A PROJECT REPORT

on

"COMPARATIVE ANALYSIS OF TASK CONFIGURATION SCHEDULING ALGORITHMS IN HADOOP"

COEN 279

BY

Aditya Randive 1409938
Caleb Fujimori 1425405
Saurabh Somani 1420248

ABSTRACT

Hadoop is a java open-source framework for distributed storage and processing on computer clusters with the very large data set. Hadoop core consists of two parts: 1) Hadoop Distributed File System (HDFS) which is the storage part. 2) MapReduce as a processing part. Files in Hadoop is split to smaller pieces (called blocks/chunks) its size is 64MB or 128MB, which are distributed over nodes in the cluster. Since files are stored in a distributed manner and processed parallely, scheduling of tasks becomes a critical issue for the overall performance of the system. There are many factors that determine the scheduling paradigm in Hadoop. In this project we provide a critical analysis of the 3 important factors and 4 different scheduling algorithms. We suggest an improvement strategy for the 4 different scheduling algorithms. We also provide a comparison between 4 different types of schedulers in Hadoop and discuss the future scope of the project in brief.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to Prof. Farokh Eskafi for providing us with an opportunity to explore our interests in Operating System and HDFS. Without his tremendous support, encouragement as well as valuable inputs, this project couldn't have materialized. The support received from all the members who contributed to this project was vital for our success.

INDEX

Section 3] CALEB FUJIMORI

1] Crucial factors for Scheduling of Tasks

1.1 Locality :

Locality refers to the locality of the input data. When the input data is closer to the node on which computation takes place, it causes a lower cost for data transfer. Locality is a very important issue affecting the performance in a shared cluster atmosphere because the network bisection bandwidth is limited. Actually, high locality helps in enhancing the throughput of tasks. The most efficient case occurs when processing takes place on the same node where the data is stored but generally it is very difficult to get such a scenario because the requirements of jobs are different at different times which may not be satisfied by a node's resources. Hence, it is preferred to execute a task on the same rack.

1.2 Fairness:

When we talk about large data, we know that the number of jobs received at an instant is very large. Hence there has to be a way to ensure that the job which arrives first is executed first. For example, we have loaded a node with many tasks and it is executing a heavy workload job now, in that case all the waiting jobs will have to wait which is unfair. The demand of the workload can be elastic, so a load to each job-sharing group should be considered.

1.3 Workload:

Every Task Tracker has a limit as to how many jobs can be executed at a time on it (we call them slots). The number of slots is 2 by default and we can set it as per our case. When Job-Tracker assigns job to Task Tracker, it follows FCFS (First Come First Serve) strategy and so it creates possibility that a node may get overloaded whereas another is idle. Hence there has to be an efficient way to do this assignment of jobs.

1.4 Data Locality Aware Scheduling Algorithm

- In application requiring large data processing, the input given to a map task can be expected to be much larger in size than the codes involved in the task performing.
- To reduce the cost involved in transmission of data, the task is assigned in such a way that input data is closest to the node that requests the task.
- Two things are considered while scheduling -> 1) Waiting Time 2) Transmission Time

- If the transmission time is longer than the waiting time, the method reserves the task for the node that possess the input data so as to take advantage of data locality
- Else it the task is scheduled on a node which is in a different rack.
- Disadvantage of this algorithm is that it doesn't fully utilize the resources. So, eventually one node will be heavily loaded while others remain idle. This approach sacrifices throughput in order to achieve speedy scheduling of tasks. Also, this approach doesn't take into considerations the diversity of applications.

1.5 Workload Characteristic Oriented Scheduling Algorithm

- Applications running on the same platform can have two types of characteristics which are CPU intensiveness and I/O intensiveness. Such kind of diversity is ignored in Hadoop as FCFS scheduling approach is used by the current Hadoop scheduler.
- Tasks exhibiting the same resource utilization are more probable to being dispatched to the same machine. Resource contention can occur in such approach which also leads to reduction in throughput.
- WCO is different from other schedulers as it considers the diversity of applications whereas the earlier works consider only one type of MapReduce applications.
- WCO can detect the workload type by using a prediction mechanism which works on the basis of I/O rate. It also uses a strategy in which it combines workload of different types and executes them on the same machine. This improves the usage efficiency of CPU as well as I/O resource.
- Computing rate ratio(CRR) has been used to characterize the task as CPU or I/O intensive. If the CRR approaches 1 then it is CPU intensive and if it approaches to 0 then it is I/O intensive.
- For estimation of workload characterization and estimation process is used which consists of two phases: 1) Sampling 2) Adjustment.
- Sampling consists of two steps 1) Task selection 2) Generalization of workload characteristic.
- In WCO, task selection is done in a dynamic way. A job can be submitted at any time and concurrent execution of tasks takes place through sharing of resources of a node by slots. Larger is the number of tasks, lesser is the overhead. Random and static analysis strategies are used in task selection.
- In the Second Phase, i.e. Generalization of workload characteristic, characteristic of a task indicates tendency of resource utilization of a task. All map tasks handle the same sized input data in map phase and the user defined map function for processing the workload is the same. User

defined function for every reduce task is also the same as the workload patterns of map task have uniform distribution. The input to each reduce task obtained from the map task is approximate.

- Adjustment is basically dispatch of remaining tasks are made to execute with other jobs which were found to have comparable workload characteristics.

- The disadvantage of WCO scheduler does not take into consideration the historical data from prior execution of the same job. Disadvantage is that the characterization of jobs on the basis of the computing rate may increase the waiting time of a job which has a moderate computing rate. But the advantage is that it increases throughput & performance of the system.

1.6 Improvement suggested: Tabular Approach

- The improvement consists of two phases: 1) Building a lookup table on top of WCO scheduling algorithm 2) Using Data Locality Aware algorithm to select the node from the table.

- Lookup table on top of WCO scheduling algorithm indicates "Number of Free Slots" per "Node". Through this table we eliminate the disadvantage of WCO algorithm's excessive recalculation of computing rate for historical data. This is because, from the table we have a higher level of abstraction describing the available free slots over all the nodes. Once we know which all slots are available for scheduling, we run data locality algorithm to achieve closeness to the data.

- This improvement creates an overhead which is building the table on the top of WCO and running the Data locality as the second phase. However, this overhead can be ignored as we achieve diversity of applications, removal of redundant calculations, improved data locality. Thus, this approach provides us with Speedy scheduling of tasks with an improvement on the Throughput & Performance of the system.

Method	Description	Strength	Incorporating Proposed Tabular Form
WCO Scheduler	Characterization of jobs into I/O & CPU intensive. 2 phase estimation Sampling & Adjustment	Improvement in resource utilization and application performance Shows 17% performance improvement in terms of throughput	If no node is free, defer from queueing the job
Data Locality Aware	Computes waiting time using remaining execution time Divides tasks in smaller phases to ensure constant processing time	Out performs the native Hadoop Reduce task scheduler by an average of 7% to 11.6% Provides an efficient equation to calculate waiting time	It can be used to decide among the nodes having same number of free slots

Description for Tabular Approach	Strength
Maintains a table for each Job Tracker	Distributes Workload efficiently
Any scheduling of new job begins with lookup in the table	Ensures Fairness as each job is probable o be finished in appropriate time
Table has to be updated every time a job is scheduled or a scheduled job has been completed	Helps improving the efficiency of existing algorithms

1.7 Comparison of different Hadoop Schedulers

We also provide an analysis on comparison for Hadoop schedulers – FIFO, FAIR, CAPACITY & DELAY.

Scheduler	Advantages	Disadvantages	Remarks
FIFO	Easy to be implemented	Bad data locality , small job will wait long time, starvation, no fair resources sharing.	Homogeneous System, Static allocation, non primitive
FAIR	Fair and equal share of resource allocation. Short response time with a small delay due to complex calculations	Complex settings, unbalanced workload because of no job weight	Homogeneous, static allocation
CAPACITY	Unused capacity used by jobs n the queue	More complex in implementation	Homogeneous, static, non-primitive
DELAY	Simplified Scheduling, No affect of complex calculations	Doesn't works in all situation, not optimal sharing of resources	Homogeneous, static allocation

1.8 Future Scope

- The future of scope for this Project can be a critical analysis of the performance of the different Hadoop Schedulers, FIFO, FAIR, CAPACITY & DELAY for different applications such as Sort, Word Count, Supervised & Unsupervised learning algorithms, Data Warehouse Operations on Cloud against a benchmark "CLOUDRANK-D".
- The paper, "Evaluating task scheduling in Hadoop-based cloud systems" which was referred as future scope of investigation for this project suggests a maximum throughput can be achieved by DELAY-FAIR scheduler over FIFO, FAIR, CAPACITY & DELAY schedulers on applications mentioned above.

2] JOB-AWARE SCHEDULING:

2.1 Problem:

The existing scheduling algorithms pertaining to HDFS address any one of the below limitations, not all of them:

*Limited Utilization of computing resources
*Limited applicability towards Heterogeneous cluster
*Random scheduling of non-local map tasks
*Negligence of small jobs in scheduling

2.2 Proposed Solution to the above problem:

The Job-Aware scheduling considers all the above limitations of HDFS scheduling process.

The Job-Aware scheduling algorithm considers the following list of criteria for scheduling non-local tasks. These are also applicable to heterogeneous cluster.

1) Job execution time
2) Earliest Deadline first
3) Workload of the job

The first two criteria ensure decrease in average waiting time.
The third criteria mentioned above ensures efficient utilization of resources.

2.3 Terminologies and Notations:

- MapReduce slots: Total number of map and reduce tasks that can be implemented parallely on a cluster node.
- Locality Marker: Used to ensure slave nodes get an equal opportunity to grasps local map tasks.
- Local Map task: Map task implemented on a slave node containing input data.
- Non-local map task: Map task implemented on a slave node not containing the input data.
- Heartbeat: A signal carrying details regarding storage, unused map and reduce slots from slave nodes to the master node.

We use N nodes = $\{n_1, n_2, .., n_n\}$.
Each cluster consists of multiple slave nodes = $\{SN_1, SN_2, \ldots, SN_{n-1}\}$
Single Master node

2.4 Description of the algorithm:

* The Master Node MN maintains a job queue JQ.
* At the arrival of a new job, it is submitted to JQ.
* A job is divided into numerous map and reduce tasks.
* Every map tasks can further be classified as local map tasks t l and non-local map tasks t nl.
* Whenever, a new node is added to i th slave node SN i has a free slot count sc n . It sends a heartbeat to MN.
* The scheduler in MN is responsible for scheduling jobs to SN i.
* Scheduler checks for unassigned local map tasks t_l of the

2.5 Algorithm Pseudo-code

```
Input: Job Queue, N slave nodes
Output: Result of completed jobs
//scheduling non-local map task based on job awareness
1.  FOR i = 1 to N
2.          initialize NL i = NULL
3.  FOR i = 1 to N
4.          MN receives heartbeat from SN i :
5.  WHILE sc i > 0
6.          initialize flag = NL i
7.  FOR k = 1 to L JQ
8.          IF J k contains t l then
9.                  allocate t l of J k to SN i
10.                 sc i = sc i - 1
11.                 IF NL i equals to NULL then
12.                         initialize NL i = 1
13.                 ELSE
14.                         NL i = NL i + 1
15. IF flag equals to NL i then
16.         initialize NL i = 0
17. ELSEIF NL i equals to 0 then
18.         IF (jobs are interactive)
19.                 min_task(JQ, SN i )
20.         ELSEIF (jobs have a strict deadline)
21.                 deadline_aware(JQ, SN i )
22.         ELSEIF (high load on cluster)
23.                 workload_aware(JQ, SN i )
24. sc i = sc i – 1
```

2.6 Performance analysis and Results

The performance analysis of the proposed algorithm is evaluated using the following parameters:

a) Average waiting time.
b) Resource utilization of the cluster

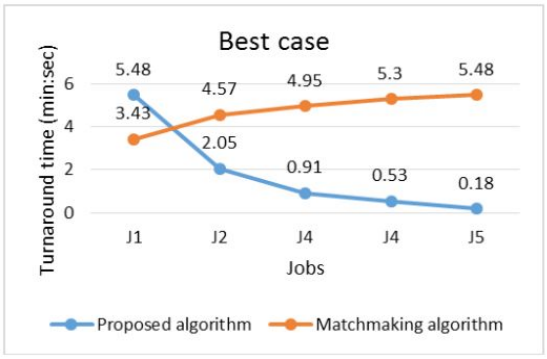
The analysis is carried out with respect to the implementation of the MapReduce WordCount job.
5 such jobs were submitted to the Job Queue simultaneously. The three criteria described earlier are used in the method to implement the proposed algorithm. The performance of the algorithm is compared with the Matchmaking algorithm using the Best Case, the Worst Case as well as the Average case.

- Best Case: Job consists of all non-local map tasks.
- Worst Case: Job consists of all local map tasks.
- Average Case: Job consists of several non-local and local map tasks.

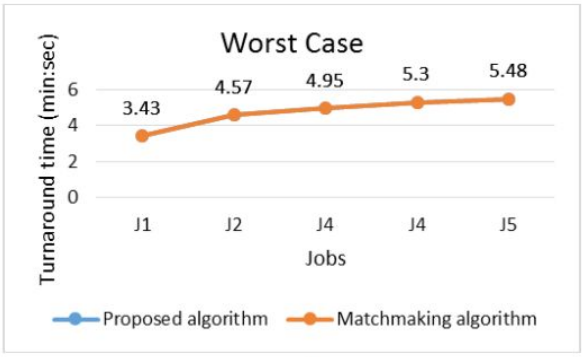
The execution time of the jobs submitted are:

Job id J1: 3 minutes 43 seconds
Job id J2: 1 minute 14 seconds
Job id J3: 38 seconds
Job id J4: 35 seconds
Job id J5: 18 seconds

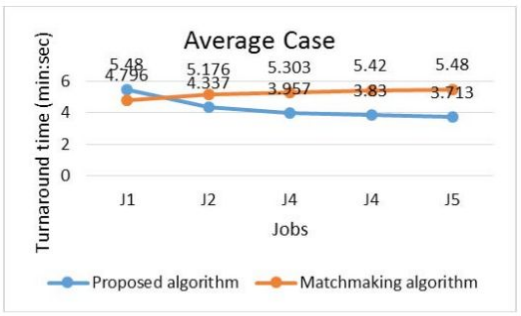
The results show the proposed algorithm **reduces** the average waiting time by around **70%** in **best case** and **20%** in **average case** if the jobs are scheduled using based on job execution time.



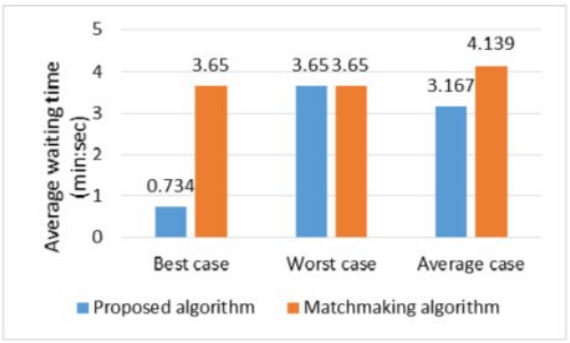
(a)



(b)



(c)



2.7 Scope for improvements: Scheduling using Heuristics and Machine Learning - A Mathematical Analysis:

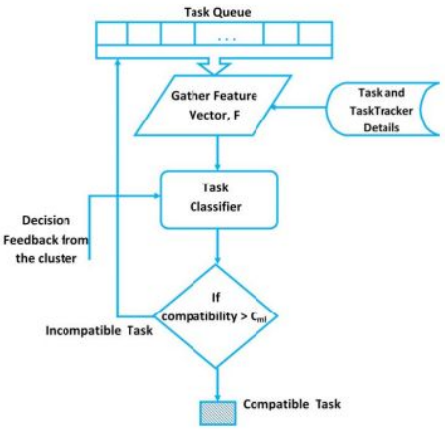
In the below analysis, we suggest delegating the scheduling decisions to a Heuristics based system for a more refined and informed approach:

a) Below, the task vectors are computed to be used while applying the new approach:

CPU-intensive, Memory-intensive, Disk-intensive, Network-intensive

$$\vec{T}_k = E_{cpu}e_1 + E_{mem}e_2 + E_{disk}e_3 + E_{nw}e_4$$

$$\vec{T}_{compound}(i) = \vec{T}_1 + \vec{T}_2 + .. + \vec{T}_n$$



b) In the below computation, the probability of the task being compatible with the incoming data is deduced. Also, the lower the value of *'arccos'*, the better the compatibility of the task.

Incremental Naïve-Bayes

Interfaces:

- *LISP*
- *PROLOG*

$$P(task_k = compatible \mid F) = \frac{P(F|task_k = compatible) \times P(task_k = compatible)}{P(F)}$$

The quantity $P(F|task_k = compatible)$ thus becomes:

$$P(F|task_k = compatible) = P(f_1, f_2, ..f_4|task_k = compatible)$$

where $f_1, f_2, ..f_4$ are the features of the classifier ($\{\Phi, \Sigma, \overrightarrow{T}_k, \overrightarrow{T}_{compound}(i)\}$). We assume that all the features are independent of each other (Naive-Bayes assumption). Thus,

$$P(F|task_k = compatible) = \prod_{j=1}^{4} P(f_j|task_k = compatible)$$

$$angle = \arccos\left(\frac{\overrightarrow{T}_k \cdot \overrightarrow{T}_{availability}(i)}{|\overrightarrow{T}_k||\overrightarrow{T}_{availability}(i)|}\right)$$

3] Deadline Aware Scheduling Algorithms

As Distributed Computing is now very common, having reliable real-time scheduling is important if the job needs to happen in real-time or if completing a job by a deadline is part of a Service Level Agreement. The goal of any deadline-sensitive scheduler is to maximize the Probability of Completion before Deadline (PoCD) while balancing the cost of execution. This section examines two existing YARN task scheduling algorithms, proposes improvements and analyzes the expected performance improvements.

3.1 Outliers

The tasks in a job can have a wide distribution of execution time mainly due to unequal data sizes, network congestion and hardware failure. These outliers contribute heavily to job runtime, thereby decreasing the PoCD.

3.2 Speculative Resume

The two strategies used to reduce the time to completion of these outlier tasks are launching duplicate attempts of the task and rescheduling the task to a different machine. Launching duplicate attempts causes the task to execute on a different machine, so if the extended execution time is due to machine or placement failure, the execution time of the duplicate should be lower. However in these cases, a duplicate may not be necessary and simply rescheduling the task to a different machine may suffice.

3.3 SRQuant

SRQuant is a Speculative Resume scheduler based on the following parameters:

D	Deadline
r	# of speculative attempts to launch
r_{max}	Maximum acceptable r
t_{est}	Time to decide r
t_{kill}	Time to decide which attempt finishes
ϕ_{est}	Progress of task at t_{est}
β	Task time distribution exponent
t_{min}	Minimum execution time of task
N	Number of tasks in phase
θ	Tradeoff factor between cost and PoCD
C	Cost of Computation per time
R_{min}	Minimum acceptable PoCD

SRQuant assumes the attempt execution time follows a pareto distribution $(\frac{t_{min}}{D})^{\beta}$ and attempts to maximize the utility function *U(r)* given by

$U(r) = \log(R(r) - R_{min}) - \theta \cdot C \cdot E_r(T)$

Where *R(r)* is the PoCD, and $E_r(T)$ is the expected execution time of the job.

$$R(r) = [1 - \frac{(1-\varphi_{est})^{\beta(r+1)} t_{min}^{\beta(r+2)}}{D^{\beta}(D-t_{est})^{\beta(r+1)}}]^N$$

$$E_r(T) = N[\frac{t_{min} D\beta(t_{min}^{\beta-1} - D^{\beta-1})}{(1-\beta)(D^{\beta} - t_{min}^{\beta})} \cdot (1 - (\frac{t_{min}}{D})^{\beta}) + (t_{est} + r(t_{kill} - t_{est}) + \frac{t_{min}(1-\varphi_{est})^{\beta(r+1)}}{\beta(r+1)-1} + t_{min}) \cdot (\frac{t_{min}}{D})^{\beta}]$$

SRQuant works by launching one attempt of each task and after that task runs for t_{est}, linearly searching all $r<r_{max}$ to maximize *U(r)*.

SRQuant was shown by Xu et al. to outperform a default Hadoop deadline unaware scheduling algorithm with speculation(Hadoop-S) by up to 31% in terms of PoCD and reduce the cost by up to 13%. Compared to a deadline unaware scheduling algorithm without speculation (Hadoop-NS) SRQuant improved PoCD by up to 89% in terms of PoCD but increases cost by up to 14%

3.4 SRQuant improvements

An analysis by Microsoft of their Bing servers showed that 40% of the outlying tasks could be explained by the abnormally large amount of data these tasks had been assigned, so duplicating these tasks would only increase the cost of execution without increasing the PoCD.
SRQuant could be improved by making t_{min} a function of the size of data being processed by the task instead of setting it constant for all tasks. Assuming the 40% of tasks that will not benefit from extra attempts contributed to 40% of the extra tasks spawned by SRQuant, this could reduce the difference in cost between SRQuant and Hadoop-NS by up to 40%.

Reduce tasks require reading data from all the previous phases Map nodes, so there will be a high amount of network to any rack with a Reduce task. Microsoft found that spreading the initial attempt of Reduce tasks could improve the completion time of a reduce task by up to 34%. Since reduce phases take up 17% of most jobs execution time the PoCD and number of extra attempts launched could reduce by up to 17%.

3.5 LASER

LASER is another speculative resume scheduler very similar to SRQuant but it uses Deep Neural Networks (DNN) instead of a parametric model to estimate execution time of the task and another DNN to determine number of extra attempts. LASER uses the below features to calculate the expected task execution time and *r*.

Feature	Description
$location$	Location host
$taskID$	Identifier of task
φ_j	Task progress score
rec_{read}	Amount of records read
rec_{write}	Amount of records written
$Byte_{read}$	Amount of bytes read from HDFS
$Byte_{write}$	Amount of bytes written to HDFS
t_{cpu}	Amount of CPU consumed time
t_{jvm}	Amount of time for launching JVM
t_{gc}	Amount of time for garbage collection
τ_{est}	The time of estimating the task execution time

LASER was shown to outperform Hadoop-S in terms of PoCD by up to 7% and up to 13% in terms of cost.

3.6 LASER Improvements

LASER could improve by taking into account the same characteristics of outliers presented in the SRQuant improvements. Including the amount of data being processed as a feature in the neural nets and spreading the nodes that reduce tasks are scheduled on should increase PoCD and reduce cost.

4] BIBLIOGRAPHY

1. S. Liu, J. Xu, Z. Liu and X. Liu, "Evaluating task scheduling in hadoop-based cloud systems," *2013 IEEE International Conference on Big Data* , Silicon Valley, CA, 2013, pp. 47-53.
2. S. Pati and M. A. Mehta, "Job aware scheduling in Hadoop for heterogeneous cluster," *2015 IEEE International Advance Computing Conference (IACC)* , Bangalore, 2015, pp. 778-783.
3. A. Maheshwari, A. Bhardwaj and K. Chandrasekaran, "Hadoop Task Scheduling - Improving Algorithms Using Tabular Approach," *2015 Fifth International Conference on Communication Systems and Network Technologies* , Gwalior, 2015, pp. 1034-1038.
4. E. Mohamed and Z. Hong, "Hadoop-MapReduce Job Scheduling Algorithms Survey," *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, Macau, 2016, pp. 237-242.
5. M. Xu, S. Alamro, T. Lan and S. Subramaniam, "LASER: A Deep Learning Approach for Speculative Execution and Replication of Deadline-Critical Jobs in Cloud," *2017 26th International Conference on Computer Communication and Networks (ICCCN)* , Vancouver, BC, 2017, pp. 1-8.