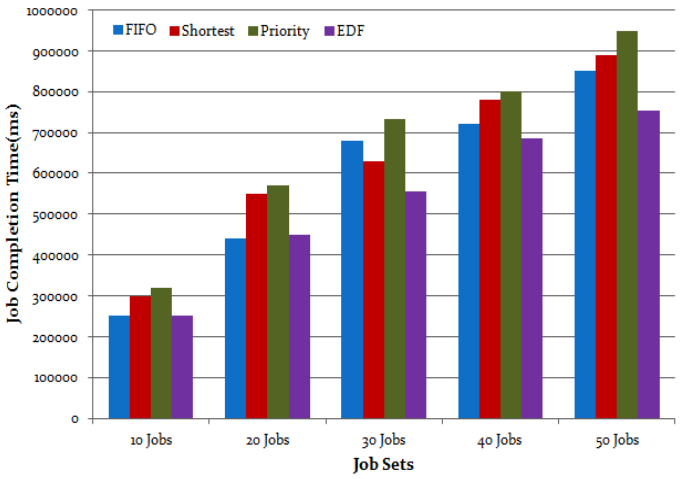

Figure 9. Comparison for Job Completion Time

Figure 10 shows the average waiting time for different job sets in two Hadoop Server. The average waiting time for different job sets also decreases same like job completion time when jobs are scheduled using proposed EDF scheduling.

The third experiment concludes that the proposed EDF scheduling in private cloud using Hadoop MapReduce Server is very efficient with respect to job completion time and average waiting time when compared with FIFO scheduler, Shortest Job First scheduler and Priority Based scheduler.

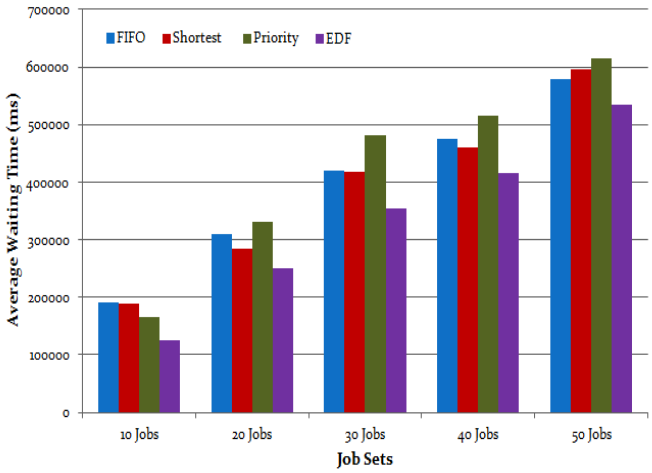

Figure 10. Comparison for Average Waiting Time

6.4. Experiment Four: Comparison for Throughputs

Fourth experiment is conduced to compare the throughput of FIFO scheduler, Shortest Job First scheduler, and Priority Based scheduler with EDF scheduler. Here the throughput is the measure of number of jobs scheduled per minute by each of the scheduler. Different job sets are scheduled in two Hadoop MapReduce Server using all four scheduling approach. Figure 11 clearly shows that throughput of EDF is higher as compared to other three scheduling approach. The throughput for scheduling 50 jobs using EDF scheduler is above 4 where as for FIFO , Shortest Job First and Priority Scheduler it is below 3.5.

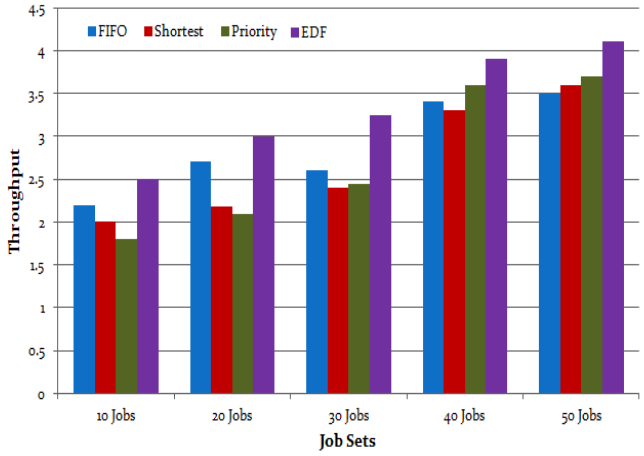

Figure 11. Comparison of Throughput

Results of experiment four indicates that the throughput of proposed EDF scheduling approach for private cloud is higher as compared with FIFO scheduler, Shortest Job First scheduler and Priority Based scheduler

7. CONCLUSION

This work presents the implementation of Earliest Deadline First Scheduling in Eucalyptus private cloud using Hadoop MapReduce framework. EDF scheduling approach has been designed to work in dynamic environment, in which new scheduling request can be submitted at any time and in which MapReduce workload of one Hadoop server share physical resources with other Hadoop servers.

The experimental result shows that, in EDF MapReduce scheduling, almost each job completes their execution within their respective deadlines; hence it has almost no deadline miss or sometimes at the high load time it leads to very less deadline miss. EDF MapReduce scheduling technique leads to the lowest executing time and lowest average waiting time for job sets as compared to the FIFO, Shortest Job First and Priority Based scheduling.

References

[1] Baoxmin Xu, Chunyan Zhao, Enzhao Hua and Bin Hu, "Job Scheduling algorithm based on Berger model in cloud environment", *Elsevier publications*, March 2011.
[2] Fei Teng , "Management des donn'ees et ordonnancement des t'aches sur architecture distributes", Version 1 - 12, E'cole central paris,Jan 2012.
[3] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-timeenvironment", *Journal of the Association for Computing Machinery*, 20(1):46–61, 1973.
[4] P. Uthaisombut. Generalization of edf and llf, "Identifying all optimal online algorithms for minimizing maximum lateness", Algorithmica, 50:312–328, 2008.
[5] Earliest deadline first scheduling. http://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling
[6] Open Nebula, http://www.opennebula.org
[7] Apache Software Foundation. Hadoop map/reduce tutorial. [Online]. Available: http://hadoop.apache.org
[8] Eucalyptus, https://www.eucalyptus.com/eucalyptus-cloud/iaas
[9] H. Topcuoglu, S. Hariri and M. Y. Wu , "Performance effective and lowcomplexity task scheduling for heterogeneous computing", *IEEE Trans. On Parallel and Distributed Systems*, Volume 13, Issue 3, pp. 260-274, 2002.
[10] Vijindra,Sudhir Shenai, "Survey on scheduling Issues in Cloud Computing" ,*International Conference On Modeling Optimization And Computing*, 2012.
[11] Adnan Bouakaz and Jean-Pierre Talpin, "Buffer Minimization in Earliest-Deadline First Scheduling of Dataflow Graphs", June 20–21,Seattle, Washington, USA,2012.
[12] Shiyao Chen,Ting He,Ho Yin Starsky Wong,Kang-Won Lee and Lang Tong, "Secondary Job Scheduling in the Cloud with Deadlines", july 2004.
[13] Shinpei Kato and Nobuyuki Yamasaki, "Semi-Partitioning Technique for Multiprocessor Real-Time Scheduling", Department of Information and Computer Science Keio University, Yokohama, Japan,2003.
[14] Chen He, Ying Lu and David Swanson, "Matchmaking: A New MapReduce Scheduling Technique", University of Nebraska-Lincoln Lincoln, U.S.
[15] Tseng-Yi Chen, Hsin-Wen Wei, Ming-Feng Wei and Ying-Jie Chen, "LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment",*International Conference on Collaboration Technologies and Systems (CTS)*, 2013 .

[16] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy,S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters", EECS Department, University of California, Berkeley Tech. Rep. UCB/EECS-2009-55, Apr 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009.

[17] Quan Chen, Daqiang Zhang and Song Guo, "SAMR: A Self-adaptive MapReduce Scheduling Algorithm In Heterogeneous Environment", *2010 10th IEEE International Conference on Computer and Information Technology*, 2010.

[18] Jorda Polo, David Carrera, Yolanda Becerra, Jordi Torres and Ian Whalley, "Performance-driven task Co-Scheduling for Mapreduce Environments", *IEEE on Network Operations and Management Symposium (NOMS)*, 2010 .

[19] Figure from slide deck from Google MapReduce course. Available under Creative Commons Attribution 2.5 License. tinyurl.com/4zl6f5.

B. Sathish Babu received his PhD in ECE from Indian Institute of Science, Bangalore, India. He is a professor at the Department of Computer Science and Engineering, Siddaganga Institute of Technology,Tumkur,Karnataka,India. His research interests include, Cloud Computing Scheduling and Security Issues, Privacy issues in WSN, Grid Computing and more. He has published his research findings in many national and international journals as well as conference proceedings. He is author of book titled "Mobile and Wireless Network Security", Tata McGrawHill published in 2010 and "Communication Protocol Engineering", PHI published in 2014. Further info on his homepage: http://pet.ece.iisc.ernet.in/sathish/