**MassiveDataCruncher**

**S. Arun**

Contents

[1. Introduction 3](#_Toc71172177)

[1.1 Jgroups scheduler 3](#_Toc71172178)

[1.2 Standalone scheduler 4](#_Toc71172179)

[1.3 Ignite Scheduler 4](#_Toc71172180)

[1.4 Apache MESOS scheduler 5](#_Toc71172181)

[1.5 Apache Hadoop YARN scheduler 5](#_Toc71172182)

[2. Partitioning of file blocks 6](#_Toc71172183)

[2.1 Optimization of MDC Job pipeline 6](#_Toc71172184)

[3. Data Pipeline tasks to Stage conversion. 7](#_Toc71172185)

# Introduction

The MassiveDataCruncher is a number crunching scalable server which manipulates data in csv and json format. The MDC is master worker architecture which has scheduler in the master and container as worker. The worker connects to master via JGroups using multicast protocol. The worker launches the task executor whenever a job is submitted. The Heartbeat which uses JGroups will get message from task executors about the job status. The status of the job can be either SUBMITTED, RUNNING, COMPLETED or FAILED. There are multiple schedulers such as jgroups, standalone, apache ignite, apache MESOS and Hadoop YARN.

## 1.1 Jgroups scheduler

The jgroups scheduler works in autonomous fashion submits job as stages and tasks in the order of DAG topological sorting and launches the task executors via containers. The task executors communicate with each other when a stage and tasks depends on parent located in another task executor via jgroups protocol. When a parent task is in running state the child task in another executor will have to wait and proceed only after parent task completes. The parent task can reside in another task executor the waiting task executor can send message via WHOIS command. The running task executor will receive the WHOIS request and send WHOIS response message with the current status of the parent task. The WHOARE command in scheduler will be sent to task executors to monitor all the tasks the task executors are executing. The status from WHOARE response from all the task executor will be considered whether job is completed. Once the final tasks are completed then scheduler will obtain the results via the final tasks.

Jgroups scheduler

Container1

Container2

TE1

TE2

TE3

TE4

In the above diagram the scheduler will determine the number of TEs to launch and communicates to the container to launch 4 Task executors. The number of TEs will decide on the various factors such as total file blocks with common block size, total number of CPUs together with all the containers and memory of each TEs to launch. Each TEs will allocate one CPUs and similar configuration of heap memory. Once all the TEs are allocated and started the scheduler will determine how to launch tasks in each of the TEs.

## 1.2 Standalone scheduler

The standalone scheduler works as jgroups for communication by using heartbeat. The central DAG scheduler will execute the DAG graph in the top to bottom and left to right via DExecutor utility. The DAG scheduler will obtain results of the final stage in DAG which has no successors. The heartbeat in scheduler will receive the status of the tasks from launched task executors and send the status to the scheduler to determine whether the tasks are completed to obtain final results.

Standalone scheduler

Container1

Container2

TE1

TE2

TE3

TE4

## 1.3 Ignite Scheduler

The ignite scheduler submits task to ignite cluster. The job to be submitted via ignite scheduler is converted to the stages and the stages are grouped to tasks and each stages are submitted to ignite cluster via ignite DExecutor utility. The ignite cluster caches the text file in the compressed form using the LZF compressor and then stored in the ignite cache cluster. The tasks are submitted via affinity run method to execute tasks. The cache keys are stored in the ignite server in partitioned form and the tasks are executed on the server where cache key and value is available. The backup is available in another server if the one or more servers are down. The backup configuration is configured in each of the server where cache key and values are partitioned for a cache. The cache can also be replicated in each of the server in ignite cluster.

Ignite Server1

Ignite Server2

Ignite Server3

Standalone scheduler

## 1.4 Apache MESOS scheduler

The apache mesos scheduler submits job to MDC scheduler will be converted to the stages and tasks. The mesos MDC framework first register the scheduler and task executor framework to mesos master and then submits the tasks in the form of DAG to the mesos scheduler. The mesos scheduler executes the DAG and gets the final results once the final tasks been completed. The mesos framework closes the resources allocated after the completion of the jobs submitted. The mesos master manages resources like cpu and memory and provide the offers to the mesos scheduler framework. If the scheduler accepts the resources allocated by mesos as offers, then tasks will be executed to mesos executors. The final stages been completed once all the offers to the tasks are executed by mesos and reaches to the final stage of the DAG.

Mesos

Worker1

Mesos

Worker2

Mesos

Worker3

Mesos Master

Mesos MDC Scheduler

## 1.5 Apache Hadoop YARN scheduler

The job submitted via YARN scheduler contains the MDC scheduler and YARN resource manager (RM) and node manager (NM). The MDC scheduler determines the scheduler type and based on the type the MDC scheduler submits job. The MDC scheduler converts the Job in stages and tasks and stored in HDFS. When the YARN resource manager receives the MDC job the RM will allocate the containers. The number of containers are determined by the total file length and the available cpus and memory. After allocating the containers the application master will be created and started in one of the containers and the job will be provided to app master. The app master will start executing the tasks by the YARN scheduler and frequently monitors the tasks and gets the status of the tasks. Once the final stages been completed the containers will be deallocated and returns to the RM. The resource manager will return the results to the MDC scheduler and job is completed.

Node Manager 1

Node Manager 2

Node Manager 3

YARN RM

MDC Scheduler

# Partitioning of file blocks

The partitioning is performed by MDC pipeline by the user defined block size. If the user defined block size is true then block size and various parameters such as max memory, min memory and cpus are decided by the user else the parameters are automatically defined by the file blocks length. The block size of the file stored in HDFS is 128 MB whereas the user defined block size of any file system can be the range from 1 to 256 MB. If the user defined block size is less than 1 or greater than the 256 MB, then error is thrown. If the file length is 1 GB and the blocks size is 256 MB, then number of partitioning the files will be approximately 4 i.e. (1024/256). The number of partition, total cpus available and the total memory decides the number of parallel executors.

1 GB file

1st

256 MB

2nd

256 MB

3rd

256 MB

4th

256 MB

## 2.1 Optimization of MDC Job pipeline

The data pipeline of MDC job consists of various tasks also called functional interfaces. The tasks in the data pipeline are grouped into stages by combining the several tasks into stages. For example, if the data pipeline consists of Map, filter, MapTuple, ReduceByKey and coalesce and if the number of partitions is 4 then total stages are 2 i.e. Stage1 (Map, filter, MapTuple, ReduceByKey) and Stage2 (coalesce) with parallel stage execution as 4.

Map

Filter

MapTuple

Coalesce

ReduceByKey

Stage1 (Map, filter, MapTuple, ReduceByKey)

Stage2 (Coalesce)

Partition1

Partition2

Partition3

Partition1

# Data Pipeline tasks to Stage conversion.

In the data pipeline before the tasks are executed, the task graph in the order are first converted to stage graph and based on the partitions the logical stages in stage graph are converted to physical execution graph. In the graph below the logical graph is converted to physical execution graph after partition inputs are provided to Stage1 and the output will contain lists of lists. The conversion is common to all the schedulers, only the execution scheduler will vary.

Map

Filter

MapTuple

ReduceByKey

Stage1 (Map, filter, MapTuple, ReduceByKey)

File Block Partition1

File Block Partition2

File Block Partition3

File Block Partition1

Stage1 Partition1

Stage1 Partition2

Stage1 Partition3

Stage1 Partition3

Physical Execution Plan Graph

Logical graph

output