

3.4.1 Plagiarism check through software

- **Mechanism of detecting plagiarism**
- **Home page for Drill Bit Software**
- **Bills for purchasing Software**
- **Sample Plagiarism Report**

**GEETHANJALI COLLEGE OF ENGINEERING&TECHNOLOGY
(UGC AUTONOMOUS)
Cheeryal (V), Keesara (M), Medchal Dist. TS- 501 301**

Mechanism of detecting Plagiarism

Objectives

- To promote integrity in research.
- To protect ethical considerations in conducting research.
- To facilitate standardized documentation of research .
- To promote standards and practices for discouraging plagiarism.

Tool for checking plagiarism

In order to promote ethical research, the institution mandates that students to check their work plagiarism. UG/PG students must ensure that the similarity index is below 20%, as is required by most journals. They should submit evidence proving of the same.

The college uses the “Drill Bit” plagiarism detecting licensed software for checking plagiarism in the following documents:

- 1) Faculty publications
- 2) UG/PG Thesis
- 3) UG/PG paper publications and
- 4) M.Tech project reports

PRINCIPAL
Geethanjali College of Engg. and Tech.
Cheeryal (V), Keesara (M), Medchal Dist.(T.S.)-501 301.



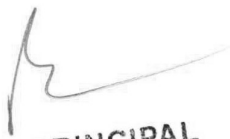
Home Page

Instructions :

1. Create a Folder
2. Select the folder to submit the document

Folders

SI No	Folder ID	Folder Name ↕	Status	Creation Date ↕	Action
1	89510	Ravi Shankar	active	2022-04-30	Edit ▾ Select
2	89465	krishnasoo943	expired	2022-04-29	Edit ▾ Select
3	83862	KrisnaPriya	expired	2022-01-30	Edit ▾ Select
4	82950	Rakesh1	expired	2022-01-19	Edit ▾ Select
5	82221	Naupal	expired	2022-01-12	Edit ▾ Select
6	81909	Rakesh	expired	2022-01-09	Edit ▾ Select
7	80928	ANIL	expired	2021-12-29	Edit ▾ Select
8	77625	Suresh	expired	2021-11-13	Edit ▾ Select
9	75269	MBA 2019	expired	2021-10-03	Edit ▾ Select


PRINCIPAL
Geethanjali College of Engg. and Tech.
Cheeryal (V), Keesara (M), Medchal Dist.(T.S.)-501 301.

TAX INVOICE

ORIGINAL FOR RECIPIENT



From

Pinnacle Nanotech India Pvt Ltd
11-8-237/4/204, C S Nilayam, Kranthi Nagar Colony,
Saroornagar,, Hyderabad, Telangana 500035

GSTIN 36AAFCP4375H1Z7
PAN AAFCP4375H

Invoice No. : INV/TS/20041
Invoice Date : 09/02/2021
Reference No : -
Place of supply : 36-Telangana
Due Date : 09/02/2021

Billing Address

Geethanjali College of Engineering and Technology
Cheeryal Village, Keesara Mandal, Medchal District., Hyderabad,
Telangana 501301, Telangana

Shipping Address

Geethanjali College of Engineering and Technology
Cheeryal Village, Keesara Mandal, Medchal District., Hyderabad,
Telangana 501301, Telangana

#	Description	HSN / SAC	Qty	Rate / Unit	Taxable Value	CGST	SGST / UTGST	Total Amount
1	DrillBit Extreme Anti Plagiarism software - 500 Document Submissions one user account 12 months Annual Subscription Cloud Based Anti-Plagiarism Software Service Customer Name: Geethanjali College of Engineering and Technology	997331	1.00 UNT	65,000.00	65,000.00	1,625.00 (2.5%)	1,625.00 (2.5%)	68,250.00
TOTAL (₹)					65,000.00	1,625.00	1,625.00	68,250.00

Bank Details:

Account Number : 31224292839 IFSC : SBIN0011666
Bank Name : STATE BANK OF INDIA Branch Name : RAMAKRISHNAPURAM, Hyderabad

Taxable Amount ₹ 65,000.00

Total Tax ₹ 3,250.00

Total amount (in words) Sixty Eight Thousand Two Hundred Fifty Rupees Only

Total Amount ₹ 68,250.00

Notes:

As per Notification: "Goods and Service Tax concession certificate in terms of Notification No. 45/2017-Central Tax (Rate) Dated: 14th Nov 2017 & 9/2018-Central Tax (Rate) Dated: 25th Jan 2018/ Notification No. 47/2017-Integrated Tax (Rate) Dated: 14th Nov 2017 Issued by the Ministry of Finance Department of Revenue of Government of India." The above exempted tax notification has been mentioned in the PO copy by Geethanjali College of Engineering and Technology. We have charged the exempted GST rate @5% on the Invoice value instead of Normal rate @18% as applicable. In future if any tax liability or a Notice raised by the Concerned GST Departments, the same GST liability will have to be borne by the Geethanjali College of Engineering and Technology Only. We (Pinnacle Nanotech India Pvt Ltd) are not the responsible for any future Tax Liability for this particular transaction.

Pinnacle Nanotech India Pvt Ltd



Authorised Signatory

Terms & Conditions:

1. We declare that this invoice shows the actual price of the goods described and that all particulars are true and correct.
2. Subject to Hyderabad Jurisdiction.
3. Interest will be Charged @18% if not paid with in due date
4. TDS Not Applicable NOTIFICATION NO. 21/2012 S.O. 1323(E), DATED 13-6-2012

PAGE - 1

PRINCIPAL
Geethanjali College of Engg. and Tech.
Cheeryal (V), Keesara (M), Medchal Dist.(T.S.)-501 301..

TAX INVOICE

ORIGINAL FOR RECIPIENT



From

Pinnacle Nanotech India Pvt Ltd
11-8-237/4/204, C S Nilayam, Kranthi Nagar Colony,
Sarooranagar,, Hyderabad, Telangana 500035

GSTIN 36AAF4375H1Z7

PAN AAF4375H

Invoice No. : INV/TS/21110
Invoice Date : 23/12/2021
Reference No : GCET/CSE/PO/21-22, Dt:
16/11/2021
Place of supply : 36-Telangana
Due Date : 27/12/2021

Billing Address

Teja Educational Society
Geethanjali College of Engineering and Technology, Cheeryal
Village, Keesara Mandal, Hyderabad, Telangana 501301,
Telangana

Shipping Address

Teja Educational Society
Geethanjali College of Engineering and Technology, Cheeryal
Village, Keesara Mandal, Hyderabad, Telangana 501301,
Telangana

#	Description	HSN / SAC	Qty	Rate / Unit	Taxable Value	CGST	SGST / UTGST	Total Amount
1	DrillBit Extreme Anti Plagiarism software - 500 Document Submissions one user account 12 months Annual Subscription Cloud Based Anti-Plagiarism Software Service	997331	1.00 UNT	65,000.00	65,000.00	1,625.00 (2.5%)	1,625.00 (2.5%)	68,250.00
TOTAL (₹)					65,000.00	1,625.00	1,625.00	68,250.00

Bank Details:

Account Number : 1443280000000819

IFSC : KVBL0001443

Bank Name : Karur Vysya Bank

Branch Name : ABIDS, Hyderabad

Taxable Amount ₹ 65,000.00

Total Tax ₹ 3,250.00

Total amount (in words) Sixty Eight Thousand Two Hundred Fifty Rupees Only

Total Amount ₹ 68,250.00

Notes:

As per Notification: "Goods and Service Tax concession certificate in terms of Notification No. 45/2017-Central Tax (Rate) Dated: 14th Nov 2017 & 9/2018-Central Tax (Rate) Dated 25th Jan 2018/ Notification No. 47/2017- Integrated Tax (Rate) Dated 14th Nov 2017 issued by the Ministry of Finance Department of Revenue of Government of India." The above-exempted tax notification has been mentioned in the PO copy by Teja Educational Society (Geethanjali College of Engineering and Technology). We have charged the exempted GST rate @5% on the Invoice value instead of the Normal rate @18% as applicable. In future, if any tax liability or a Notice is raised by the Concerned GST Department, the same GST liability will have to be borne by the Teja Educational Society (Geethanjali College of Engineering and Technology) Only. We "Pinnacle Nanotech India Pvt Ltd" are not responsible for any future Tax Liability for this particular transaction.

Pinnacle Nanotech India Pvt Ltd



Authorised Signatory

Terms & Conditions:

1. We declare that this invoice shows the actual price of the goods described and that all particulars are true and correct.
2. Subject to Hyderabad Jurisdiction.
3. Interest will be Charged @18% if not paid with in due date
4. TDS Not Applicable NOTIFICATION NO. 21/2012 S.O 1323(E), DATED 13-6-2012

PRINCIPAL
Geethanjali College of Engg. and Tech.
Cheeryal (V), Keesara (M), Medchal Dist.(T.S.)-501 301.

PAGE - 1

TAX INVOICE

ORIGINAL FOR RECIPIENT



From

Pinnacle Nanotech India Pvt Ltd
11-8-237/4/204, C S Nilayam, Kranthi Nagar Colony,
Sarooranagar,, Hyderabad, Telangana 500035

GSTIN 36AAF4375H1Z7

PAN AAFCP4375H

Invoice No. : INV/TS/19036
Invoice Date : 09/01/2020
Reference No : -
Place of supply : 36-Telangana
Due Date : 09/01/2020

Billing Address

Teja Educational Society
Geethanjali College of Engineering and Technology, Cheeryal
Village, Keesara Mandal, Hyderabad, Telangana 501301,
Telangana

Shipping Address

Teja Educational Society
Geethanjali College of Engineering and Technology, Cheeryal
Village, Keesara Mandal, Hyderabad, Telangana 501301,
Telangana

#	Description	HSN / SAC	Qty	Rate / Unit	Taxable Value	CGST	SGST / UTGST	Total Amount
1	Anti-Plagiarism Software - 500 limited uploads with 1 user Account For 1 year. Geethanjali College (03.12.2019 to 03.12.2020)	997331	1.00 UNT	62,000.00	62,000.00	1,550.00 (2.5%)	1,550.00 (2.5%)	65,100.00
TOTAL (₹)					62,000.00	1,550.00	1,550.00	65,100.00

Bank Details:

Account Number : 31224292839

IFSC : SBIN0011666

Bank Name : STATE BANK OF INDIA

Branch Name : RAMAKRISHNAPURAM, Hyderabad

Taxable Amount

₹ 62,000.00

Total Tax

₹ 3,100.00

Total amount (in words) Sixty Five Thousand One Hundred Rupees Only

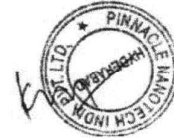
Total Amount

₹ 65,100.00

Notes:

As per Notification: "Goods and Service Tax concession certificate in terms of Notification No. 45/2017-Central Tax (Rate) Dated: 14th Nov 2017 & 9/2018-Central Tax (Rate) Dated 25th Jan 2018/ Notification No. 47/2017-Integrated Tax (Rate) Dated 14th Nov 2017 Issued by the Ministry of Finance Department of Revenue of Government of India." The above exempted tax notification has been mentioned in the PO copy by Teja Educational Society vide PO no: GCET/CSE/PO/59/19-20. Dt. 18/11/19, Anti-Plagiarism. We have charged the exempted GST rate @5% on the Invoice value instead of Normal rate @18% as applicable. In future if any tax liability or a Notice raised by the Concerned GST Departments, the same GST liability will have to be borne/cleared by the Teja Educational Society Only. We (Pinnacle Nanotech India Pvt Ltd) are not the responsible for any future Tax Liability for this particular transaction.

Pinnacle Nanotech India Pvt Ltd



Authorised Signatory

Terms & Conditions:

1. We declare that this invoice shows the actual price of the goods described and that all particulars are true and correct.
2. Subject to Hyderabad Jurisdiction.
3. Interest will be Charged @18% if not paid with in due date

PAGE - 1

PRINCIPAL
Geethanjali College of Engg. and Tech.
Cheeryal (V), Keesara (M), Medchal Dist.(T.S.)-501 301.



Pinnacle Nanotech India Pvt Ltd

PROFORMA INVOICE

Pinnacle Nanotech India Pvt. Ltd. #11-8-237/4/204, C S Nilayam, Kranthi Nagar Colony, Saroomagar, Hyderabad-500035 info@pinnacle nanotech.com		Pro Invoice No.: HYD 18013	Dt:28/11/2018		
		Buyers Order No & Date:	GCET/CSE/PO/47 22/11/2018		
Buyer: Geethanjali College of Engineering and Technology Cheeryal Village, Keesara Mandal, Medchal District., Hyderabad, Telangana 501301		Terms of Delivery:			
Sr. No.	Description of Goods	Qty	Rate	Per	Amount
1	DrillBit Extreme Anti Plagiarism Software Unlimited pages and 500 number of uploads (1 Year Lic)	1	65,000	No.	65,000.00
	CGST@9 %				5,850.00
	SGST@9 %				5,850.00
	Total				76,700.00
Amount Chargeable (In Words): Seventy Six Thousand Seven Hundred Rupees Only					
Company's CIN: U72200TG2009PTC065627 Company's GSTIN: 36AAFPC4375H1Z7 Company's PAN: AAFPC4375H Company's TAN: HYDP06686B Company's IEC: 0916504328 Declaration: We declare that this invoice shows the actual price of the goods described and that all particulars are true and correct. E & O.E					
Bank Details for RTGS/NEFT: PINNACLE NANOTECH INDIA PVT LTD. A/c No: 31224292839, IFSC Code: SBIN0011666, State Bank of India, RAMAKRISHNAPURAM Branch, Hyderabad.					

For PINNACLE NANOTECH INDIA PVT LTD.



(Authorized Signatory)

Reg Office. #11-8-237/204, Flat No.204, CS Nilayam, Kranthi Nagar Colony, Saroor Nagar, Ranga Reddy, Hyderabad, Telangana – 500 03

Corporate Office. # Plot No.: 7-145, Opp IDBI Bank, Habsiguda, Hyderabad - 500 007, Telangana

Ph: 040-4260 6941, E-Mail: info@pinnacle nanotech.com, URL: www.pinnacle nanotech.com


PRINCIPAL
Geethanjali College of Engg. and Tech.
Cheeryal (V), Keesara (M), Medchal District, Hyderabad

Your Report is Document Error

This report encountered more than 16 modified Alpha-Numeric Words.

Ex:

runl

runl

runl

runl

runl

runl

runl

runl

runl

runl

Note: Please make a proper usage of these words.



PRINCIPAL

Gesthanjali College of Engineering and Technology
(Autonomous)
Cheeryal (V), Keerasara (M), Medchal Dist. (T.S.) - 501 301

DrillBit Anti-Plagiarism Report

Dynamic load aware map reduce scheduler

By

A Sree Lakshmi



Geethanjali College of Engineering and Technology
(Autonomous)
Cheeryal (V), Kacasa (M), Maddur Dist. (T.S.) - 501 301

Document : Thesis for plagiarism_DB.pdf (2800 KB)

Submission Id : 190404012107

Submitted Date : 04-Apr-2019 01:29:24

SIMILARITY REFERENCE

13 **73** **0** **A**

SIMILARITY % MATCHED FILES ERRONEOUS WORDS GRADE

S-Good
A-Satisfactory
B-Upgrade
C-Poor

Sl.No LOCATION PRIMARY SOURCE %

1	2	pdfs.semanticscholar.org Internet Data	4
2	5	hadooptutorial.info Internet Data	1
3	4	www.ibm.com Internet Data	1
4	19	www.sciencedirect.com Internet Data	1
5	6	www.ncbi.nlm.nih.gov Internet Data	<1
6	61	docs.oracle.com Internet Data	<1
7	22	A Fast, Efficient Domain Adaptation Techniquefor Cross- Domain	<1



Geethanjali College of Engineering and Technology
(Autonomous)
Cheerayal (V), Koozuru (M), Medchal Dist. (T.S.) - 501 304.

Electroencephalography(E E G)- Based
Emotion Recog
Journal-www.mdpi.com

- | | | | |
|----|----|---|----|
| 8 | 52 | A Hybrid Cryptosystem using DNA, OTP and RSA- www.ijcaonline.org
Publication | <1 |
| 9 | 16 | Paper Published in International Journal of Science and Research (IJSR) www.ijsr.net
Publication | <1 |
| 10 | 60 | Dynamic process monitoring using multiscale PCA
IEEE Publication | <1 |
| 11 | 55 | A new MIC slot-line aerial
IEEE Publication | <1 |
| 12 | 14 | Article Published in Procedia Technology - SCIDIR
Journal | <1 |
| 13 | 49 | abiiid.files.wordpress.com
Internet Data | <1 |
| 14 | 23 | Article Published at African Journal of Advances in Electrical and Computer | <1 |


PRINCIPAL

Geethanjali College of Engineering and Technology
(Autonomous)
Cheeryal (V), Keesra (M), Medchal Dist. (T.S.) - 501 301

15	20	Markov random field modeling for three-dimensional reconstruction of the left ventricle in cardiac angiography IEEE Publication	<1
16	1	Com Prehensive Studyon Mach Inability Ofsusta Inable And Conventionalfibrere Inforcedpolymer Composites Publication	<1
17	31	Article Published by Elsevier (2012) - www.sciencedirect.com Publication	<1
18	7	Article Published in Propulsion and Power Research - SCIDIR Journal	<1
19	54	PDF File Data arizona.openrepository.com Internet Data	<1
20	45	www.dtic.mil Internet Data	<1
21	12	www.frontiersin.org	


PRINCIPAL

Internet Data

<1

22 53 yuba.stanford.edu
Internet Data

<1

23 28 2-D geometric shape recognition using canny
edge detection technique
IEEE Publication

<1

24 47 Article Published by International Research
Journal of Engineering and Technology
(IRJET) - www.irjet.net
Publication

<1

25 21 Assessment of Skills and Adaptive Learning for
Parametric Exercises C, by Muñoz-Merino,
Pedro - 2018
Publication

<1

26 72 ieeexplore.ieee.org
Internet Data

<1

27 15 A novel sampling method for multiple
multiscale targets from scatteri, by Liu,
Xiaodong - 2017
Publication

<1

28	67	Secure Approach for Data in Cloud Computing- www.ijcaonline.org Publication	<1
29	68	Student Paper Published - International Journal of Science and Research (IJSR) Publication	<1
30	46	Article Published in Sensors Journal - MDPI 2016 Student Paper.	<1
31	30	New snubbers with energy recovery into a local power supply IEEE Publication	<1
32	65	Aligarh Muslim University Paper - www.inflibnet.ac.in Student Papper	<1
33	50	Heuristic Event Filtering Methodology for Interval based Temporal- www.ijcaonline.org Publication	<1
34	11	modification of linux kernel with jumpingvirtual clock round robin scheduling Publication	<1


S. V. S. Srinivas

Geethanjali College of Engineering and Technology,
(Autonomous)
Cheerla (V), Keesara (M), Medchal Dist. (T.S.) - 501 301

35	70	laser Nonlinear resonance route and precursors, by Bonatto, Cristian - 2017 Publication	<1
36	57	jestec.taylors.edu.my Publication	<1
37	40	AN ECONOMIC ANALYSIS OF DEMAND AND SUPPLY OF HIGH VALUE AGRICULTURAL by SHIVENDRA KUMAR -2010 , Krishikosh Publication	<1
38	18	ijecs.com Publication	<1
39	32	Angular glint in polarization agility systems IEEE Publication	<1
40	17	Development of Transgenic Rice with special reference to Abiotic Str by Dhvale, Rashmi V - 2013 , krishikosh Publication	<1
41	71	www.lkr.uni-hannover.de Internet Data	<1

42 69 Paper published at BioMed Radiation Oncology
Journal - www.ro-journal.biomedcentral.com <1
Publication

43 62 www.ijeei.org <1
Internet Data

44 37 Development of an artificial neural network for
estimation of sedimen by Sahu, Sumit - 2012 , <1
krishikosh
Publication

45 8 Floyd-A Algorithm Solving the Least-Time
Itinerary Planning Problem in Urban Scheduled <1
Public Transport Network
Student Paper Published in Hindawai

46 73 ON LINE ANALYTICAL PROCESSING
SOLUTION FOR HUMAN RESOURCE <1
MANAGEMENT BY ARPAN KUMAR MAJI
2017 - KRISHIKOSH
Publication

47 51 Published in Materials Science, Faculty of
Design and Technologies - www.matsc.ktu.lt <1
Publication

48	42	Paper Published in International Journal of Materials - MDPI 2015 Student Paper.	<1
49	26	Efficient index caching schemes for data broadcasting in mobile computing environments IEEE Publication	<1
50	13	Coastal imagery from the polarimetric airborne SAR PHARUS IEEE Publication	<1
51	66	A partial variational approach for arbitrary discontinuities in planar dielectric waveguides IEEE Publication	<1
52	64	math.hws.edu Internet Data	<1
53	63	orbit.dtu.dk Internet Data	<1
54	59	An Overview of Image Steganography Techniques , ISSN 2319-7242, - www.ijecs.in Publication	<1

55	41	New Phytologist Vol-31 - 1932 , krishikosh Publication	<1
56	33	Developing a Web-Based System for Large-Scale Environmental Hydraulic, by Xie, Hao Yapa, Poo- 2006 Publication	<1
57	24	www.research.ibm.com Internet Data	<1
58	10	Cost Efficient Scheduling of MapReduce Applications on Public Clouds, by Zeng, Xuezhi Garg,- 2017 Publication	<1
59	56	Cross-domain image localization by adaptive feature fusion IEEE Publication	<1
60	39	Genetic studies for grain yield and Turcicum leaf blight resistance i by Mir, Shams-Ud-Di - 2012 , krishikosh Publication	<1
61	36	Maintainingthesearchenginefreshnessusingmobileagent	<1


PRINCIPAL

Geethanjali College of Engineering and Technology
(Autonomous)
Cheruvu (V), Keasara (M), Medchal Dist. (T.S.) - 501 301

62 48 strategic.mit.edu
Internet Data <1

63 38 A modified cryptographic approach for
securing distributed data storage in cloud
computing <1
IEEE Publication

64 25 Optimal geometric configurations for
mitigation of magnetic fields of underground
power lines <1
IEEE Publication

65 34 Some Analysis of Type 2 Diabetes Microarray
Data. by Sahu, Chinmayi K-2007 , Krishikosh
Publication <1

66 27 Beyond Batch Processing Towards Real-Time
and Streaming Big Data <1
Journal - www.mdpi.com

67 9 Aachraya Nagarjuna University Thesis
Published - www.infilibnet.ac.in <1
Student Paper

68 58 Demonstrating polynomial run-time growth for


PRINCIPAL
Geethanjali College of Engineering and Technology
(Autonomous)
Chennay (V), Keesara (M), Medchal Dist. (T.S.) - 501 301

local search matching

IEEE Publication

<1

69

44

IMMUNOMOLECULAR DETECTION AND
CHARACTERISATION OF POTYVIRUSES
INFECTI BY KRISHNAPRIYA, P 2015 -
KRISHIKOSH

Publication

<1

70

43

Paper Published in International Journal of
Science & Research (IJSR) -- www.ijsr.com

Publication

<1

71

3

Intra-bus crosstalk estimation using word-level
statistics

IEEE Publication

<1

72

29

www.dx.doi.org

Internet Data

<1

73

35

Student Article Published in MDPI Journal of
www.journals.elsevier.com

Journal

<1

Report & Checker Details

Total Number of: [Pages]

[114]

[Words], [Characters], [Image Count]

[21106], [110662], [67]


PRINCIPAL

Jeethanjali College of Engineering and Techno...
(Autonomous)
Cheeryal (V), Keerasa (M), Medchal Dist. (T.S.) - 501 301

Report: [Data Scanned]	[100 %]
[Similarity Data] , [Own Worked Data]	[13 %] , [87 %]
Checker Switches: [Erroneous Words]	[ON]
[Ignore Case Sensitive] , [RGB Selector]	[ON] , [ON]
[Include Quotes] , [Include Bibliography]	[OFF] , [OFF]
[Direct Link Reference] , [Plagiarism Rule]	[ON] , [Minimum Five Consecutive Word -> ON]



PRINCIPAL

Geethanjali College of Engineering and Technology
(Autonomous)
Cheerla (V), Keesara (M), Medchal Dist. (T.S.) - 501 301

Software Displays Similarity Data

1 ABSTRACT Map reduce framework is one of the most popularly used programming model for parallel processing of big data applications on a cluster of physical machines. Hadoop uses map reduce framework for distributed processing of big data applications. Running map reduce on cloud has many advantages like on-fly establishment of clusters, scalability, pay as you go model. But the runtime of Map reduce jobs is affected by the virtualization overhead when executed on cloud environments. An intelligent scheduling of map reduce tasks can reduce the makespan of all big data applications submitted by users to a public cloud thereby reducing the rent to be paid for the cloud infrastructure. The existing schedulers for map reduce framework are FIFO, Fair and Capacity schedulers. All these schedulers are static in nature and do not consider the run time behavior of the tasks and nodes in scheduling decisions of map and reduce tasks. Different map reduce applications have different resource usage pattern. The runtime of a task scheduled on a virtual machine is affected by the load on other virtual machines running on the same physical machine, as all the IO communication and CPU communication goes through the hypervisor, and these overheads increase as the number of VMs per bare-hardware node increases. When map/reduce tasks are scheduled on virtual machines, if resource usage of jobs and resource availability of virtual machines are considered in the decision of scheduling of multiple map and reduce tasks of different jobs, an optimized execution time can be obtained. A virtual machine executing on a node that is IO heavy is more suitable for a task which needs less amount of IO. There is a need of scheduler that suits the cloud environment, where scheduling of map and reduce tasks is done in consideration with the resource characteristics of the virtual machines and the map/reduce task. Dynamic load aware scheduler is designed where scheduling decisions are done based on map or reduce tasks resource requirements and virtual machines resource availability in terms of CPU and IO. After the completion of few map tasks, the job related statistics are used to calculate the CPU utilization and IO utilization of the map task of a particular job and categorize it. If a task is categorized as more IO intensive, then a virtual machine with less IO utilization is used for scheduling. The incomplete map tasks of that job are scheduled depending on its resource requirements and the node resource utilization patterns. The node resource utilization patterns are determined by using supervised learning technique, linear regression. The schedulers designed are observed to be more efficient in improving the runtimes of the jobs. This work can be further extended to use the determined task resource usage characteristics for proper recommendation and automation of parameter settings of map reduce environment like the split size of the map task, number of mappers that can be executed in parallel on a single virtual machine, number of virtual machines needed. As the proposed work is implemented as a contribution to Hadoop source code, it can be used for further extension by researchers. 2 2 Keywords: YARN- Yet Another Resource Negotiator HDFS- Hadoop Distributed File System MR- MapReduce CS- Capacity Scheduler CSLA- Capacity Scheduler Load Aware VM- virtual machine nVMs- Number of Virtual machines RTW- RandomTextWriter WC- WordCount MPI- Message Passing Interface FIFO- First In First Out EMR- Elastic Map Reduce LSB- Least Significant Bit MSB- Most Significant Bit AWS- Amazon Web Services DM- Difference method MAE- Mean Average Error RMSE- Root Mean Square Error LR- Linear Regression 2 CHAPTER 1: INTRODUCTION The current trend of applications are mostly internet based which generates lot of data. Social networking sites like facebook generates 4 pcta bytes of data[1], has 100million hours of video watch time, 350 million photos upload

every day. Search engines like Google processes 40,000 search queries per second on an average which is 3.5 billion searches per day[2]. Such application maintains a data warehouse in zeta bytes scale. There are plenty of real world applications in the field of business, health care, government, industry, media and entertainment industry, transportation industry, banking sector, education sector, and scientific community that work with huge volumes of data. Such applications faced many challenges with traditional data processing tools. To overcome these challenges, big data processing solutions like Hadoop were introduced. Hadoop [3] uses map reduce framework for big data processing by using the computational power of multiple nodes. Map Reduce frameworks are commonly used for parallel processing of such applications. From the parallelism technologies of MPI [4, 5] to current day map reduce programming, parallel processing of data has taken advantage of processing power of multiple nodes to have efficient execution times. Map Reduce framework breaks the computation into small tasks and runs these tasks parallel on multiple machines which can also scale easily to large cluster of machines. Map Reduce has become popular programming model for big data processing by leading companies like Google, Facebook, Yahoo etc. Hadoop is an open source big data processing engine which uses map reduce framework. Hadoop is also supported and deployed by various cloud vendors like Amazon, Cloudera, Azure etc. Amazon EMR (Elastic Map Reduce) provides managed Hadoop framework across dynamically scalable virtual machine instances of Amazon EC2. EMR cluster can be launched in minutes without any worry about cluster setup and configuration, machine provisioning with easy implementation of scalability and availability. Google cloud, Cloudera, Microsoft Azure, IBM Biginsight etc., are among leading cloud providers for big data infrastructures and services of Hadoop. The wide adoption of Hadoop makes the performance of Hadoop a crucial parameter and has become a major research topic. Executing Hadoop on cloud has become a common choice due to its easiness of creating clusters of huge capacity in very less time and the possibility of using the cluster and paying for it only during the requirement of resources. In consideration with growing importance and usage of Hadoop map reduce framework on cloud, efficient scheduling of map reduce tasks is needed for improving the performance of map reduce framework in virtualized environments. This thesis proposes an optimized load aware scheduler for map reduce frameworks which schedules the map reduce tasks based on the resource utilization of virtual machines

1.1. MOTIVATION

The invention of Hadoop and cloud computing have paved the way for multiple users/organizations to process their big data applications on huge infrastructures in an easy and flexible way. The schedulers provided in Hadoop package are designed for cluster environment and perform well in such environments, but needs some optimization, when executed in virtualized environments like cloud [14]. The existing schedulers for map reduce framework are FIFO, Fair and Capacity schedulers. These schedulers are designed in a static way [15] where the scheduling decision is either based on time of arrival or the maximum capacity guaranteed. These schedulers do not consider any dynamic behavior of machines or jobs in scheduling decisions of map and reduce tasks. The existing schedulers of map reduce framework of Hadoop can be improved further in their execution times if the runtime behavior of machines and jobs is included in the scheduling decisions. When Hadoop jobs are executed in virtualized environments like cloud, these dynamic parameters play an important role in determining the time taken by the task to finish the execution. Virtualization and multi-tenancy feature of cloud affects the performance of these jobs when executed on cloud [16, 17]. When map/reduce tasks are scheduled on virtual machines, if resource usage of jobs and resource availability of virtual machines are considered in the decision of scheduling

Geethanjali College of Engineering and Technology,
(Autonomous)
Cheerla (V), Keesara (M), Medchal Dist. (T.S.) - 501 301

of multiple map and reduce tasks of different jobs, an optimized execution time can be obtained. A virtual machine executing on a node that is IO heavy ²¹ is more suitable for a task which needs less amount of IO. There is a need of scheduler that suits the cloud environment, where scheduling ² of map and reduce tasks is done in consideration with the resource load characteristics of the virtual machines and the map/reduce task. Virtual machine load characteristics include load on virtual machine in term of its CPU usage and IO usage, number of tasks currently running on the VM. The runtime of a task scheduled on a virtual machine is affected by the load on other virtual machines running on the same physical machine, as all the IO communication and CPU communication goes through the hypervisor, and these overheads increase as the number of VMs per bare-hardware node increases. Job characteristics include number of tasks remaining to complete the job, is job a CPU intensive or IO intensive? All jobs do not use the resources in the similar way [18]. Few jobs like word count needs lot of CPU processing than the IO data transfer where as some other jobs like pi, grep needs lot of IO data transfer than CPU processing [19]. If such behavior of job can be analyzed during the execution of initial map/reduce tasks, that knowledge can be used in scheduling the remaining map/reduce tasks of the job to efficiently schedule the tasks based on the resource availability of the node. Job finishing time plays a crucial role in cloud computing environments as the users should pay per use and most of the cloud models follow hourly model for pricing. An intelligent scheduling of map reduce tasks can reduce the makespan of all big data applications submitted by users to a cloud thereby reducing the rent to be paid for the cloud infrastructure.

1.1.1. Study done on Amazon EMR This section describes about the motivational study done to determine the effect of virtualization on the performance of virtual machines while executing the map reduces tasks. Thought all map tasks execute the same map function on the data chunks of same size the performance of each virtual machine varies due to the current load on the virtual machine. Hadoop execution is tested on Amazon EMR [44] to study about the performance of execution of hadoop in cloud environments. Ganglia monitoring tool ²² [45] is used to monitor the different parameters during the execution of different jobs in Amazon EMR. Amazon EMR is a ¹⁹ cloud service where the hadoop cluster for specified number of machines is provided to user through internet on pay as you use model. Three test cases were used using TeraSort and WordCount programs provided in the examples jar file of Hadoop package. Fig.3.1 indicates sample of Hadoop Job history on Amazon EMR. Test Case 1: TeraSort Input data size: 1 GB Test Case 2 :TeraSort Input data size: 1 GB Test Case 3: WordCount Input data size: 1 GB No of maps=File Size/Split size =1024MB/64MB= 17 map tasks No of reducers=3 (programmatically). Master instance (1): ip-172-31-15-98.us-west-2.compute.internal Core instance (2): ip-172-31-15-106.us-west-2.compute.internal ip-172-31-15-107.us-west-2.compute.internal Fig.3.1. Map Reduce Job history on Amazon EMR Table 3.1 Indicates ²³ the execution times of different tasks, Taskid with letter 'm' corresponds to map task and with letter 'r' corresponds to reduce task. Table 3.1 Execution Times of Map and Reduce tasks on Amazon EMR

Test Case	Taskid	Machine instance	Time of Execution
1	m00001	107 22 107 21	106 56
1	m00002	106 50 106 42	106 55
1	m00003	107 21 107 21	107 26
1	m00004	107 21 106 40	107 27
1	m00005	106 41 106 40	106 53
1	m00006	106 40 107 20	106 53
1	m00007	107 21 107 20	107 27
1	m00008	107 21 106 41	106 53
1	m00009	106 39 106 41	106 53
1	m00010	106 39 107 20	107 27
1	m00011	107 20 107 20	107 27
1	m00012	106 35 106 34	106 42
1	m00013	107 20 107 20	107 27
1	m00014	107 21 107 24	107 28
1	m00015	106 21 106 23	106 34
1	m00016	107 17 107 17	107 23
2	r00000	106 107 106	106 106

108 106 143 r00001 106 68 106 64 106 80 r00002 107 40 107 40 107 51 From ²⁴ the execution times of the three test cases the following Table 3.2 shows comparison of performance of different virtual machines on Amazon EMR Table 3.2 Comparison of performance of virtual machines on Amazon EMR 2 Test Case Machine Instance No. of Mappers scheduled No. of Reducers scheduled Total Time spent on map tasks (sec) Total Time spent on reduce tasks(sec) Average time for map task (sec) Average time for reduce task(sec) TeraSort 106 8 2 31 5 175 39.375 87.5 107 9 1 184 40 20.44 40 TeraSort1 106 8 2 303 172 37.875 86 107 9 1 183 40 20.33 40 WordCount 106 8 2 399 223 49.875 11.5 107 9 1 238 51 26.44 51 From Table ²⁵ 3.2 it is observed that the machine instance 172-31-15-106 has an average time for map tasks as 39.375sec whereas the machine instance 172-31-15-107 has 20.44 sec. As cloud runs on virtualized environments, jobs executed on different Virtual machines affect performance as all the requests to common resources like hard disk, memory, I/O devices[46] etc., go through the same hypervisor. The ²⁶ execution time of the hadoop ²⁷ job can be improved if ²⁸ the map and reduce tasks that are scheduled to a machine considers the factor of current load and performance of the virtual machine. 1.2. PROBLEM STATEMENT Proposed a dynamic load aware scheduler for map reduce frameworks which ²⁹ schedules the map reduce tasks to virtual machines. 1.3. OBJECTIVES OF RESEARCH 2 The objectives of proposed work are: 1. To categorize the jobs into CPU intensive and IO intensive depending on the resource usage statistics of the map task execution 2. To predict and ³⁰ categorize virtual machines as CPU heavy or IO heavy virtual machines based on their current CPU and IO utilizations of the virtual node. 3. To dynamically ³¹ schedule the map reduce tasks based on task intensiveness and node heaviness in such a way that scheduler tries to schedule tasks whose resource requirements matches with the resource availability of the node to the extent possible with data locality consideration. For a particular job using the following representations Total number of map tasks for n jobs =M (Number of map tasks in a job=file size/input split size) Total number of reduce tasks for n jobs =R (Number of reduce tasks in a job = <<property as set by user>>) If N jobs are submitted to a cluster of K machines taken from the cloud, then these M map tasks are to be distributed to these K virtual machines ³² with an objective of minimum total execution time. If $t_{mi,Hi}$ denotes the time taken by the map task m_i (in the set of all map tasks of n jobs executed on machine H_i and let $n_{mi,Hi}$ represent number of map tasks scheduled on machine H_i . Then average time taken by $n_{mi,Hi}$ tasks on machine H_i is $\frac{\sum_{i=1}^M t_{mi,Hi}}{n_{mi,Hi}}$ Similarly the average time taken by $n_{ri,Hj}$ reducer tasks on machine H_j is $\frac{\sum_{i=1}^R t_{ri,Hj}}{n_{ri,Hj}}$. The objective ³³ is to minimize the overall execution time on all the three nodes. $\text{Min} [\sum_{j=1}^K \{ \sum_{i=1}^M \frac{t_{mi,Hj}}{n_{mi,Hj}} + \sum_{i=1}^R \frac{t_{ri,Hj}}{n_{ri,Hj}} \}]$ In cloud environment $t_{mi,Hj}$ or $t_{ri,Hj}$ also depends on the VM it is being deployed along with the job. 1.4. ORGANIZATION OF THESIS Further sections of this thesis give the details about the design, implementation, evaluation and conclusions of the research ³⁴ done. The theoretical analysis of the proposed method for ³⁵ determination of virtual machine load characteristics, job resource usage characteristics and dynamic load aware ³⁶ scheduling is given in section 2. Implementation and results are given in section 3 and section 4 gives the conclusion and future scope of this research. 2 CHAPTER 2: LITERATURE SURVEY 2.1. Hadoop YARN schedulers Hadoop framework is provided with three schedulers [15] 1) FIFO scheduler 2) Fair scheduler and 3) Capacity scheduler. Hadoop provides a plug and play scheduler which enables users to choose a scheduler as per their requirements. Resource Manager Module is responsible for managing and allocation of resources to application submitted to the Hadoop cluster. Resource scheduler a part of Resource Manager handles the scheduling ³⁷ aspects based on the

scheduler specified in `yarn.resourcemanager.scheduler.class` property of file `yarn-site.xml`. `<property> <name>yarn.resourcemanager.scheduler.class</name> <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value> </property>` Hadoop uses a queue data structure to place the applications submitted for execution. The hierarchy of queues and the scheduling policy adopted by the user decides about sharing of the resources in the cluster to different applications. FIFO scheduler:- This scheduler follows a strategy of First In First Out. The job submitted first in the queue is submitted first to the cluster and it uses all the resources of the cluster as required. It schedules the remaining resources to the jobs in the queue in the order of their submission time. When a heartbeat message is received from a datanode indicating a free slot to execute task: 1) It selects a job having pending task with highest priority and oldest submission time 2) The scheduler picks a task in the selected job whose data is closer to the datanode to achieve data locality. The task is selected in the order of locality where the preference is given based on the availability of input split data of the task with the order as data on the same node, data on the same rack and data on the remote rack. FIFO scheduling strategy is not advantageous in the multi tenancy environment. The other two schedulers of hadoop support multi tenancy to efficiently use the cluster resources among multiple users, thus avoiding a single user occupying the whole cluster. Fair scheduler: - Fair scheduler [17] creates pools for multiple users where each pool is guaranteed a share of resources fairly which results in the fair share of the resources in the cluster and more jobs can be executed concurrently. Fair share automatically balances the resources between all running jobs. When a first job is submitted to the cluster, all the resources of the cluster are allotted to that job as it is the only running job in the cluster. When second job is submitted, Fair scheduler tries to allocate half of the resources to each job so that both the jobs get fair share of cluster resources. Though it is not immediately possible to give the half share of resources, but as the containers are released by the first job, a fair share of resources is done among both the jobs i.e. First job submitted occupies the entire cluster initially and as other jobs are submitted every pool with jobs are provided with equal share of resources. This technique of scheduling has better cluster utilization and timely completion of smaller jobs. To set fair scheduler for hadoop framework the `yarn.resourcemanager.scheduler` class property in `yarn-site.xml` file is set to `org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler` Capacity scheduler: - Capacity Scheduler [18,19] uses multiple queues/pools in a hierarchical way where each queue is guaranteed some fraction of physical resources in the cluster which enables more jobs to be executed concurrently. FIFO scheduling is used by default for scheduling the jobs within a queue which can also be changed to other scheduler. It allows sharing cluster resources among each organization giving minimum guarantee of the capacity as specified in the configuration file `capacity-scheduler.xml`. However it allows the queue to use more than the guaranteed resources if the resources are idle. Capacity scheduler is provided as default scheduler in Hadoop 2.0. Capacity scheduler can be set for hadoop framework by specifying the `yarn.resourcemanager.scheduler` class property in `yarn-site.xml` file as `org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler`. All the map reduce schedulers provided in the hadoop package are based on some static decisions like the time of arrival, filling up to the guaranteed capacity. These schedulers do not include dynamic parameters which play a crucial role in determining the runtime of map reduce tasks especially in multitenant environments like cloud.

2.2. RELATED WORK ON MAP REDUCE SCHEDULERS

This section gives a consolidated review on the research done by many researchers for map reduce schedulers based on the resource usages. Most of the works done for Map reduce schedulers



used Hadoop map reduce framework or map reduce simulators for the evaluation of their algorithms. K.Arun Kumar et al. proposed context aware scheduler (CASH) [15] where scheduling decisions are done in consideration with job characteristics and node characteristics. CASH classifies the nodes statically instead of dynamically as computational or IO good based on the performance of CPU and IO. Moreover the work was demonstrated on a simulation environment and not on the realistic Map Reduce environment. They neglected the IO bound shuffle phase where map outputs are moved to the reducers. Their work is also based on old architecture of Hadoop. Their work is different in which nodes are classified statically as CPU good or IO good without considering the current load on the machines. Nodes are classified based on their CPU & IO performance and once classified they are considered under the same classification for the complete duration of the cluster CRESPE [20] was designed by Chen, K. et al. to optimize the resource provisioning to map reduce programs in consideration with monetary cost and time cost. It is basically a cost model for optimal setting of number of mappers and reducers that minimizes the monetary cost or the time cost. Their work is more concentrated on decision of number of nodes for map and reduces tasks in order to reduce the execution time which in turn reduces the monetary cost. 2Yi Yao et al. designed HaSTE [21] considering fitness and urgency of tasks to decide the best fit based on resource availability. All tasks of Jobs are considered as one group. Each resource has two dimensional matrix which indicates resource availability at server which gets updated at heartbeat. A job with more progress in its map phase is considered to be more urgent to schedule. The residual capacity of the resources in terms of CPU and memory is considered in scheduling decisions. IO related parameter is not used in their work which is very crucial in cloud environment than CPU and memory as all virtual machines share the same IO devices through hypervisor. Jorda Polo et al. proposed Resource aware adaptive scheduling technique [22] for map reduce multi job workloads that aims at improving utilization across machines while observing completion time goals. Authors used job profiling information to dynamically adjust the number of slots on each machine. Job completion time estimator was designed to estimate the number of map tasks that should be allocated concurrently to meet completion time goal of each time. This work is completely based on the job characteristics irrespective of the node load. Job Aware scheduling algorithm for Map Reduce Framework [23] was proposed by Radheshyam Nanduri et al. where scheduling is done with concern of the tasks that are already running on the node so that next scheduled task does not overload the system. Authors represented the task as a vector indicating CPU, memory, disk and network resource usages with maximum magnitude of 100 and used incremented Naïve Bayes classifier. They used angle between vectors to determine the dissimilarity of resource usage pattern. Operations on vectors take more time for determining the task characteristic. Moreover their work considers only the resource consumption of running 2map reduce tasks and neglects the resource consumption done by other applications running on the same node. Our proposed work is to categorize the nodes based on the resource consumption on the node by all applications. Their work was for Hadoop1 where current Hadoop2 architecture does not fix the type of slot (map slot/reduce slot). Self-adaptive map reduce scheduling (SAMR) [24] algorithm in heterogeneous environment adapts to the continuously varying environment automatically by calculating the progress of tasks dynamically. SAMR splits the given job into many fine grained tasks and assigned them to a group of nodes. While executing these fine grained map & reduce tasks, it stores the historical information on every node. Based on the historical information SAMR will adjust the time weight of each stage of the map & reduce tasks. It calculates the speculation

ve tasks ⁴² needed based on the progress of each task which is calculated accurately. It classifies the identified nodes which are slow in progress as slow nodes and accordingly launches the backup speculative tasks on them. It ensures that the backup tasks are not launched on nodes which are identified as slow nodes. A fine grained and dynamic map reduce task scheduling [25] scheme for the heterogeneous cloud environment was proposed by yingchi Mao et al. This ²¹ work is also based on collecting historical real time online info from each node in the cluster and select the appropriate parameters in order to identify slow running tasks. nodes are statically categorized based on node capability as high performance and low performance map nodes. Tasks are also classified as slow or fast map tasks based on their performance. This technique tries to launch the backup map tasks onto nodes which are categorized as high performance map nodes thereby increases the performance of speculative tasks. This technique not suitable for cloud environment as the node has other 2virtual machines and the performance of virtual machines also depends on the load of other VMs running as all the resources of the physical machine are shared virtually by all virtual machines. This work differs in the way that the nodes are classified. The nodes are statically classified as done in [CASH] by K.Arun Kumar et.al. Sewoog Kim et al. designed a Burstiness aware I/O schedule[26] for map reduce framework on virtualized environment. This work considers the I/O bottleneck of the virtualization software and designed a ² burstiness aware I/O scheduler for map reduce applications. Map and reduce tasks when executed on virtual machines, there is a possibility of I/O bottleneck due to I/O interferences caused by bursty I/Os triggered by different VM's. They proposed an algorithm which detects the I/O burstiness of a VM and schedules the bursty VM's using round robin algorithm in order to reduce the I/O interferences and frequent context switches. E-Hadoop [27], a network I/O aware scheduler for elastic map reduce cluster performs online job profiling and uses this profiling information in scheduling tasks based on available network bandwidth. It tries to scale number of containers running in parallel based on online profiling of map reduce jobs and available network bandwidth. In this method, network is used as part of allocation policy. In hadoop the scheduler checks whether the job's profile is available when client submits a job. It allocates containers based on predefined per task bandwidth value from job profiler which monitors the network traffic of every task. But it mainly worked for cross cloud platform. ShyamDeshmukh et al. proposed job classification for map reduce scheduler in heterogeneous environment [28]. It has a scheduling framework that takes into account the actual resource requirements of the job. This scheduler classifies tasks into schedulable and non-schedulable classes based on whether a job will overload a node. Job will be classified as non-schedulable if that job is unlikely to successfully run on that node. This prevents jobs from failing by executing partially which needs rescheduling. It uses overload rule that can correctly identify given state of a node as being overloaded or under loaded based on the available metrics. They used perception classifier (single layer neural network algorithm) for supervised classification of I/O job into a binary output. The main drawback of this technique is that 1) ganglia monitoring tool gives the node features of that particular VM. The load status of VM's on the same node is not considered 2) used only CPU utilization, job features need to be considered relatively. Prism [29] is a fine grained resource aware scheduling for map reduce. It considers varying resource consumption of task to task and job to job. They demonstrated that the resource demands in term of CPU, memory, disk, network usage is not same in all phases of same task. It performs resource aware scheduling ¹³ at the level of task phases. This work differentiates the resource usage at different phases and accordingly allocates the resources required at each fine grain. The resource requirements for



Gesthanjei College of Engineering and Technology
(Autonomous)
Cheerla (V), Keesara (M), Medchal Dist. (T.S.) - 501 301

each map task of the same job would be same as every map task performs the same instructions on different input splits which are also of same size expect for last split. This work also did not consider the node load levels during the resource estimation. Xiaohong Zhang et al. Locality aware dynamic VM reconfiguration on map reduce clouds [30]. It considers the fluctuating demands on computing resource of each node due to data locality and task behavior. They proposed a dynamic resource reconfiguration (DRR) for data intensive computing on clouds using dynamic VM reconfiguration. This technique maximizes the utilization of resources by adjusting the computing capability of individual virtual machines. It mainly focused on data locality. It enhances locality aware task scheduling by dynamically increasing or decreasing the computing capabilities of each node. However the currently available cloud services dynamic reconfiguration of VM is not available. This technique works with changing the configuration of virtual machine dynamically to improve the data locality of tasks i.e it increases the computing resource of a virtual machine, if the next task has its data split on the VM. Dyscale [31], a map reduce job scheduler for heterogeneous multicore processors was designed by Feng Yan et al. Dyscale tries to configure cores into categories like slow core and fast core and uses the information in scheduling. It achieves performance objectives by exploiting the capabilities of heterogeneous cores. It creates different resource pools and categorizes core types and uses a mechanism for scheduling jobs with different performance objectives for utilizing the created virtual clusters. But this work is based on the assumption that the slots are fixed as map slot and reduce slot which is part of older versions of hadoop. Dynamic load parameter is not considered in categorizing the nodes. The contributions as given in Table 1.1 are done by different researchers for optimization of map reduce tasks execution time by different scheduling techniques. Table 1.1: Limitations of related work

Author's Paper Title	Concept proposed	Limitations
2K.Arun Kumar et. al. context aware scheduler (CASH) (2012)	classifies the nodes statically as computational or IO good based on the performance of CPU and IO	1. completely a static approach of classifying the node 2. The work was demonstrated on a simulation environment and not on the realistic Map Reduce environment. 3. They neglected the IO bound shuffle phase where map outputs are moved to the reducers 4. Work on old architecture of Hadoop
Chen, K. et. al. CRESPE (2014)	optimize the resource provisioning to map reduce programs in consideration with monetary cost and time cost	1. It is basically a cost model for optimal setting of number of mappers and reducers that minimizes the monetary cost or the time cost 2. Their work is more concentrated on decision of number of nodes for map and reduces tasks in order to reduce the execution time which in turn reduces the monetary cost.
Yi Yao et. al. HaSTE (2014)	considers fitness and urgency of tasks to decide the best fit based on resource availability	1. A job with more progress in its map phase is considered to be more urgent to schedule 2. IO related parameter is not used in their work which is very crucial in cloud
Radhesyam Nanduri et. al. Job Aware scheduling algorithm for Map Reduce Framework (2011)	scheduling is done with concern of the tasks that are already running on the node so that next scheduled task does not overload the system	1. Operations on vectors take more time for determining the task characteristic. 2. Work on old architecture of Hadoop
Quan Chen et. al. Self-adaptive map reduce scheduling (SAMR) (2010)	adapts to the continuously varying environment automatically by calculating the progress of tasks dynamically and node classification as slow and fast node	3. completely a static approach of classifying the node 4. More suitable for heterogeneous environments
Shyam Deshmukh et. al. job classification for map reduce scheduler	classifies tasks into schedulable and non-schedulable classes based on whether a job will overload a node	1. The load statu

Geethanjali College of Engineering and Technology
(Autonomous)
Cheerla A.P. Konara (M), Medchal Dist. (T.S.) - 501 301

s of VM's on the same node is not considered. 2. used only CPU utilization. 2 heterogeneous environment (2013) using perception classifier Feng Yan et.al. Dyscale: A Map reduce job scheduler for heterogeneous multicore processors (2017) It creates different resource pools and categorizes core types and uses a mechanism for scheduling jobs with different performance objectives for utilizing the created virtual clusters. 1. This work is based on the assumption that the slots are fixed as map slot and reduce slot. 2. Cores are statically configured as fast or slow cores. Few other works [32, 33, 34, 35, 36, 37, 38, 39] are contributed for map reduce tasks scheduling with different goals like burstiness aware, network I/O aware, locality aware etc., to improve the performance of map reduce framework. As part of this research, the study of node load prediction [40, 41, 42, 43] works done where different machine learning algorithms are used to predict the load on node, work better for large history data samples of node resource usage and not for fewer history data. The proposed work for node load prediction can perform better for fewer history data of CPU and IO utilizations. Conclusion: 2 However, the above works done by different researchers for optimization of execution time for Map reduce frameworks, the dynamic parameters related to computing, input output processing required by the job and current load on nodes together is not considered in scheduling decisions. The execution times of map reduce jobs vary in cloud environments than in physical clusters due to its virtualization and multitenant features. Map reduce jobs when executed in cloud environments, execution time plays a very crucial role as the users need to pay for the resources used and most of the cloud providers charge the resources in an hourly model. An efficient scheduler is required which can use the resources available in the cloud in an optimal way for map reduce tasks by scheduling the tasks to the nodes based on the resource usage of the tasks and node load.

CHAPTER 3: BACKGROUND/ PRELIMINARIES

3.1. Big Data & Hadoop

Big Data: Big Data is a distributed framework for parallel processing of large scale applications. Big Data is often referred with 5 Vs which are Volume, Variety, Velocity, Value, and Veracity. Volume- can process large volumes of data Variety- can handle variety of data Velocity-increasing speed at which the data is created, processed, stored and analyzed Value- ability to transform data into information Veracity- trustworthiness of the data with data confidentiality and Availability All the 5 Vs make Big data processing very popular as most of the current day applications need analysis of variety of data with large volumes and from different sources. **Hadoop:** Hadoop [3, 7] is an open source framework for parallel processing of big data sets on a network of machines. Hadoop uses map reduce framework to process the data parallel. It uses a strategy of moving computation to data where data is distributed among multiple nodes in the cluster. Hadoop is very commonly used for data analysis and processing due to its advantages like:- 1. Automatic parallelism using map reduce technology 2. Handles structured, semi structured and unstructured data 3. Fault tolerant using the concepts like secondary name node, replication of data etc. 4. Moving computation to data 5. Easy to program 6. Uses distributed file system 7. Multi-tenancy The main components of Hadoop are: - 1) Map Reduce framework 2) YARN 3) HDFS. **MapReduce:** Map Reduce is a distributed programming model as discussed in the above section that enables parallel processing of large data sets on multiple nodes. Map Reduce is the data processing module of Hadoop. **Hadoop Distributed File System (HDFS):** HDFS [8] is the distributed file system that provides high-performance storage and access to data using multiple nodes in a cluster. HDFS is distributed storage module of Hadoop that can store big data with high availability and implicit implementation of fault tolerance. **YARN:** YARN [9] (Yet Another Resource Negotiator) is the core component of hadoop that takes care of cluster resource management, scheduling applications submitted to

the 2Map Reduce Job Map Task Input Data Map TaskMap TaskMap Task Intermediate Data Reduce Task Red
 uce Task Reduce Task Output Data cluster based on type of the scheduler and allocation of resources to the ap
 plications. It uses Hadoop Mp reduce module for big data processing and Hadoop HDFS module for distributed
 data storage. 3.2. Map Reduce Framework Map Reduce[6] is a distributed data processing framework introduced
 by Google that provides features like automatic parallelization, scalability, ease of programming, fault tolerance,
 data locality awareness. Map Reduce framework is suitable for parallel processing of large scale data processing in e
 nvironments like grid, cluster and cloud. It includes both sequential and parallel processing and is divided into two p
 hases of processing: Map phase and Reduce phase. Map and reduce tasks are executed in parallel depending on numb
 er of machines available as shown in Fig 1.1. 2Fig.1.1: Map Reduce framework Map reduce framework is more su
 itable for the problems that can be solved by using divide and conquer approach. It follows a programming model w
 here a map function is applied on set of input key value pairs to generate intermediate key value pairs. The intermedi
 ate key value pairs from the map tasks are shuffled and sorted. Reduce function is applied on the intermediate key val
 ue pairs from different map tasks to generate the final output which is also in key value pair form. Map Reduce fram
 e work has the automatic parallelization of execution by scheduling multiple map tasks onto a cluster of machines. A
 map reduce job when submitted to queue 1. The input file is split into N multiple chunks of same size (ex.64M
 B data per chunk depending on the split size user configuration parameter). These splits are distributed among
 multiple nodes of the cluster using distributed file system 2. The user program Master and Slave instances are c
 reated and executed on machines of the cluster 3. Master application is to assign the N map tasks (one Map tas
 k per split) to execute on the distributed N data chunks and R Reduce tasks for the slave applications. Numb
 er of tasks that can be executed parallel depends on the number of machines running the slave application in the
 cluster. 24. The slave application assigned with map task passes its corresponding input data split as key-value
 pairs and executes the map function as defined by the user. Map task produces output as new key value pairs (i
 ntermediate key-value pairs). If multiple pairs have same key, they are merged into a list of values to the corres
 ponding key. 5. The mappers output which are in the form of intermediate key values will be organized into R
 partitions using hash function on the intermediate keys as $\text{hash}(\text{intermediate key}) \bmod R$, all the intermediate int
 ernal key value pairs with the same intermediate key comes into one partition. The outputs of mappers after par
 titioning are saved into local disk. Map task sends the filenames and their locations to the master application. 6.
 Master initiates the slave application with reduce task and informs the locations of the partitions generated by the
 completed map tasks. The Reducer task reads its corresponding partition files generated by different map tasks
 and sorts by intermediate keys. 7. Once sorting / shuffling is done, the reduce task processes the intermediate key
 s value pair by executing the reduce function () as defined by the user and generates new key value pairs. The
 final key value pairs generated by reduce task are written into the disk. 8. The status of completion of all map a
 nd reduce tasks is taken care by the Applications Master application. It tries to reschedule the assigned but not
 completed map or reduce tasks to other slaves. The Fig<> indicate the execution workflow in map reduce fram
 e works. 2Fig: Execution Workflow in Map Reduce framework Apart from automatic parallelizing of work anothe
 r key feature of map reduce frameworks is its availability where master regularly pings to slaves to determine the
 liveliness of the nodes. Master considers a slave as failed node if there is no response for a stipulated time inte
 rval. Master does not schedule future tasks onto the node which are categorized as failed nodes. All the map tas

Sue
 PRINCIPAL

Geethanjali College of Engineering and Technology
 (Autonomous)
 Cheeryal (V), Keesera (M), Medchal Dist. (T.S.) - 501 301

ks that are scheduled on the failed nodes are rescheduled to other slave nodes. The other important key feature of map reduce framework is that it is locality based. The input file is divided into chunks of equal size usually (64MB/128MB) and distributed over multiple nodes of the cluster. Master application schedules the map tasks to the slaves by consider the locality of data to reduce the network traffic. Master tries to schedule a map task whose input chunk of data is available on the slave node which is called as data local data locality. If such machine is not idle then the master finds another free slave machine on the same rack which is called as intra rack data locality. If intra rack node is not free then the master looks for a machine in the same network switch for scheduling the map task which is called as inter rack data locality.

3.2.1. MAP REDUCE FRAMEWORK IN HADOOP

Hadoop is an open source framework provided by Apache Software Foundation that stores and process big data on a cluster of machines. Hadoop is widely adopted due to its parallel processing capabilities of huge data. It uses HDFS for distributed data storage and map reduce framework to process the data stored on HDFS parallelly. HDFS uses 2master slave architecture where master process is called as NameNode. NameNode stores the metadata of HDFS and keeps track and control of all the files stored across the cluster. The actual data is stored and processed in the nodes across the cluster which are called as data nodes. NameNode splits the input files into chunks of size 64MB/128MB by default on MR1/MR2. The chunk size is a user configurable property named as dfs.block.size in the file hdfs-site.xml. The blocks are copied into HDFS of data nodes with a replication of each block on multiple nodes for high availability and fault tolerance implementation of hadoop. By default the replication factor is 3 i.e. each chunk is copied into three data nodes. Replication factor is also a user configurable property named as dfs.replication in hdfs-site.xml file. NameNode uses the mechanism of heartbeat to detect the failures of data nodes in the cluster. Every data node sends a heartbeat message at regular interval as specified in the property dfs.heartbeat.interval in hdfs-site.xml which is by default 3 milliseconds. If the NameNode does not receive the heartbeat message from a DataNode for a maximum time period as given in the property dfs.namenode.state.datanode.interval in hdfs-site.xml which is by default three times the heartbeat interval, it considers the data node as a failed node. NameNode uses metadata to copy data available on the failed node to another node. HDFS framework is completely implemented for the distributed management of data over cluster of machines. Hadoop uses map reduce framework for processing of data in HDFS. Map Reduce framework of Hadoop also uses master slave architecture. MRv1 calls master process as Job Tracker and slave process as Task Tracker where Task Tracker runs on each node in cluster. Job Tracker executes the jobs submitted by client by talking to NameNode for determining the location of the data splits of that particular job. Job tracker creates the map tasks based on the number of splits of input data and schedules them to the task trackers based on the data locality. After assigning the works to task trackers, job tracker monitors the progress of work assigned to task tracker and submits the status of program execution to the client. MRv2 [17] replaces job tracker and task tracker with three processes namely Resource Manager (one per client), Application Master (one per application) and Node Manager (one per node). The functionality of job tracker process is divided into two processes:- Resource Manager and Application Master. Resource Manager: Cluster has one Resource Manager which is the controller of resources of jobs submitted to the hadoop cluster. It includes two main sub processes named as scheduler and Applications Manager. Scheduler performs scheduling of the resources based on the requirements of the user submitted applications. Resources of the cluster are allocated as containers. Resource Manager is responsible to allocate any cont

ainer to applications. ApplicationsManager module of Resource Manager performs functionalities of accepting jobs submitted to Hadoop cluster, negotiation of resources for initiating the Application Master of the job and maintains various applications submitted to the Hadoop cluster. Application Master: ApplicationMaster process is created per application once the application is scheduled to start its execution. Application Master manages the execution flow of the job by requesting resources from Resource Manager. It performs the complete life cycle of application and keeps track of the progress of application. It maintains and updates the metrics related to the execution of the application. 2Application Master requestsResourceManager for resources through AllocateRequest specifying the number of containers with the required resource configuration indicating memory, disk, CPU resources, node location, etc., The Resource Manager responds to the Application Master by informing set of newly allocated containers, current state of available resources using AllocateResponse class. Application Master of every application regularly sends heartbeat messages to the Resource Manager to indicate the liveness of the Application Master. All the resource requests for application are also made along with the heartbeat message. Application Master updates the information regarding the required containers and released containers to the Resource Manager. Resource Manager responds to Application Master indicating the list of nodes satisfying the resource requests provided by the Application Master. Application Master communicates with the Node managers to allocate containers to schedule its Map and Reduce tasks. Node Manager: This is a per node process which creates containers on the node to execute tasks, monitors the resource usage of containers and reports the same to Resource Manager in regular intervals. Node Manager creates resource containers with arbitrary size in contrast to fixed number of slots of fixed size for Map and Reduce tasks in MRv1. Node Manager keeps track of the health of the node on which it is executing and sends heartbeat message to Resource Manager to specify the liveness of the node. Node Manager on communication with Application Master creates containers for tasks. Node Manager also updates the Resource Manager with the container statuses running on the node like number of containers completed. It includes various components named ResourceLocalizationService which is responsible for downloading the required input data needed by containers by implementing the required access control restrictions, ContainersLauncher which maintains a pool of threads which are used to prepare and launch containers, ContainerMonitor monitors the resource usage of each container not to exceed its allocation and kills such containers. 3.2.2. Map Reduce Application workflow in Hadoop Application workflow in Hadoop framework [17] as given in Fig 1.2 is Step 1: client submits an application to the Resource Manager Step 2: Resource Manager allocates a container for Application Master Step 3: Application Master registers with Resource Manager Step 4: Application Master requests Resource Manager for containers with resource specifications required to run the tasks. Step 5: Resource Manager responds by providing the list of node managers that can be used for creation of containers in accordance with the scheduler Step 6: Application Master requests node managers to launch containers Step 7: map task/reduce task is executed in the container Step 8: client can monitor application status by contacting to Resource Manager/Application Master Step 9: Application Master unregisters itself from Resource Manager once all tasks finish execution. Fig.1.2: Application workflow in Hadoop 3.3. BIG DATA ON CLOUD Big data and cloud computing are two fundamental technologies used very commonly for modern day applications. Big data is all about dealing with large scale of data which may be structured, unstructured or semi structured, whereas Cloud computing is all about providing infrastructures or platform or software applications over the internet. Cloud co

Computing and big data is a perfect combination [10] where the resources required by big data and business analytics can be easily acquired by cloud which leverages a cost effective and scalable solutions for many businesses by reducing the investment cost of infrastructures needed for the applications. Cloud Computing: Cloud computing uses virtualization of computing resources to run multiple virtual machines on a physical machine. It enables elastic and highly available computing resources with low prices using pay-as-you-use model. Cloud computing offers three types of services IaaS (Infrastructure as a Service):- It enables users to use unlimited storage and processing power and pay as per the usage on hourly basis. It is a cost effective solution for enterprises where expenses related to owning infrastructures, maintain the infrastructures can be avoided. PaaS(Platform as a Service) :- It provides a development and deployment environment for application development over the internet. It provides resources with required runtime environments, libraries, databases, object storage etc., SaaS(Software as a Service):- It provides the cloud based applications with the required software licensing and delivery on a subscription basis over the internet. Users can use the services of the cloud based on their requirement whether only the bare infrastructure is needed or with some software environments or a built in application itself. Cloud Architecture: Cloud is usually deployed in three models: private, public and hybrid. Private Cloud: These are dedicated resources available at one organization and do not share the resources. The resources can be used from in-house or from external sources. Public cloud: This is a computing services provided by cloud providers over the internet. It allows general public to use and pay as per their usage of the computing resources or applications. It shares the infrastructure available at cloud provider end to multiple users using virtualization with security concerns and performance guarantees. Hybrid cloud: It is a combination of private and public cloud deployments. The hybrid cloud enables to deploy the base load with in-house resources and rent the cloud resources for a short period to meet the increasing demand of resources or functionality organization can decide what functionality or data to be implemented on in-house cloud and what functionality or data to be moved to cloud. Cloud computing enabled many startups and enterprises to rent computing resources and storage of any size with high availability, scalability and reliability. Most of the cloud based big data processing adopts Hadoop for data processing. Map Reduce is the standard used for such large big data processing framework. There are different ecosystems of hadoop used for different type of applications like Hive, Pig etc. Hive, Pig work on top of Hadoop which process huge data sets that cannot be handled by traditional database systems where the SQL like queries are transformed to Map Reduce jobs. There are multiple benefits of preferring big data processing on cloud [11, 12] like simplified infrastructure, reduced capital expenditure, security and privacy implementation by cloud provider, efficient usage of resources by virtualization, elasticity with automatic scalability, high availability and fault tolerance implementation. Cloud providers for BigData: Many cloud service providers offer establishment of Hadoop clusters which can be easily scaled automatically based on the demand of the resources needed to complete the submitted jobs. Some of the commonly used cloud providers for big data processing [13] are 1. Amazon EMR :Amazon Elastic Map Reduce(EMR) offers big data cloud services of Hadoop. It also supports other popular big data tools like Hive, HBase, Pig, Spark which can be integrated with other Amazon services like Amazon EC2, S3 for computing power and storage service 2. HDInsight:HDInsight is a big data processing tool provided by Microsoft Azure. It supports big data tools like Hadoop, Hive, HBase, Spark, R, etc. It provides easy plugins for IntelliJ and Eclipse versions. 3. Data lake Analytics:It is other big data cloud service provided by Microsoft



which is based on YARN. It is easy to use where developers need to know only one language U-SQL. It can be easily integrated with visual studio. The data lake provides the pricing based on the job, instead of hourly model as used by many other cloud providers.

4. Oracle Big Data cloud service:- It provides automated big data analytics services using Hadoop and Spark. It can be easily integrated with Oracle database and other big data cloud services like Big data SQL cloud service, Big data cloud machine. It also supports spatial and graph processing and analytics using R.

5. Qubole:-It is the first autonomous big data platform. It can easily form mashup with AWS, Oracle cloud and Microsoft Azure. It supports Hadoop, Hive, Spark, Presto, etc.

6. Google Big Query:-It is easy to use cloud service and has a reputation of fast and high performance for various benchmark tests. It offers a free tier upto 1 TB of analyzed data and 10 GB of stored data. It can be easily integrated with Google Cloud platform Artificial Intelligence and Machine Learning tools.

7. IBM Big data Solutions:-It provides features such as store, manage and analyze data. It supports various data sources like business analysts, Data scientists and supports popular data bases like DB2, Infosphere. IBM BigInsights on cloud provides Hadoop as a service on IBM cloud infrastructures.

2CHAPTER 4: PROPOSED WORK

The proposed method for dynamic load aware scheduler for map reduce tasks is designed to schedule the tasks based on job resource requirements and node resource availability. Nodes are associated with tag based on the prediction of node resource availability indicating whether the node is CPU heavy or IO heavy. Jobs are associated with tag indicating whether the map/reduce tasks are CPU intensive or IO intensive. If a node is found to be CPU heavy then the job which is less CPU intensive is more suitable.

Algorithm ScheduleJob(Queue q, Node n)

1. LeafQueue queue = ((LeafQueue) reservedApplication.getQueue());
2. Check for reserved applications
3. If (available reserved applications)
 - 3.1 submit(reservedapplicationid, Queue, Node)
 4. else
 - 4.1. for(job: jobs in queue)
 - 4.1.1. Pick the jobs {Ji} which have node local data
 - 4.1.2. Pick the job in {Ji} such that (job.tag AND node.tag==0)
 - 4.1.3. Resource requirement vector=ResourceCalculator(applicationid)
 - 4.2. end for
 - 4.3. check for resource availability on the node
 - 4.4. submit(applicationid, Queue, Node)
 5. end if
 6. end algorithm

The proposed model is designed and implemented in two different frameworks:

 1. Actual map reduce framework using Hadoop 2.5.2 version.
 2. Simulation using CloudSimEx which is a map reduce simulator for cloud.

4.1. PROPOSED DYNAMIC LOAD AWARE SCHEDULER FOR HADOOP MAP REDUCE FRAMEWORK

Hadoop 2.5.2 is used as base version for contribution of code to determine resource usage characteristics in terms of its CPU usage and IO usage. This research work includes the contribution for the following work under Hadoop package

 1. Dynamic load aware scheduling of map/reduce tasks for capacity scheduler.
 2. Determination of resource usage characteristics of map reduce tasks.
 3. Prediction of node resource availability status for datanode.

The job resource requirement characteristics and virtual machine resource availability are determined and are used in scheduling decisions. The process used for determination of job tag and node tag is given in section 2.1.2 and 2.1.3. Before allocating a container to the node a check is done by comparing the node tag and job tag from multiple leaf queues to pick the job that suits the node load without deviating from maximum capacity guaranteed and in consideration with data locality.

Algorithm assignContainers(Resource clusterRes, FicaSchedulerNode node)

 1. alloc=false;
 2. turn=1;
 3. count=0;
 4. size=activeApplications.size();
 5. for(FicaSchedulerAppapp:activeApplications)
 - 5.1. if(load-aware)
 - 5.1.1. count++;
 - 5.1.2. if node has data local for application app
 - 5.1.2.1. selectedapp=app
 - 5.1.3. n=nodetag of node;
 - 5.1.4. j=jobtag of app;
 - 5.1.4.1. if(j==0)
 - 5.1.4.1.1. selectedapp=app
 - 5.1.4.1.2. match=true
 - 5.1.4.2. else
 - 5.1.4.2.1. if(j AND n=0)
 - 5.1.4.2.1.1. select


PRINCIPAL

Goethanjali College of Engineering and Technology
(Autonomous)
Cheerla (V), Keesara (M), Madhachal Dist. (T.S.) - 501 301

edapp=app 5.1.4.2.1.2. match=true 5.1.4.2.2. end if 5.1.4.3. end if 5.1.4.4. if (match=false) 5.1.4.4.1. if(count<size) continue 5.1.4.4.2. else 25.1.4.4.2.1. if(turn=1) 5.1.4.4.2.1.1. turn=2 5.1.4.4.2.1.2. return NULL_ASSIGNMENT 5.1.4.4.3. end if 5.1.4.5. end if 5.1.5. end if 5.1.6. CSAssignment assignment=assignContainersToNode(node, selectedapp) 5.1.7. return assignment 5.2. end if 6. end for

The above algorithm is evaluated by establishing a Hadoop cluster on private cloud environment and public cloud environment. The following sections describe the process for determination of job tag and node tag.

4.2. Determination of Resource Usage Characteristics for Map Reduce Tasks

Determination of the resource usage behavior for a map/reduce is done by using the statistics generated by the task tracker during the task execution. The knowledge determined by initial map/reduce tasks can be applied for future map/reduce tasks of the same job as the functionality is same for all map/reduce tasks of the same job as well as the amount of input data processed by all map tasks is same except for the last map task.

2Resource usage characteristics of map/reduce task is analyzed and represented by a two bit tag where least significant bit is used to denote IO usage and the most significant bit is used to denote the CPU usage with the interpretation as given in Table 2.1. This two bit tag represents the CPU and IO requirements of a task which enables to schedule the future tasks with respect to their CPU and IO usages.

Tag	Tag value	Interpretation
00	0	CPU light and IO light process
01	1	CPU light and IO heavy process
10	2	CPU heavy and IO light process
11	3	CPU heavy and IO heavy process

TaskAttemptImpl class is being modified to use UpdateCounters() method to determine the CPU and IO tags. The CPU_MILLI_SECONDS, GC_TIME counters of the task are used to determine CPU tag and the counters HDFS_BYTES_READ, HDFS_BYTES_WRITTEN, FILE_BYTES_READ, FILE_BYTES_WRITTEN are used to determine IO tag from the set of counters related to CPU and IO usages.

Determination of IO tag & CPU tag for task

IO tag is determined by using above mentioned counters which indicates the number of bytes read and written from local file system and HDFS. Total number of bytes transferred= HDFS_BYTES_READ +HDFS_BYTES_WRITTEN +FILE_BYTES_READ + FILE_BYTES_WRITTEN Total bytes(in MB)= Totalnumber of bytes /10241024 2IO throughput = Total bytes/MB elapsed time Total number of bytes in MB within the elapsed time is used to determine IO throughput value. IOthroughput value calculated for the task is used to determine the IO tag which indicates whether the task is IO Heavy or IO light. If IOthroughput is greater than 5 then iotag=1 otherwise iotag=0 The counter values CPU_MILLI_SECONDS, GC_TIME, elapsed time are used to determine CPU usage characteristics of a task and in turn for a job CPU utilization= Total time elapsed time *100 / (CPU_i+GCTIME elapsed time *100) If a task has CPU utilization greater than 50% then it is considered as CPU heavy else as CPU light task.

4.3. Prediction of Node Resource Availability Status for Datanode

Shell script is written to determine the node resource usage pattern (Node Status) in terms of CPU heaviness and IO heaviness. This script is executed at regular intervals of time of 2 seconds. Node tag of two bits is associated with the datanode where the MSB is used to represent the CPU tag and LSB is used to represent the IO tag. The CPU tag is set to 1 if CPU utilization is greater than 50% and otherwise to 0. IO tag is set to 1 if IO utilization is greater than 50% and otherwise to 0. The Table 2.3 indicates the tag values that represent the current load on the datanode.

NodeTag	Node Tag Value	Description
00	0	CPU free IO free
01	1	CPU free IO busy
10	2	CPU busy IO free
11	3	CPU busy IO busy

iostat -x command is used to determine the CPU utilization and IO utilization of the node. iostat with -x option is used to display the extended statistics. The details regarding the CPU utilization %use

r, %nice, %system indicates percentage of CPU utilization that occurred by executing at the user level, user level with nice priority and system level respectively and disk IO %utilization is to be retrieved from the output of iostat command. Linear regression is used to estimate the CPU utilization and IO utilization of the node using the previous history of the node. Linear regression is used to estimate the relationship between one dependent variable and one or more independent variables. In this proposed model for node load prediction in terms of CPU and IO is done using linear regression where dependent variable is considered as the current CPU(or IO) utilization and the independent variable is previous time interval CPU (or IO) utilization. Linear regression model is used to predict the value in the next time unit from the learning done from previous utilizations. It is an efficient technique for accurate estimates. The Linear model for the prediction used is in the form of equation 1. $Y_t = \theta_0 + \theta_1 Y_{t-1} + \dots$ (1) Where, Y_t means Y measured in time period t. In this model Y_t is calculated from the previous Y_{t-1} . This model estimates the next CPU utilization of a virtual machine by considering the previous CPU utilizations. If C_{t-1} is CPU utilization at time t-1 and C_t is CPU utilization at time, t then there is a linear relation between C_t and C_{t-1} as given in equation 2. We used least squares method to determine the relationship between the CPU utilizations at different time slots. $C_t = \theta_0 + \theta_1 C_{t-1} + \dots$ (2) Where $\theta_0 = (C_t - C_{t-1}) / (C_t - C_{t-1})$ and $\theta_1 = (C_t - C_{t-1}) / (C_t - C_{t-1})$. Similarly linear regression is applied for IO utilization estimation for next time slot from previous IO utilizations as given in equation 3. $I_t = \theta_0 + \theta_1 I_{t-1} + \dots$ (3) Where $\theta_0 = (I_t - I_{t-1}) / (I_t - I_{t-1})$ and $\theta_1 = (I_t - I_{t-1}) / (I_t - I_{t-1})$. The shell script that determine the near future CPU and IO utilization of a particular node using linear regression is being invoked in monitoring thread of NodeResourceMonitorImpl class of Hadoop package. The CPU tag and IO tag is calculated at regular interval of time in the monitoring thread. The tag specifies the next CPU and IO utilization estimation based on previous history using linear regression. This information is passed to Resource Manager through heartbeat.

CHAPTER 5: IMPLEMENTATION AND RESULTS

5.1. Contributions to Hadoop framework

To implement the proposed dynamic load aware scheduler for hadoop framework. Capacity scheduler is modified to include the dynamic decisions for scheduling based on the resource usage of tasks and resource availability of the nodes. Capacity scheduler uses hierarchical queues for job submissions with capacity guarantees to each queue. Capacity of each queue indicates the percentage of resources available for applications submitted to that queue. Hierarchical queues are used to provide resource distribution, access restrictions for various organizations and groups utilizing the cluster resources. The root queue is the top level predefined queue available under which different parent queues can be created for different organizations. The parent queues can further be divided into child queues where each organization / group can divide the cluster resources into multiple child queues indicating sub organizations is created in a parent queue using the property `yarn.scheduler.capacity.<queue-path>.queues` in the `capacity-scheduler.xml` file. Queue path is the full path of the queue starting from root in the hierarchy. Example to create the parent queues under root queue `<property> <name>yarn.scheduler.capacity.<queue-path>.queues </name> <value> Q1,Q2,Q3</value> <description> Parent queue under root </description> </property>` The above property create three queues namely Q1, Q2, Q3. Once the Parent queues are created then the cluster resources can be divided among them specifying the percentage of resources guaranteed by using the property `yarn.scheduler.capacity.<queue-path>.capacity`. Parent queues can be used to directly to submit the applications or further divide into child queues. Leaf queues of this hierarchical queue structures are the only queues to which the applications can be submitted for the Hadoop cl

uster. Jobs in leaf queues are scheduled in FIFO manner which can be configured to use Fair scheduler instead of FIFO. Initially the capacity is given to the applications submitted to the leaf queues of an organization within capacity guarantees. If any organization does not require the capacity granted then the resources will be shared to the leaf queues of other organizations. Resource Manager always ensures the capacity guaranteed to the organization and manages the cluster utilization among multiple organizations using the cluster (allowing organization to use excess capacity which is not being used by others). Different parameters are available in capacity-scheduler.xml using which we can configure the properties like maximum capacity, minimum-user-limit-percent, user-limit-factor, maximum-allocation-mb, maximum-allocation-vcores, etc., Other properties in capacity-scheduler.xml are used for specifying the application limits of running and pending applications, queue administration and permissions, queue mapping, queue lifetime, applications priority etc. assignNodeLocalContainer() assignRackLocalContainer() assignOffswitchContainer() allocateContainersOnNode() ParentQueue.assignContainers() ParentQueue.assignContainersToChildQueues() LeafQueue.assignContainer() Containers are allocated on response to NODE_UPDATE event received from a node. NODE_UPDATE event is handled by invoking the nodeUpdate(RMNode) in the scheduler classes (FIFOScheduler/FairScheduler/CapacityScheduler based on the configuration) which updates the information of the newly launched containers and completed containers before allocating the new containers using allocateContainersOnNode(). Allocation of containers is done in the order of data locality as shown in Fig <>. Fig: Hierarchy of data locality Capacity scheduler of Hadoop uses queue comparator to determine the queue which is having less resources compared to the capacity guaranteed using usedCapacity(). Once the leaf queue is selected by the capacity Scheduler as given in Fig<>, then it assigns containers to the task with data locality preference with methods as given in fig ____.

2 PQ LQ1(20%) LQ2(40%) LQ3(40%) root Fig: Capacity scheduler container allocator internals Once the leaf queue with less resource allocation compared to minimum guarantee is selected, then the data locality preference and priority is used to select a task from the current active applications under that leaf queue. If a task with data local to node is not available then a rack local task is chosen where scheduling is always done in consideration with capacity guaranteed, priority and data locality in the capacity scheduler provided by hadoop.Yarn.scheduler.capacity.node-locality-delay property of capacity-scheduler.xml specifies number of missed opportunities for scheduling and capacity scheduler can schedule task on rack local container. Example: Fig: Queue hierarchy example For example if the queue capacities are given as leafqueue1(LQ1) as 20%, Leafqueue2(LQ2) as 40% and leafqueue3(LQ3) as 40% as given in fig <>. If number of machines in the cluster is 100 then 20 machines are to be given to LQ1, 40 machines to LQ2 and 40 machines to LQ3. If any queue does not require the allocated capacity then the resources can be shared to other queues. Queue can get back minimum guarantee resources when required and share its resources to other queues when not required. If currently 20 machines are given with LQ1, 35 machines to LQ2 and 38 machines to LQ3 and on receiving a NODE_UPDATE event it selects the LQ2 by queue comparator as it has less resources than minimum guarantee provided. On selecting LQ2, it considers the tasks of active applications (applications under execution) in the order of priority and data locality and schedules them by creating a container on the node. activeApplications is implemented as TreeSet<applicationComparator> where the comparator is based on order of application id and priority. Capacity Scheduler has two modes using which the timing for allocation of containers is decided as Asynchronous or Synchronous. The property yarn.scheduler.capacity.schedule-asynchronously.enable property

y in capacity- scheduler.xml file enables to set asynchronous scheduling of capacity scheduler. By default ⁶¹ this property is set to false. If this property is set to true then an asynchronous scheduling of tasks is performed by capacity scheduler. In asynchronous mode of scheduling (AsynchronousScheduleThread), the schedule() method invokes allocateContainers() method regularly at an interval of 100ns. It allocates the containers based on the resource availability and the requests made by Application Master, whereas if asynchronous mode is set to false then the capacity scheduler allocates containers synchronously with the NODE_UPDATE event. The proposed work is to modify the scheduling decision of capacity scheduling where the leaf queue is selected ² based on the resource usage tag of the job in the leaf queue and the 2node resource availability tag satisfying capacity guarantees. To create the new configuration property in capacity-scheduler.xml for consideration whether load awareness to be included in scheduling decisions the configuration variables NM_LOAD_AWARE, DEFAULT_NM_LOAD_AWARE are created in YARNConfiguration class and the new property labeled as loadaware is created in capacity-scheduler.xml with a default value to false. If Hadoop users want to use the proposed loadaware scheduler this property is to be set to true.

5.2. Implementation for node resource status tag determination NodeResourceMonitor program is written to determine the node tag which runs as a ⁶¹ continuous thread to monitor the node resource usage. NodeResourceMonitor thread is started only if loadaware ⁶¹ property is set to true in the configuration file of capacity- scheduler.xml. This thread invokes the shell script written for linear regression for determination of node tag from the previous CPU and IO utilizations using Runtime process as Process p=Runtime.getRuntime().exec("bash /home/sreelu/script5"); Where script5 file complete path is "/home/sreelu/script5". Once this script is executed the tag value is set to the node by reading the output of the created Runtime process. ⁶⁴ get methods are provided in this thread to enable the NodeManager to retrieve the node tag when required. Every NodeManager creates aNMContext with the modified NodeResourceMonitor to invoke the monitor thread. createNMContext(_____, nodeResourceMonitor) 2Once contribution for the determination of node tag is done for NodeManager, there needs to be communication of this tag to the ResourceManager to enable loadaware scheduling. The node heartbeat is used for the communication of node tag between NodeManager and ResourceManager. NodeHeartbeatRequest, NodeHeartbeatRequestPBImpl classes are modified to create an overridden constructors for node heartbeat with an additional parameters of node tag and get and set methods for node tag. NodeStatusUpdaterImpl class is being modified to use overwritten HeartbeatRequest only if load aware property is set by the user in capacity- scheduler.xml. Otherwise the original HeartBeatRequest without node tag gets invoked. NodeManager class updates the node tag to the Resource Manager using the modified HeartBeatRequest message which is used in scheduling decisions. Classes that are modified: FicaSchedulerNode, RMNode, RMNodeImpl, ResourceTrackerService. Node status update to the Resource Manager: Once the node status is determined, it is communicated to the Resource Manager using heartbeat message. To implement the required process of node status determination the following change were made in hadoop package: 1. Inclusion of Boolean variable named loadaware in yarn_server_common_service_protos.proto 2. All the modules modified in Hadoop package are being written in consideration with the loadaware variable to differentiate if the loadaware property is set by the user. 2NodeResourceMonitorImpl class continuously runs a Monitoring thread to determine tag for a node in terms of CPU utilization and IO utilization. The tag determined is used by the Node Manager and passed to Resource Manager through the heartbeat message. Sample screenshot indicating node tag determination is shown in Fig.____ Fig: Sample screenshot


PRINCIPAL
Geethanjali College of Engineering and Technology
(Autonomous)
Cheerlay (V), Kamsani (M), Medchal Dist. (T.S.) - 501 501

enshot indicating node tag determination 5.3. Evaluation of Node resource status prediction: To evaluate the proposed prediction model of linear regression for prediction of node resource status another bash script is written to monitor the real CPU and IO utilizations using iostat and the awk script as mentioned in previous section. The observed values are compared with estimated values using, Mean Absolute Error (MAE), Root Mean Square Error (RMSE). To check the correctness of the node status prediction different loads are created on the datanodes using the tool stress-ng tool [31, 32]. Stress-ng is a tool available of Linux operating system which can be used to load and stress system for different performance testing features like CPU, I/O, disk performance, VM stress, memory. It has different stress test related to CPU performance tests using floating point integer control flow and bit manipulation. To perform CPU stress test we use --cpu option or -c option. Examples for usage of stress-ng: i) stress-ng --cpu 2 Where 2 denotes number of spawn workers using sqrt() function for stress on CPU ii) Stress-ng -io 4 2 Denotes 4 spawn workers using sync() function for stress on io iii) Stress-ng -vm N is used to create N spawn workers using malloc() function for stressing VM iv) Stress-ng --cpu 2 -timeout N We can specify timeout after N seconds using the option timeout v) Stress-ng --cpu 2 -p 40 Stress CPU using 2 spawn workers upto cpu utilization percentage of 40%. Stress-ng tool is used to stress the node in terms of CPU and IO to check for evaluation of node prediction model. Four test cases are used to evaluate the prediction model. Test Case1: No stress Test Case2: Stress on CPU Test Case 3: Stress on IO Test Case 4: Stress on CPU and IO Test case samples of 100 are used and when merged on the common field of time between estimated utilizations and real utilizations of CPU and IO, lead to 90 to 95 samples. Table 2.4 gives the error value calculations for CPU utilization and IO utilization respectively for different test cases as mentioned above Table 2.4: Linear Regression for CPU and IO utilization prediction

Test case	CPU utilization prediction MAE	IO utilization prediction MAE	CPU utilization prediction RMSE	IO utilization prediction RMSE
Test case-1	0.087196	0.087933	0.001465	0.003689
Test case-2	0.100976	0.100887	0.011254	0.037082
Test case-3	0.120874	0.122632	0.109223	0.111756
Test case-4	0.185433	0.186654	0.190876	0.190437

The results in Table 2.4 indicate the correctness of the forecasting model when applied for CPU utilization and IO utilization. The model is observed to be efficient in determining the future CPU and IO loads based on the previous history of usages using linear regression for a virtual machine. 5.4. Implementation for job tag determination To determine the job tag of a particular application from the initially completed tasks, the following contributions were made to the Hadoop package. The counter values CPU_MILLI_SECONDS, GC_TIME, HDFS_BYTES_READ, HDFS_BYTES_WRITTEN, FILE_BYTES_READ, FILE_BYTES_WRITTEN of the task are used to determine job tag indicating whether a task is CPU intensive or IO-intensive as described in section 3.2. Update MilliCounters method of TaskAttemptImpl class (which is created for every task attempt of map/reduce tasks of a map reduce application) which updates and prints the different counters related to task execution like is being modified to determine the job tag. Counter values are retrieved from the TaskAttempt and used in the determination of the task tag. 2Ex: taskAttempt.getCounters().findCounter("org.apache.hadoop.mapreduce.Taskcounter", "CPU_MILLI_SECONDS").getValue() Once the initially completed tasks determine the task tag, the job is also associated with a tag by taking the average of initially completed task tags. Job class and JobImpl classes are modified to include the get and set methods for job tag from the task tags. Up to five task tags are used to finalize the job tag from the task tags as all the tasks of one job have same runtime behavior(All tasks apply same map function on same size of data) Classes modified are: ApplicationAttemptId, FicaSchedulerApp. The sample screenshot


PRINCIPAL
 Geethanjali College of Engineering and Technology
 (Autonomous)
 Chennai - 600 044, Madurai Dist. (T.N.) - 501 305

nsnot for job tag determination is shown in fig.____ Fig: Sample screenshot indicating job tag determination 5.5 . Evaluation of Jog Tag determination HiBench[49] benchmarking tool of Hadoop is used for testing the proposed job tag determination built into Hadoop package. HiBench is a Big Data benchmark suite. It consists of benchmarks for Hadoop and Spark applications. HiBench includes different size of workloads in terms of tiny, small, large, and very large which can be specified in hibenconf file. The data size of the workload can be varied in the conf file of the corresponding workload. Test cases were run using RandomTextWriter, WordCount, Sort(large), Dfsioe, NutchIndex, Pagerank, Grep workloads of HiBench.

21. RandomTextWriter: This benchmark creates random text of specified size. **The generated data can be used as input for other benchmark programs.**

2. Sort: It sorts the given input file

3. WordCount: It determines the frequency count of each word in the input data.

4. TeraSort: It is a standard benchmark that performs sorting on tera bytes of data. The input data for tera sort can be generated by TeraGen program of Hadoop.

5. Enhanced DFSIO (dfsioe): It is used to test the I/O throughput of HDFS in the Hadoop cluster. It generates large number of writes and reads simultaneously. It measures the average I/O rate, average throughput of each map task created, and the aggregated I/O throughput of HDFS.

6. PageRank: This workload benchmarks PageRank algorithm for Hadoop. It is a search engine ranking benchmark where the web data (where the hyperlinks follow the zipfian distribution) is used for the data source.

7. Nutch indexing: This workload benchmarks the indexing sub-system in Nutch which is a popular search engine by Apache. This workload also uses the Web data whose hyperlinks and words follow the Zipfiandistribution. The tags obtained for these workloads are given in Table 2.2.

Table 2.2 : Experiment results for Job Tag determination

Benchmark	CPU Tag	IO Tag	obtained	Job Tag	obtained	Job characteristic
RandomTextWriter	0	1	01	1	IO intensive	
WordCount	1	0	10	2	CPU intensive	
Sort(large)	0	1	01	1	IO intensive	
Dfsioe	1	0	10	2	CPU intensive	
NutchIndex	1	0	10	2	CPU intensive	
2Pagerank	0	1	01	1	IO intensive	
Grep	1	0	10	2	CPU intensive	

From the results it is observed that sort, page rank, text writer are IO heavy jobs and word count, nutchindex, dfs are CPU heavy jobs.

5.6. Implementation of LoadAware Capacity scheduler for Hadoop Map Reduce framework

Capacity scheduler loadaware is written such that it gets only invoked if loadaware **property is set to true** by the user. If loadaware is set, ParentQueue class is traversed to a leaf queue which has an application with appropriate job tag with reference to the node **tag on which the container is to be created.** LeafQueue class **retrieves the job tag and the node tag to decide about schedulability of the task on the node.** If the leaf queue under consideration does not have job suitable to be schedulable on the node based on resource usages, then the next leaf queue of the parent queue is selected for testing schedulability of the tasks. This continues until a leaf queue with appropriate job schedulable on the considered node is obtained. If no such leaf queue is obtained then **it schedules the job in the first leaf queue with less resources against guaranteed capacity as done in default capacity scheduler.** To reduce the time incurred in searching the leaf queues until an appropriate job is obtained, Parent queue is associated with an array of tags of active applications in the leaf queues present in it, so that there is no need to traverse each leaf queue to search for an application. Looking at tags array of Parent queue, a decision is made whether to traverse the leaf queues of that parent queue or not.

2root PQ1(50%) PQ2(50%) LQ1(10%) LQ2(20%) LQ3(20%) LQ4(30%) LQ5(25%) Node1 (nodetag=1) Node2 (nodetag=0) Node3 (nodetag=1) Node4 (nodetag=2) 1 0 2 LQ1 LQ2 LQ3 LQ4 LQ5 1 0 Job Tags Job Tags Nodes Job Queus

Fig: Example of Load aware capacity scheduling Considering the default FIFO technique for the set of

jobs in a leaf queue and the leaf capacity configurations and Job queues as shown in fig < The Node with tag =1 is suitable for job in LQ2,LQ3 as the job tag=0 for LQ2 and job tag=2 for LQ3 (i.e Node tag=1 (IO heavy node) is more suitable for the job in LQ2 whose job tag=0 (CPU light and IO light) and for the job in LQ3 whose job tag=2 (CPU heavy and IO light)). Hence job task which is CPU intensive is scheduled on a node which is IO intensive. Node 2 with tag=0 is suitable for any leaf queue. Node 3 with node tag=1 is more suitable for the jobs in LQ2,LQ3,LQ5. Job task from leaf queue 2 gets scheduled on Node3. Node4 with tag 2 can schedule jobs from LQ1,LQ2, LQ4 and LQ5. Job task from LQ1 is checked initially for capacity guarantees and gets scheduled on Node4 if not meeting the capacity guarantees. Proposed dynamic load aware scheduler schedules the tasks from the queues in comparison with job tags and node tags as explained above with consideration of data locality.

5.7. Evaluation of Dynamic Load Aware Scheduler

The proposed Dynamic Load Aware Scheduler for Hadoop framework is evaluated on Private Cloud and public cloud environments.

5.7.1. Evaluation of Dynamic Load Aware Scheduler on Private Cloud
 The proposed scheduler is evaluated on private cloud setup using VMware tools: VMware ESXi server [50] and VCenter [51,52]. Four VMs are created with the following configurations: OS: Ubuntu14.0, Memory: 4GB Hard Disk: 200GB. To specify Hadoop framework to use the proposed load aware algorithm, the property yarn.node.manager.load-aware is specified to true in yarn-site.xml. Jobs RandomText Writer, WordCount, Pi, Sort, grep are used for evaluation and in order to have accurate results each pair of jobs is executed for two times and the average of execution time is considered. Two queues (default and alpha) are created with capacity configuration of 50% each. The makespan times of different test cases using Capacity scheduler and Capacity Scheduler Loadaware for data size of 1GB, 2 GB and 3 GB are specified in Table 3.1, Table 3.2, and Table 3.3 respectively. Their comparisons can be seen in graphs of Fig.3.1, Fig.3.2, and Fig.3.3.

Table 3.1: Makespan times for data size=1GB on private cloud

Test case	Job	run 1	run 2	Avg. ET (sec)	Make span	run 1	run 2	Avg. ET (sec)	make span diff %
RTW & WC	RTW	98	102	100	110	85	89	87	95.5
	WC	108	112	110	97	94	95.5	95.5	WC & Sort
WC & Sort	WC	104	101	102.5	102.5	89	87	88	88
	Sort	14.5	0.14	146	3	sort	99	98	98.5
Sort & Pi	Sort	84	82	83	RTW & sort	RTW	106	110	108
	Pi	108	112	118	115	115	-7	-0.06	481
Sort & Pi	Sort	100	96	98	110	107	108.5	Sort & Pi	sort
	grep	97	100	98.5	98.5	85	87	86	86
Pi & grep	Pi	80	90	85	69	70	69.5	Sort & Gre p	sort
	grep	92	96	94	94	80	83	81.5	81.5
Sort & Gre p	Sort	102	106	104	104	112	113	112.5	112.5
	grep	108	108.5	RTW & grep	RTW	102	110	106	106
RTW & grep	RTW	95	101	98	98	84	87	85.5	85.5
RTW & PI	RTW	12.5	0.126	90	4	pi	80	90	85
	PI	83	85	70	74	72	WC & grep	WC	102
WC & So rt	WC	102	106	104	104	112	113	112.5	112.5
So rt & Pi	So rt	92	90	91	80	78	79	RTW & PI	RTW
	PI	95	101	98	98	84	87	85.5	85.5
So rt & Gr ep	So rt	12.5	0.127	55	1	Pi	82	79	80.5
WC & gre p	WC	71	70	70.5	2	RT W & WC	WC	& So rt	RT W
RT W & g rep	RT W	0	20	40	60	80	100	120	CS
RT W & P I	RT W	0	20	40	60	80	100	120	CS

Table 3.2: Makespan times for data size=2GB on private cloud

Test case	Job	run 1	run 2	Avg. ET (sec)	Make span	run 1	run 2	Avg. ET (sec)	make span diff %
RTW & WC	RTW	124	130	127	133	106	110	108	116
	WC	17	0.1278	2	WC	131	135	133	114
WC & Sort	WC	133	136	134.5	134.5	118	114	116	116
	Sort	18.5	0.1375	46	sort	125	129	127	110
Sort & Pi	Sort	118	121	119.5	128.5	122	125	123.5	138
	Pi	-9.5	-0.0739	3	sort	128	129	128.5	140
Sort & Gre p	Sort	182	186	184	184	155	157	156	156
	grep	122	118	120	99	102	100.5	WC & grep	WC
WC & gre p	WC	138	140	139	139	145	14		

Sive
 PRINCIPAL

8 146.5 146.5 -7.5 - 0.0539 6 grep 110 109 109.5 119 120 119.5 RTW & grep RTW 127 133 130 130 112 110 111 111 19 0.1461 54 grep 114 113 113.5 99 96 97.5 2RTW & PI RTW 131 138 134.5 134.5 110 115 112.5 112.5 22 0.1635 69 Pi 86 86 86 80 76 78 RT W & W C WC & So rt RT W & s ort So rt & Pi So rt & Gr e p WC & gre p RT W & g rep RT W & P I 50 70 90 110 130 150 170 190 CS CSLA Map Reduce jobs M ak es pa n tim e (se c) Fig.3.2: Comparison of CS and CSLA on private cloud for data size 2GB Table3.3: Makespan times for data size=3GB on private cloud data size=3GB Capacity scheduler Capacity scheduler load aware Test case Job run1 run2 Avg. ET (sec Make span run1 run2 Avg. ET (sec Make span diff % RTW & WC RT W 171 167 169 169 143 125 134 134 35 0.20710 1WC 149 146 147.5 114 125 119.5 WC& Sort WC 150 149 149.5 149.5 121 119 120 120 29.5 0.19732 4 sort 139 136 137.5 114 112 113 RTW & sort RT W 162 165 163.5 163.5 172 173 172.5 172.5 -9 -0.05505 sort 139 144 141.5 154 152 153 Sort & Pi sort 132 137 134.5 134.5 116 114 115 115 19.5 0.14498 1 pi 129 132 130.5 101 104 102.5 Sort & Gre p sort 228 222 225 225 180 179 179.5 179.5 45.5 0.20222 2 grep 162 159 160.5 120 121 120.5 2WC & grep WC 152 154 153 153 160 163 161.5 162 -9 -0.05882 grep 151 154 152.5 159 165 162 RTW & grep RT W 171 174 172.5 172.5 141 139 140 140 32.5 0.18840 6 grep 161 154 157.5 118 120 119 RTW & PI RT W 178 183 180.5 180.5 137 138 137.5 137.5 43 0.23822 7 Pi 125 128 126.5 110 100 105 RT W & W C WC & So rt RT W & s ort So rt & Pi So rt & Gr e p WC & gre p RT W & g rep RT W & P I 50 70 90 110 130 150 170 190 210 230 CS CSLA Map Reduce jobs M ak es pa n tim e (se c) Fig.3.3: Comparison of CS and CSLA on private cloud for data size 3GB The above results in Table 3.1 to Table 3.3 indicate a good performance improvement from 12% to 23% for jobs of size 1GB, 2GB, 3GB when executed on private cloud of 4 machines except for the combination of jobs with very similar resource usage intensiveness like RTW and sort when executed simultaneously. RTW and sort are both IO intensive and because of which there is no advantage of this scheduler when both the jobs are grouped as a set of two jobs. Proposed model is also tested using combination of 4 jobs with two queues of 50% capacity each on four virtual machines. The set of four jobs with input data size of 3GB 2 are submitted with two jobs to each queue. Table 3.4 to Table 3.6 indicate the makespan time for different combinations of jobs and Fig. 3.4 provides the comparison of Capacity scheduler and capacity scheduler load aware on private cloud for four jobs with data size 3GB. Table3.4: Makespan times for data size=3GB on 4 VMs in private cloud n=4 Capacity scheduler Capacity scheduler load aware Test case run1 run2 Avg makespan (sec) run1 run2 Avg makespan (sec) diff % RTW, WC, sort, Pi 313 325 319 278 273 275.5 43.5 0.136364 RTW, sort, grep, WC 335 323 329 273 292 282.5 46.5 0.141337 Sort, Pi, WC, grep 303 310 306.5 319 315 317 -10.5 -0.03426 Sort ,Grep, Pi, RT W 337 345 341 303 292 297.5 43.5 0.127566 Table3.5: Makespan times for data size=3GB on 5 VMs in private cloud n=5 Capacity scheduler Capacity scheduler load aware Test case run1 run2 Avg. makespan (sec) run1 run2 Avg. makespan (sec) diff % RTW & WC, sort, Pi 296 305 300.5 250 244 247 53.5 0.178037 RTW, sort, grep, WC 308 315 311.5 246 266 256 55.5 0.17817 Sort, Pi, WC, grep 288 294 291 290 285 287.5 3.5 0.012027 Sort & Grep, Pi, RTW 327 322 324.5 278 264 271 53.5 0.164869 Table3.6: Makespan times for data size=3GB on 6 VMs in private cloud n=6 Capacity scheduler Capacity scheduler load aware Test case run1 run2 Avg. makespan (sec) run1 run2 Avg. makespan (sec) diff % 2RTW & WC, sort, Pi 278 285 281.5 222 214 218 63.5 0.225577 RTW, sort, grep, WC 289 299 294 216 237 226.5 67.5 0.229592 Sort, Pi, WC, grep 269 275 272 264 255 259.5 12.5 0.045956 Sort & Grep, Pi, RTW 311 307 309 249 237 243 66 0.213592 0 50 100 150 200 25


PRINCIPAL
 Goethanjali College of Engineering and Technology
 (Autonomous)
 Cheerayal (V), Kaesara (M), Medchal Dist. (T.S.) - 501 30

0 300 350 RTW,WC,sort,pi RTW, sort,grep,WC sort, Pi, WC, grep sort & grep, pi, RTW Map Reduce jobs M a k es pa n tim e(se c) Fig.3.4: Comparison of CS and CSLA on private cloud for four jobs with data size 3GB The above results in Table 3.4 to Table 3.6 indicate a good performance improvement from 12% to 22% for jo bs of size 3GB when executed on private cloud of 4 machines except for the combination of jobs with very simila r resource usage intensiveness like Sort, Pi, WC, grep when executed simultaneously. 5.7.2. Evaluation of Dyna mic Load Aware Scheduler on Public Cloud To evaluate proposed scheduler on public cloud a hadoop cluster i s created on Amazon EC2 [53] machines with basic OS as Ubuntu14.0. T2.medium instances were used by vary ing the number of instances for data nodes. Two Hadoop clusters were created simultaneously on EC2 machines by changing the configuration load-aware property created in Yarn-site.xml.sample screen is given in fig <. O ne cluster was set with loadaware property to true and the other cluster with loadaware property to false as give n in fig <. Both clusters were created with equal number of virtual machines and same test 2cases are executed to evaluate the performance of the proposed scheduling algorithm. Few sample screens are given in fig < to fi g <. Fig: Sample screen of Hadoop cluster creation 2Fig: Sample screen for load aware property 2Fig: sample screen of Hadoop test case execution Fig: Sample screen of applications submitted to Hadoop cluster on AWS 2 Fig: Sample screen of Hadoop application status on AWS Fig: Sample screen of Hadoop working nodes on A WS 2The makespan times of different jobs are evaluated on the multi node cluster created on virtual machines by changing the above mentioned property. Jobs RTW, WC, Pi, Sort, grep are used for evaluation and in order t o have accurate results each pair of jobs is executed for two times and the average of execution time is conside red. Table 3.7 to Table 3.9 indicates the makespan of different test cases for capacity scheduler and dynamic loa d aware scheduler and for data sizes of 1GB, 2GB, and 3GB.Their comparisons can be seen in graphs of Fig.3. 5, Fig.3.6, and Fig.3.7. Table3.7: Makespan times for data size=1GB on public cloud AWS data size=1GB Capa city scheduler Capacity scheduler loadaware Test case Job run 1 run2 Avg. ET (sec) ma k es pa n run 1 run 2 Avg. ET (sec) makes pan diff % RTW & WC RTW 128 132 130 133 116 113 114.5 114.5 18.5 0.1391 WC 1 30 136 133 106 107 106.5 WC& Sort WC 132 138 135 144 102 106 104 126 18 0.125 sort 146 142 144 127 125 126 RTW & sort RTW 125 130 127.5 127 .5 135 139 137 137 -9.5 -0.0745 sort 120 124 122 132 135 133 .5 Sort & Pi sort 121 126 123.5 123 .5 108 109 108.5 108.5 15 0.12146 pi 90 95 92.5 77 79 78 Sort &Grep s ort 123 121 122 122 108 105 106.5 106.5 15.5 0.1270grep 97 99 98 86 89 87.5 WC & grep WC 91 96 93.5 1 04 99 103 101 111.5 -7.5 -0.0721 grep 106 102 104 112 111 111.5 RTW & grep RTW 120 122 121 121 107 1 03 105 105 16 0.1322 grep 95 97 96 85 83 84 RTW & PI RTW 118 123 120.5 120 .5 105 107 106 106 14.5 0.1203Pi 93 89 91 84 82 83 2RTW & W C WC & So rt RT W & s ort So rt & Pi So rt & Gr ep WC & gre p RT W & g rep RT W & P I 50 60 70 80 90 100 110 120 CS CSLA Map reduce jobs M a k es pa n tim e(se c) Fig.3.5: Comparison of CS and CSLA on public cloud for data size 1GB Table3.8: Makespan times for dat a size=2GB on public cloud AWS data size=2GB Capacity scheduler Capacity scheduler loadaware Test case Jo b run 1 run 2 Avg. ET (sec) Make span run 1 run 2 Avg. ET (sec) Make span diff % RTW & WC RT W 174 178 176 176 152 150 151 151 25 0.1420 5 W C 176 173 174.5 149 150 149.5 WC& Sort W C 168 170 169 1 69 139 142 140.5 140.5 28.5 0.1686 4sort 140 146 143 120 125 122.5 RTW & sort RT W 180 178 179 179 1 89 189 189 189 -10 - 0.0559sort 131 138 134.5 148 142 145 Sort & Pi sort 138 140 139 141.5 118 120 119 1 21 20.5 0.1448 8pi 143 140 141.5 120 122 121 Sort &Grep sort 145 139 142 146 118 119 118.5 123.5 22.5 0.


PRINCIPAL
 Jeethanjali College of Engineering and Technology
 (Autonomous)
 Cheerayal (V), Keesara (M), Medchal Dist. (T.S.) - 501 301

1541 1 grep 147 145 146 125 122 123.5 WC & grep W C 130 128 129 155.5 137 140 138.5 162.5 -7 -0.045g
re 159 152 155.5 165 160 162.5 2p RTW & grep RT W 172 174 173 173 146 143 144.5 144.5 28.5 0.1647 4
grep 150 143 146.5 128 129 128.5 RTW & PIRT W 169 171 170 170 145 142 143.5 143.5 26.5 0.1558 8Pi
143 139 141 116 113 114.5 RTW & W C WC & So rt RT W & s ort So rt & Pi So rt & Gr ep WC & gre p
RT W & g rep RT W & P I 50 70 90 110 130 150 170 190 CS CSLA Map Reduce jobs M ak es pa n tim e(
se c) Fig.3.6: Comparison of CS and CSLA on public cloud for data size 2GB Table3.9: Makespan times for d
ata size=3GB on public cloud AWS data size=3GB Capacity scheduler Capacity scheduler loadaware Test case J
ob run 1 run 2 Avg. ET (sec) Make span run 1 run 2 Avg. ET (sec) Make span diff % RTW & WC RTW 212
209 210.5 217.5 172 175 173.5 180.5 37 0.170 11 WC 219 216 217.5 182 179 180.5 WC& Sort WC 212 218
215 215 176 175 175.5 175.5 39.5 0.183 72 sort 200 202 201 166 162 164 RTW & sort RTW 210 215 212.5
216 224 229 226.5 228 -12 -0.055 6 sort 213 219 216 227 229 228 2Sort & Pi sort 212 209 210.5 210.5 172
175 173.5 173.5 37 0.175 77 pi 170 172 171 127 130 128.5 Sort & Gre p sort 292 300 296 296 225 223 224 2
24 72 0.243 24 grep 210 208 209 166 162 164 WC & grep WC 205 208 206.5 208.5 225 221 223 223 -14.5
-0.069 5 grep 209 208 208.5 225 221 223 RTW & grep RTW 220 224 222 222 171 169 170 177 45 0.202 7
grep 210 207 208.5 178 176 177 RTW & PIRTW 219 225 222 222 168 172 170 170 52 0.234 23 Pi 166 170
168 130 129 129.5 RT W & W C WC & So rt RT W & s ort So rt & Pi So rt & Gr ep WC & gre p RT W
& g rep RT W & P I 50 70 90 110 130 150 170 190 210 230 CS CSLA Map Reduce jobs M ak es pa n tim
e(se c) Fig.3.7: Comparison of CS and CSLA on public cloud for data size 3GB The above results in Table 3.7
to 3.9 indicate a good performance improvement from 12% to 23% for jobs of size 1GB, 2GB, 3GB when exe
cuted on public cloud of 4 machines except for ²¹the combination of jobs with very similar resource usage intensiv
eness like RTW and sort when executed simultaneously. 2Proposed ²⁰method is also tested using combination of
4 jobs with two queues of 50% capacity each on four virtual machines. The set of four jobs with input data size
of 3GB are submitted with two jobs to each queue. Table 3.10 to Table 3.12 indicate the makespan time for di
fferent combinations of jobs and Fig.3.8 provides the comparison of Capacity scheduler and capacity scheduler lo
adaware on public cloud Amazon EC2 for four jobs with data size 3GB. Table3.10: Makespan times for data si
ze=3GB on 4 VMs in public cloud AWS EC2. n=4 Capacity scheduler Capacity scheduler loadaware Test case r
un1 run2 Avg. makespa n (sec) run1 run2 Avg. makesp an (sec) diff % RTW& WC, sort, Pi 420 413 416.5 36
2 365 363.5 53 0.12725 RTW, sort, grep, WC 428 432 430 383 382 382.5 47.5 0.11046 Sort, Pi, WC, grep 445
439 442 452 450 451 -9 -0.02036 Sort & Grep, Pi, RTW 415 410 412.5 358 355 356.5 56 0.13575 Table3.11:
Makespan times for job size=3GB on 5 VMs in public cloud AWS EC2 n=5 Capacity scheduler Capacity sched
uler loadaware Test case run1 run2 Avg. makespan (sec) run1 run2 Avg. makesp an (sec) Diff % RTW& WC,
sort, Pi 403 395 399 333 335 334 65 0.162907 RTW, sort, grep, WC 415 408 411.5 357 355 356 55.5 0.1348
72 Sort, Pi, WC, grep 419 430 424.5 423 425 424 0.5 0.001178 Sort & Grep, Pi, RTW 399 392 395.5 329 330
329.5 66 0.166877 Table3.12: Makespan times for job size=3GB on 6 VMs in public cloud AWS EC2 n=6 Cap
acity scheduler Capacity scheduler loadaware 2Test case run1 run2 Avg. makesp an (sec) run1 run2 Avg. makes
p an (sec) Diff % RTW& WC, sort, Pi 387 380 383.5 310 307 308.5 75 0.195567 RTW, sort, grep, WC 399 3
92 395.5 330 325 327.5 68 0.171934 Sort, Pi, WC, grep 402 412 407 397 398 397.5 9.5 0.023342 Sort & Grep,
Pi, RTW 377 383 380 303 304 303.5 76.5 0.201316 0 50 100 150 200 250 300 350 400 450 500 RTW,WC,so


PRINCIPAL

Geethanjali College of Engineering and Technology
(Autonomous)
Cheerayal (V), Kessera (M), Medchal Dist. (T.S.) - 501 301

rt, pi RTW, sort, grep, WC sort, Pi, WC, grep sort & grep, pi, RTW Map Reduce jobs. Makes a parallel execution.

Fig.3.8: Comparison of CS and CSLA on public cloud for four jobs with data size 3GB. The above results in Table 3.10 to Table 3.12 indicate a good performance improvement from 12% to 20% for jobs of size 3GB when executed on public cloud of 4 machines except for the combination of jobs with very similar resource usage intensiveness like Sort, Pi, WC, grep when executed simultaneously.

CHAPTER 6: SIMULATION OF DYNAMIC LOAD AWARE SCHEDULER FOR MAP REDUCE FRAMEWORK

The performance of scheduling policies in a real cloud computing environments like Amazon EC2, Google App Engine, Azure etc., is extremely challenging and costly. To simulate bigger workloads this proposed work has been simulated using the map reduce simulator CloudsimEx.

6.1. INTRODUCTION

6.1.1. CloudSim

CloudSim [54] provides the simulation of cloud environments by defining various classes for implementation of various components involved in the working of cloud. It has classes like DataCenter, DataCenterBroker, DataCenterCharacteristics for simulation of a datacenter. DataCenter class simulates infrastructure as a service model with multiple hosts forming a datacenter. DataCenterBroker class performs application scheduling and resource coordination. DataCenterCharacteristics class holds the configuration details of data center resources. Host class models a physical host in data center. Each host CPU unit is modeled using PE which is specified in terms of MIPS (million instructions per second). Multiple PE objects are configured on the Hosts with multiple cores. Applications running on CloudSim are modeled by Cloudlet class. Cloudlet represents the task to be executed. Cloudlets will be assigned to VMs maintained as VMList. Processing elements (PEs) of a Host are shared by virtual machines (VMs) running on that host. Two separate lists are maintained to maintain list of VMs and list of cloudlets. Once the simulator is started DataCenterBroker allocates the VMs on the hosts and schedules the cloudlets onto the VMs based on the VMSchedulerPolicy and CloudletScheduler policy respectively. All the components of CloudSim discussed above are termed as Entities and communication between entities is modeled by events. All the events between them are managed using two queues, future queue and deferred queue, where submitted events are initially placed in future queue and then moved into deferred queue based on the destination entity of the event. All the events are sorted by the time of generation of the event. All entities are programmed by event handling mechanism for different phases of execution of the tasks. Two entities are automatically started in CloudSim: 1. CloudSimInformationService (CIS) entity is responsible for registration of resources, discovery of resources, availability of the resources. 2) CloudSimShutdown (CSD) which is responsible to generate the event for end of the simulation to the CloudSimInformationService entity. Once the simulation is started CloudSim creates CIS and CSD entities. Next it creates the entities DataCenterBroker and DataCenter. DataCenter creates a hostlist comprising of the hosts present in the datacenter as mentioned in cloud.yaml file. It also creates VMlist and Cloudletlist and sends to the Broker entity. Once the hostlist, VMlist, Cloudletlist are ready all the entity threads are in running state and the simulation of CloudSim is started. All the entities in the simulation are handled and communicated through events passing activity. Every event is associated with a sourceID and a destinationID. The ids for the four entities of the CloudSim are given as Task Map, Task Reduce, Task Cloudlet id=0 for CIS, id=1 for CSD, id=2 to n for DataCenter and id=n+1 for broker. All the events that are communicated between the entities are assigned with a numerical value indicating event tag number which specifies regarding the event and the destination entity handles the event based on the event type.

6.1.2. CloudSimEx

CloudSimEx [55] is a map reduce simulator for cloud environments. CloudSi

CloudSimEx is an extension of CloudSim which is a modeling and simulation tool for cloud computing. It simulates a job as set of map and reduce tasks where map tasks can be assigned to multiple virtual machines and executed in parallel. Map task and reduce task are created as sub classes to Task class which is subclass of Cloudlet class as given in fig 6. Fig: Map Reduce classes hierarchy in CloudSimEx CloudSimEx provides simulation where different jobs can run simultaneously on multiple virtual machines. In CloudSimEx there is another entity named as MapReduceEngine that gets initiated along with CloudInformationService(CIS) and CloudShutdown. MapReduceEngine entity is for simulating the map reduce jobs execution flow where the job is divided into map tasks and the tasks are assigned to the cloudlets. The map reduce tasks represented in CloudletsList are scheduled to VMs on multiple Hosts of the datacenter using MapReduceScheduler. The default algorithms provided in MapReduceEngine are FIFO, Branch and Bound. As the proposed work is to design a scheduler for map reduce tasks of hadoop we have extended the CloudSimEx to simulate the functionality of Hadoop schedulers of map reduce tasks and designed the proposed dynamic loadaware scheduler for map reduce tasks. The configuration details of the cloud data centers are provided in simulation.properties file which indicates the cloud file property specifying the file which holds the information related to number of data centers, number of hosts, the configuration details of hosts as given in figure 7. Fig: Sample screen of application status in CloudSimEx 2 Fig: Sample screen of test case file in Cloud SimEx Fig: Cloud.yaml configuration of CloudSimEx 2

```

Process VMCreate()
After all VM's are created
Submit cloudlets
Reduce task
Map task
Check if all Map Tasks are finished
Select a proper VM based on tags of VM and task
SendNow(getVMstoDataCenterMap().get(VM.getID(), CLOUDLET_SUBMIT, cloudlet)
Cloudlet submitted
++ add cloudlet to CloudletSubmittedList
Select the first cloudlet for the selected VM
Cloudlet.setVMId(VM.getID())
CloudletSubmitted++ add cloudlet to CloudletSubmittedList
Remove cloudlet from cloudletlist
SendNow(getVMstoDataCenterMap().get(VM.getID(), CLOUDLET_SUBMIT, cloudlet)
Remove cloudlet from CloudletList
yesNo
MapReduceEngine divides the given job into map tasks and builds each task as a cloudlet.
Cloudlet is scheduled onto VM by setting the VMid to a Cloudlet.
Once the VM is decided for the cloudlet, an event CLOUDLET_SUBMIT is generated to Datacenter indicating the Cloudlet submit.
Cloudlet is added to cloudletsubmittedlist and removed
2Receive CLOUDLET_SUBMIT
Process CloudletSubmit(ev,false)
Cloudlet cl=(Cloudlet) ev.getData()
If cl.isFinished() then print cloudlet already completed
If (ack) sendNow(userid,CLOUDLET_SUBMIT_ACK,cl)
SendNow(userid, CLOUDLET_RETURN,cl)
Process this cloudlet
Set Resource Parameter
PredictFileTransferTime(FileTransferTime)
userid<- cl.getUserId()
vmid<- cl.getVMId()
Host host<-getVMAllocationPolicy().getHost(vmid,userid)
Scheduler =VM.getCloudletScheduler()
estimatedFinishTime= Scheduler.cloudletSubmit(cl, FileTransferTime)
Send (get ID(), estimatefinishtime, VM_DATACENTEREVENT) from cloudletlist.
If the task is a reduce task, a check is done if all the map tasks are done.
If all the map tasks of the corresponding reduce task finished execution then the cloudlet corresponding to that reduce task is also allocated to a VM as mentioned above.
Once the CLOUDLET_SUBMIT event is received by the datacenter, it is being handled.
Once the cloudlet is submitted, a check is done to determine whether the available number of PEs is sufficient for the cloudlet to run i.e. currentCPUs- usedPEs>=Cloudlet.getNoofPEs().
If the resources are not sufficient then the Cloudlet is 2Update cloudletProcessing()
For every Host in HostList
host.updateVMsProcessing(cloudsim.clock)
For every VM in that host
Vm.updateVmProcessing(currenttime, AllocateMIPSforVM)
getCloudletScheduler().updateVmProcessing(currenttime,MIPS share)
timespan =currenttime-getPreviousTime()
CPUS++
For each rcl: getCloudletExecutionTime()

```


PRINCIPAL
 Geethanjali College of Engineering and Technology
 (Autonomous)
 Chittoor (A), Kolleru (M), Madhachal Dist. (T.S.) - 501 301

nList() Rcl.updateCloudletFinishedSoFar() If no more cloudlets in this scheduler executionsize=0 and waitinglist size=0 then setPreviousTime (currenttime) Cloudlet Finish Set CloudletState= CLOUDLET_SUCCESS queued into waiting list until sufficient resources are available. Once the available resources are determined to be sufficient for the Cloudlet, it is executed and the execution statistics are calculated and returned. VM_DATACENTER-EVENT is generated to update status of the finished cloudlets. VM_DATACENTER-EVENT is handled by the DataCenter entity where every VM on every host is invoked for updating the processing status of different cloudlets scheduled on them. All the cloudlets that have finished execution are updated into cloudletsfinishedlist and cloudlet state is setted to indicate CLOUDLET_SUCCESS. All the PEs used by the finished cloudlets are updated into the UsedPEs. The CLOUDLET_RETURN event generated by the cloudlet submit function is handled by the MapReduceEngine entity. If the cloudlet is a reduce task then all the map tasks of the corresponding job has finished execution then reduce task is started execution. A check is done to determine pending cloudlets. If no pending cloudlets, then an event is 2finalizeCloudlet() Add cloudlet to ClouletFinishedList Update usedpes CLOUDLET_RETURN event Process clouletReturn(ev) Retrieve task from ev Add cloudlet to CloudletReceivedList If cloudlet is a map task and all map tasks are finished executing then startReducePhase(req) If ClouletList.size=0 and CloudletSubmitted.size=0 Print all cloudlets execution finished clear data centersifCloudList.size>0 and CloudletSubmit.size<=0 createVMsInDataCenter generated to clear the datacenter. Otherwise if there are more cloudlets to be submitted, an event is generated to create more VMs. The CloudSimEx is being extended to include a new dynamic loadaware scheduler to make scheduling decisions dynamic and loadaware. The new class LoadAwareScheduler is included in the MapReduceEngine module of the CloudSimEx to implement the proposed work.

6.2. CONTRIBUTIONS TO CLOUDSIMEX

To implement the proposed algorithm an array queue_capacity is defined to specify the capacity guarantees for each queue defined in the cluster Considering queue_capacity=[70,30] yes No 2If n=50 machines then number of machines for first queue is 70% of 50 machines i.e 35 machines and second queue is 30% of 50 machines i.e 15 machines. To simulate the proposed capacity scheduler of Hadoop two lists were added to CloudSimEx namely job_index, job_submitted_count. CloudSimEx simulator maintains all the cloudlets submitted as a single list (tasks of different jobs) named as CloudletList. In order to avoid changing the entire architecture of CloudSimEx a separate list called as Job_index is created. Job_index list is to indicate the starting index of the cloudlets of the jobs. This list includes the index of the cloudletlist for each job indicating the starting cloudlet i.e. Job_index[i] denotes the index of the starting cloudlet of job i+1 in the cloudletlist. Job_index[i+1] denotes the index of the starting cloudlet of job 'i+2' in the cloudletlist. Hence job_index[i+1]-job_index[i] denotes the number of cloudlets corresponding to the job i if i>0. If i=0 then job_index[0] denotes the number cloudlets of the job (first job in the list). This list is created to enable the proposed scheduler to skip checking with every cloudlet of the same job in the cloudletlist. If the task tag of the job does not match the VM tag, then the remaining tasks of the same job need not be checked for scheduling decisions. The scheduler can directly move to the cloudlet of the next job to check for schedulability. If ith job tasks are not schedulable on the VM then job_index [i] can be used directly to move to the cloudlets of i+1 th job in the cloudletlist. Job_submitted_count[] is used hold the count of cloudlets submitted under each queue. Ex. Job_submitted_count[i] indicates the number of cloudlets submitted to virtual machines for queue i. This array is defined to ensure the capacity guarantees of the queues as done in capacity scheduler of Hadoop. When a cloudlet is to be scheduled jo


 PRINCIPAL

Geethanjali College of Engineering and Technology
 (Autonomous)
 Cheerjal (V), Keesera (M), Madhurai Dist. (T.S.) - 591 30

$b_submitted_count[i]$ is referred to find whether the queue i can be selected for the next allocation depending on whether the capacity provided is less than the capacity guaranteed. If initial capacity of data center = 10 machines with above example as given in equation 1. $Job_submitted_count=[0,0]$ (number of cloudlets yet scheduled from the queue) $Queue_capacity=[70,30]$ Where queue1 is guaranteed a capacity of seven machines and queue2 is guaranteed a capacity of three machines. Considering the set of 2 jobs submitted to both the queues then a cloudletlist would be created for all the number of tasks for both the jobs in the queue. Considering Job1: MapReduce 9_2 :- 9 map tasks and 2 reduce tasks Job 2: MapReduce 15_2:-15 map tasks 2 reduce tasks $Job_index[11,28]$ $Job_index[0]=11$, indicates the starting cloudlet of the next job. $Job_index[1]=28$ indicates location of next job in queue. The cloudletlist is upto 27. Number of cloudlets of job1=11-0=11 Number of cloudlets of job2=28-11=17 2Considering the map task of first job is scheduled then the job_index is modified as $job_index=[10,27]$ as the cloudlet would be removed from the cloudletlist. The job_index is modified as different cloudlets are getting scheduled. In our proposed algorithm load aware scheduler the tag associated with the task and the VM tag is used in scheduling decisions as specified in algorithm no __MapReduceEngine is modified to include the scheduling decisions based on the job tag and VM tag. The job tag is setted in the $cloudlet_return()$ method from the task tag by using the resource usage characteristics of the task.

6.3. PROPOSED WORK SIMULATION .

The following changes are being contributed to CloudSimEx for the implementation of the proposed scheduling algorithm 1. Cloud.yaml which supports a hybrid model is changed to include only public data centers 2. Ability to specify number of machine instances as per the user request instead of all machines available in datacenter 3. Allocation of VMtype based on user request 4. To include locality info for the tasks input data as in HDFS 5. Ability for dynamic binding of cloudlet to VM 6. Ability to specify no. of machines and machine types through simulation.properties

26.3.1. Simulation of Load Aware FIFO Scheduler for Map Reduce frameworks

Dynamic Load aware scheduler is designed to submit the jobs in consideration with the job characteristics and VM characteristics where the submit cloudlets is modified to schedule the cloudlets to VM as per their characteristics. Algorithm 1 LoadAwareScheduler ()

1. n = machines specified from request in simulation.properties
2. create n Virtual Machines in the selected datacenter where the $mtype$ is available
3. while $cloudletList.size()>0$ do
 - 3.1 submitCloudlet() //based on Vmtag and job tag: Algorithm2
 - 3.2 for each vm : $vmList$
 - 3.2.1 process the cloudlet submitted to VM
 - 3.2.2 update cloudlet processing
 - 3.2.3 check cloudlet completion
 - 3.2.4 cloudletReturn() //Algorithm3
 - 3.2.5 for each finished cloudlet add new one from the waiting list
 - 3.2.6 update cloudlet execution time, finish time
 - 3.3 end for
4. end while
5. close datacenters
6. print job execution statistics

2SubmitCloudlet() algorithm identifies an appropriate task based on resource usage characteristics for each VM and assigns the task to it. Algorithm2 submitCloudlet()

1. if $cloudletList.size()=0$ exit;
2. for each vm in $VmList$
 - 2.1 for each cloudlet in $cloudletList$
 - 2.1.1 if($cloudlet instanceof ReduceTask$)
 - 2.1.1.1 if($allMapTasksFinished(cloudlet)$)
 - 2.1.1.1.1 Send cloudlet to VM;
 - 2.1.1.1.2 break;
 - 2.1.1.1.2 elsecontinue;
 - 2.1.1.1.3 end if
 - 2.1.1.2 end if
 - 2.1.1.3 if($cloudlet instanceof MapTask$)
 - 2.1.1.3.1 $id=cloudlet.getId()$;
 - 2.1.1.3.2 Request $r=getRequestFromTaskId(id)$;
 - 2.1.1.3.3 if($r.job.tag AND vm.tag =0$)
 - 2.1.1.3.3.1 $C=cloudlet$;
 - 2.1.1.3.3.2 $done=true$;
 - 2.1.1.3.3.3 break;
 - 2.1.1.3.3.4 end if
 - 2.1.1.3.4 end if
 - 2.1.1.4 end if
 - 2.2 end for
 - 2.3 if($!done \& CloudletList.size() !=0$)
 - 2.3.1 $C=cloudletList.get(0)$;
 - 2.3.2 end if
 - 2.3.3 $C.setVmId(Vm.getId())$;
 - 2.3.4 bind cloudlet C to VM;
 - 2.3.5 add cloudlet C to $cloudletSubmittedList$;
 - 2.3.6 remove cloudlet C from $cloudletList$;
 - 2.3.7 send event CLOUDLET_SUBMIT to datacenter
 - 2.4 end if
3. end for

cloudletReturn() algorithm is being implemented to analyze the characteristic of

Job so that it can be used in scheduling decisions of remaining map tasks of the job. Algorithm3 cloudletReturn()

1. id=cloudlet.getId();
2. Request r=getRequestFromTaskId(id);
3. if(cloudlet instanceof MapTask)
4. CPU=CPUUtilizationTime();
5. TET=t.getTotalTimeOfExecution();
6. CPUR=CPU/TET;
7. if CPUR >0.5 then
- 7.1 j.setTag(CPU_HEAVY);
28. else if CPUR <0.5 then
- 8.1 j.setTag(IO_HEAVY);
9. else
- 9.1 j.setTag(NORM);
10. endif
11. endif

6.3.2. Simulation of Load Aware Capacity Scheduler for Map Reduce frameworks CloudSimEx

is being contributed to simulate capacity scheduler for map reduce tasks in Hadoop. The following changes are contributed to CloudSimEx tool to model capacity scheduler and the proposed algorithm.

1. Number of virtual machines required by user is specified as parameter using cloud.yaml file as the cloud is rented by specifying the number and type of machines.
2. Binding of cloudlets to VM's is modified to be done incrementally as number of VMs is limited based on user request. Cloudlets are submitted to VMs based on the capacity guaranteed. (Algorithm1)
3. Job submission process is modified to include queues with capacity guarantee (queue_capacity variable). Queue_capacity specifies the percentage of resources guaranteed to queues. Cloudlets are remodeled to assign to n VMs as per the queue_capacity specified by user.
4. Submit cloudlet method is being overridden to submit cloudlet to a specified VM which enables assigning a task to VM that is ready to run a new task in consideration with capacity guarantees. (Algorithm2)
5. Cloudlet return as given in Algorithm 3 of previous section 2.2.1 is being modified to invoke submitCloudlet(VM vm) to submit a cloudlet to the specified VM.
6. Job_submitted_count[] array is used to list the number of cloudlets submitted for each queue to implement the capacity allocation constraints.
7. job_index[] array is used to hold the index of each job submitted to cloudletList. submitCloudlets() is invoked only once during the initial scheduling of cloudlets to VMs. Once the first cloudlet is submitted, job_submitted_count is updated to reflect the current allocation done to the queue to which the job belongs. If the allocated resource (VM) count for that request is less than the guaranteed queue capacity the next index also belongs to the same job otherwise the index is made to point to next job request to satisfy its guaranteed queue capacity.

Algorithm1 submitCloudlets()

- 1 q_c = Queue_Capacity;
- 2 count,index,q=0; lastindex = job_index[q]-1;
- 3 for each vm in VmsCreatedList
- 3.1 cloudlet = get cloudlet at index
- 3.2 if (cloudlet instanceof ReduceTask && !isAllMapTaskFinished(cloudlet.getCloudletId())) then continue;
- 3.3 r = request to which cloudlet belongs
- 3.4 set vmid of cloudlet to vm
- 3.5 put(cloudletid,vmid) into scheduling plan
- 3.6 send CLOUDLETSSUBMIT event;
- 3.7 cloudletsSubmitted++; count++;
- 3.8 job_submitted_count[q] = count;
- 3.9 for i = q to job_index.length-1
- 3.9.1 job_index[i] = job_index[i]-1;
- 3.10 lastindex = job_index[q];
- 3.11 if (!((count < q_c[q]) & (index < lastindex))) then
- 3.11.1 index = lastindex
- 3.12 count = 0; q++;
- 3.13 lastindex = job_index[q];
- 3.14 endif
- 3.15 add cloudlet to CloudletSubmittedList
- 3.16 remove cloudlet from CloudletList
- 4 end for

SubmitCloudlets(intVmid) is invoked to submit cloudlets when a VM is ready to accept a new task and finished the previously allocated task. Index is made to point to the queue which has less allocated resources than the guaranteed capacity. If the task is a reduce task and its corresponding map tasks are not completed then cloudlet from next queue is considered. If the queue is last queue tried and no more cloudlets are there next to it in the cloudletlist we simply return, otherwise point to the next queue. But if the task is a map task and within the capacity guaranteed for the corresponding queue then that task is submitted to the VM.

Algorithm2 submitCloudlets(intvmid)

- 1 if (getCloudletList().size() > 0) then
- 2 done = false;
- 3 index = 0;
- 4 for each vm in VmsCreatedList
- 4.1 if (vm.getId() == vmid) then
- 4.1.1 q = 0;
- 4.1.2 if ((job_submitted_count[q] < q_c[q]) & (job_index[q] > 0)) then break;
- 4.1.3 else q++;
- 4.1.


PRINCIPAL
 Geethanjali College of Engineering and Technology
 (Autonomous)
 Keesara (M), Medak Dist. (T.S.) - 501 301

4 break; 4.1.5 endif 4.2 if (q != 0) then vindex = job_index[q - 1]; 4.3 cloudlet = get cloudlet at index 4.4 if (cloudlet instanceof ReduceTask && ! isAllMapTaskFinished(cloudlet.getCloudletId())) then 4.4.1 if (q == q_c.length - 1) index = job_index[q - 1]; 4.4.2 if (index == 0) return; 4.4.3 endif 4.4.4 else 4.4.4.1 index = job_index[q]; q = q + 1; 4.4.5 endif 4.4.6 cloudlet = get cloudlet at index 24.4.7 endif 4.4.8 r = request to which cloudlet belongs 4.4.9 set vmid of cloudlet to vm 4.4.10 put(cloudletid, vmid) into scheduling plan 4.4.11 send CLOUDLET SUBMIT event 4.4.12 cloudletsSubmitted++; 4.4.13 job_submitted_count[q]++; 4.4.14 if (q == job_submitted_count.length - 1) then job_index[q] = 1; 4.4.15 if (q > 0) 4.4.16 if (job_index[q] == job_index[q - 1]) 4.4.16.1 t = job_index[q]; 4.4.16.2 job_index[q] = 0; job_index[q - 1] = 0; 4.4.16.3 for (int i = q + 1; i < job_index.length; i++) 4.4.16.3.1 job_index[i] = job_index[i] - t; 4.4.17 endif 4.4.18 endif 4.4.19 else 4.4.20 for (int i = q; i < job_index.length; i++) 4.4.20.1 job_index[i] = job_index[i] - 1; 4.4.21 endif 4.4.22 getCloudletSubmittedList().add(cloudlet); 4.4.23 getCloudletList().remove(cloudlet); 24.5 endif 4.6 endfor 5 endif When a cloudlet is returned, submit cloudlet decides about which job is to be given to the VM based on the job tag and VM tag within the constraints of queue capacity. Algorithm submitCloudlets() // CapacitySchedulerLoadAware 1. q_c = QueueCapacity; 2. index, q = 0; lastindex = job_index[q] - 1; 3. for each vm in VmsCreatedList 4. cloudlet = first cloudlet in cloudletList; 5.1 done = false; 5.2 for idx = 0 to job_index.length 5.3 q = idx; 5.4 if (idx == 0) then index = idx 5.5 else index = job_index[idx - 1] 5.6 if (job_submitted_count[q] < q_c[q]) then 5.6.1 cloudlet = Cloudlet at index 5.7 if (cloudlet instanceof ReduceTask && ! isAllMapTaskFinished(cloudlet.getCloudletId())) then continue; 5.8 r = request to which cloudlet belongs 5.9 if (((int) r.job.getTag() AND (int) vm.getTag()) == 0) then 25.9.1 set vmid of cloudlet to vm 5.9.2 put(cloudletid, vmid) into scheduling plan 5.9.3 send CLOUDLET SUBMIT event 5.9.4 cloudletsSubmitted++; 5.9.5 done = true; 5.9.6 break; 5.10 endif 5.11 end if 6 end for 7 if (!done) 7.1 cloudlet = first cloudlet in cloudletList; 7.2 r = request to which cloudlet belongs 7.3 q = 0; 7.4 set vmid of cloudlet to vm 7.5 put(cloudletid, vmid) into scheduling plan 7.6 send CLOUDLET SUBMIT event, 7.7 cloudletsSubmitted++; 8 end if 9 job_submitted_count[q] = job_submitted_count[q] + 1; 10 for (int i = q; i < job_index.length; i++) 10.1 job_index[i] = 1; 11 lastindex = job_index[q]; 12 if (!(job_submitted_count[q] < q_c[q]) & (index < lastindex)) then 212.1 index = lastindex; q++; lastindex = job_index[q]; 13 endif 14 add cloudlet to CloudletSubmittedList 15 remove cloudlet from CloudletList 16 end for 2

CHAPTER 7: SIMULATION RESULTS 7.1. Evaluation of Dynamic Load Aware Scheduler for FIFO Simulation. properties: cloud.file=Cloud.yaml experiment.files=test5.yaml machines=<< No of machines used is 4,5,6,7,8 >> mtype=large-aws-us-east-1 jobs used : MapReduce_9_2.yaml(10000millioninstructions) MapReduce_15_2.yaml(100000million instructions) MapReduce_30_2.yaml(100000million instructions) MapReduce_50_1.yaml(10000million instructions) The algorithm is tested with 2 test cases and each test case with number of machines as 4,5,6, 7 and 8. In each test case three jobs are submitted and the total VM processing time and Cost of each VM is studied. Table 3.13 and Table 3.14 provide the execution times of both the schedulers FIFO and FIFO with Loadawareness for Map reduce jobs: 50_1, 15_2, 9_2 and Map reduce jobs: 30_2, 15_2, 50_1. Fig.3.9 gives the comparison of FIFO and Load Aware scheduling for map reduce jobs: 50_1, 15_2, 9_2 and Fig.3.10 gives the comparison for map reduce jobs: 30_2, 15_2, 50_1. The final Execution time of the cluster with n machines to complete both the jobs is considered as the maximum amount of time used by the VMs in the cluster and similarly the cost. 2

CASE 1: test5.yaml !!org.cloudbus.cloudsim.ex.mapreduce.Experiment workloads: - !!org.cloudbus.cloudsim.ex.mapreduce.Workload [LoadAware, Public, { GOLD: 100.0, SILV

ER: 60.0, BRONZE: 0.0 }, [#[Submission Time, Deadline, Budget, Job, user class] !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_50_1.yaml, GOLD], !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_15_2.yaml, GOLD], !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_9_2.yaml, GOLD]]]

Table 3.13: Execution times for Map Reduce jobs: 50_1, 15_2, 9_2

No of machines	Algorithm	Job1 finishing time (sec)	Job2 finishing time(sec)	Job3 finishing time(sec)	Total Cloud Execution time(sec)
24	Load Aware	2749.12	2692.62	2786.62	2786.62
24	FIFO	227.62	3134.32	3176.02	3176.02
5	Load Aware	2098.27	2037.32	2128.27	2128.27
5	FIFO	205.12	2372.32	2399.38	2399.38
6	Load Aware	2000.12	2007.32	2026.93	2026.93
6	FIFO	177.62	2335.26	2363.76	2363.76
7	Load Aware	1985.12	1880.12	2007.62	2007.62
7	FIFO	165.12	2322.69	2377.69	2377.69
8	Load Aware	1414.69	1352.32	1429.69	1429.69
8	FIFO	152.62	1562.62	1589.38	1589.38

Fig.3.9: Comparison of FIFO and Load Aware scheduling for MR jobs:50_1, 15_2,9_2

CASE 2: test5.yaml !!org.cloudbus.cloudsim.ex.mapreduce.Experiment workloads: - !!org.cloudbus.cloudsim.ex.mapreduce.Workload [LoadAware, Public, 2{ GOLD: 100.0, SILVER: 60.0, BRONZE: 0.0 }, [#[Submission Time, Deadline, Budget, Job, user class] !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_30_2.yaml, GOLD] !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_15_2.yaml, GOLD], !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_50_1.yaml, GOLD]]]

Table 3.14: Execution times for Map reduce jobs: 30_2, 15_2, 50_1

No of machines	Algorithm	Job1 Finishing Time (sec)	Job2 Finishing time(Sec)	Job3Finishing time(Sec)	Total Cloud Execution time(Sec)
4	Load Aware	2542.32	2542	2602.03	2602.03
4	FIFO	42.62	3016.97	3075.01	3075.01
5	Load Aware	1910.06	1915.83	1960.54	1960.54
5	FIFO	31.61	2281.02	2325.74	2325.74
6	Load Aware	1902.32	1902.32	1939.69	1939.69
6	FIFO	30.11	2256.24	2293.89	2293.89
7	Load Aware	1895.12	1880.12	1932.62	1932.62
7	FIFO	30.09	2257.73	2287.73	2287.73
8	Load Aware	1277.32	1277.32	1307.01	1307.01
8	FIFO	30.01	1510.48	1538.51	1538.51

Fig.3.10: Comparison of FIFO and Load Aware scheduling for MR jobs: 30_2, 15_2, 50_1

The evaluation of the proposed Dynamic Load Aware scheduling algorithm indicates an improvement of 14% to 17% in the makespan of the jobs submitted to Hadoop cluster on cloud. It is observed that though finish time of job1 is less in FIFO, the load aware algorithm schedules the task in such a way that the makespan is minimized so that the rent paid for cloud usage would be less. This algorithm is more advantages when large Hadoop jobs are submitted to Hadoop cluster on cloud. It would be only helpful in the situations where makespan of jobs is important so that cloud usage cost is minimized.

7.2. EVALUATION OF DYNAMIC LOADAWARE SCHEDULER FOR CAPACITY SCHEDULER

Simulation.properties: cloud.file=Cloud.yaml experiment.files=test3.yaml machines=<< 4,5,6,7,8 >> 2mtype=large-aws-us-east-1 Map Reduce jobs used : MapReduce_9_2.yaml, MapReduce_15_2.yaml, MapReduce_30_2.yaml, MapReduce_50_1.yaml, MapReduce_100_3.yaml Experiment:test3.yaml !!org.cloudbus.cloudsim.ex.mapreduce.Experiment workloads: - !!org.cloudbus.cloudsim.ex.mapreduce.Workload [CapacitySchedulerLoadAware, Public, { GOLD: 100.0, SILVER: 60.0, BRONZE: 0.0 }, [#[Submission Time, Deadline, Budget, Job, user class] !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_50_1.yaml, GOLD], !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Request [200000, 120, 2.5, MapReduce_100_3.yaml, GOLD], !!org.cloudbus.cloudsim.ex.mapreduce.models.request.Reques [200000, 120, 2.5, MapReduce_15_2.yaml, GOLD],]]

Table 3.15 indicates the execution times of CS and CSLA in CloudSimEx for four differe

nt test cases and Fig.3.11 to Fig.3.15 gives the comparison of the performance of CS and CSLA scheduler for these test cases. Table 3.15: Execution times of CS and CSLA in cloudSimEx Test Case nVMs(N=4) nVMs(N=5) nVMs(N=6) nVMs(N=7) CS CSLA CS CSLA CS CSLA CS CSLA MR-30-2,MR-9-2 101.88 87.05 81.78 68.47 68.52 58.272 59.094 57.67 MR-50-1,MR-30- 2 313.57 240.57 255.82 187.52 224.89 157.9 201.369 137.49 MR-50-1,MR- 100-3 504.17 444.42 398.1 353.18 316.19 192.6 212.959 198 MR-100-3,MR- 70-2 530.75 488.97 439.21 347.13 371.79 319.91 325.07 305.14 N=4 N=5 N=6 N=7 0 20 40 60 80 100 120 MR-30- 2,MR-9-2 CS MR-30- 2,MR-9-2 CSLA Number of virtual machines Ex ec uti on ti m e N=4 N=5 N=6 N=7 0 50 100 150 20 0 250 300 350 MR-50- 1,MR-30-2 CS MR-50- 1,MR-30-2 CSLA Number of virtual machines Ex ec uti on ti m e Fig.3.11: Comparison of CS and CSLA Fig.3.12: Comparison of CS and CSLA for test case MR 50-1, MR 30-2 for test case MR 30-2, MR9-2 2N=4 N=5 N=6 N=7 0 100 200 300 400 500 600 MR-50- 1,MR-100-3 CS MR-50- 1,MR-100-3 CSLA Number of virtual machines Ex ec uti on ti m e N=4 N=5 N=6 N=7 0 100 200 30 0 400 500 600 MR-100- 3,MR-70-2 CS MR-100- 3,MR-70-2 CSLA Number of virtual machines Ex ec uti on ti m e Fig.3.13: Comparison of CS and CSLA Fig.3.14: Comparison of CS and CSLA for test case MR50-1, MR 100-3 for test case MR 100-3, MR70-2 0 100 200 300 400 500 600 N=4 N=5 N=6 N=7 Map Reduce jobs Ex e c uti on ti m e Fig.3.15: Simulation results of Capacity Scheduler and Capacity Scheduler Load Aware The above results Fig.3.11 to Fig.3.15 indicate an improvement in execution time of the map reduce jobs on an average of 20%in the makespan of the jobs submitted to Hadoop cluster on cloud. The dynamic load aware algorithm schedules the task in such a way that the makespan is minimized so that the rent paid for cloud usage would be less. This algorithm is more advantages when large Hadoop jobs are submitted to Hadoop cluster 2on cloud .It would be more helpful in the situations where makespan of jobs is important so that cloud usage cost is minimized.

2CHAPTER 8 :CONCLUSION AND FUTURE SCOPE 8.1 CONCLUSION A dynamic load aware scheduler for Hadoop framework is designed and implemented using Hadoop 2.5.2 version as base code. The results indicate an improvement of 12% to 23% in execution time when scheduling is done using proposed dynamic load aware scheduler compared to default capacity scheduler of Hadoop. The proposed work is also simulated using CloudSimEx simulator and results indicate an improvement of 20% in execution time for batch of jobs.The execution time is reduced as scheduling is done using node resource availability and task resource usage characteristics. 8.2 FUTURE SCOPE As the proposed work has implementation to determine the job resource utilizations, it can be used to determine the number and type of virtual machines required to get optimized execution time for future execution of the same job on cloud. The proposed work can also be further extended for automatic configuration parameter setting for a job based on job characteristics to achieve optimized

END

PRINCIPAL
 Geethanjali College of Engineering and Technology
 (Autonomous)
 K. J. Somaiya Road, K. J. Somaiya (M), Marathahalli Dist. (T.S.) - 501 301