Dynamic Split Model of Resource Utilization in MapReduce

Xiaowei Wang、Jie Zhang、Huaming Liao、Li Zha

Institute Of Computing Technology

Chinese Academy Of Sciences

LOGO

Contents

Introduction

- **MapReduce is gaining increasing popularity as a parallel programming model for large-scale data processing**
- **Traditional MapReduce platforms have a poor performance in terms of cluster resource utilization**
- **Dynamic Split Model of the Resources Utilization**
  - ✓ Dynamic Resource Allocation
  - ✓ Resource Usage Pipeline
- **Optimization verification on top of Hadoop**

Dynamic Scheduling Model

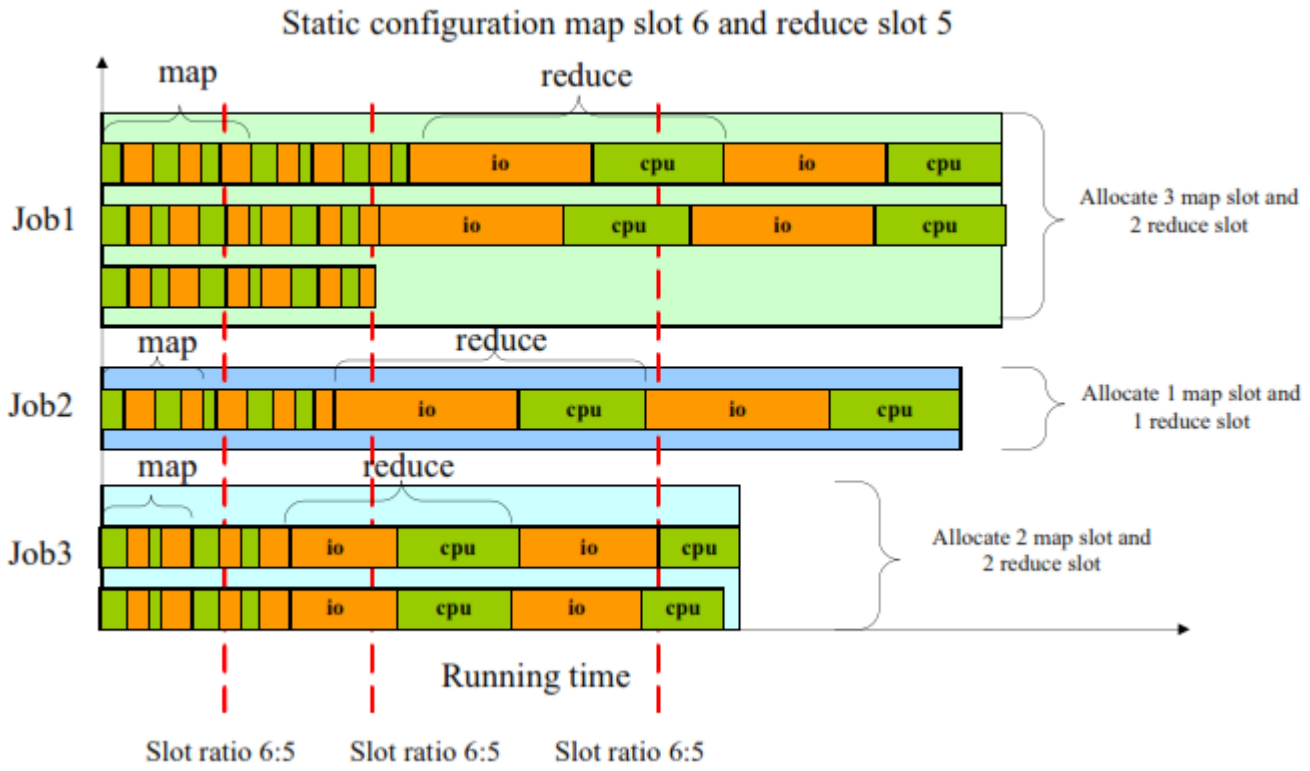


Figure1: Job execution situation on a single node in raw version Hadoop, the red dash dotted line stands for three arbitrary time point in the execution process.

Dynamic Scheduling Model

- **Node resource usage unbalanced:**
  - ✓ Different phases have different resource usage bias at different time
  - ✓ Some resources may be underused while the other overused at the same time
- **Reduce slot hoarding:**
  - ✓ First round reduce tasks will hold reduce slots for a long time if the job has a long time running map
- **Resource allocation unbalance within job:**
  - ✓ A static configuration does not consider the system load and the jobs requirement

Dynamic Scheduling Model

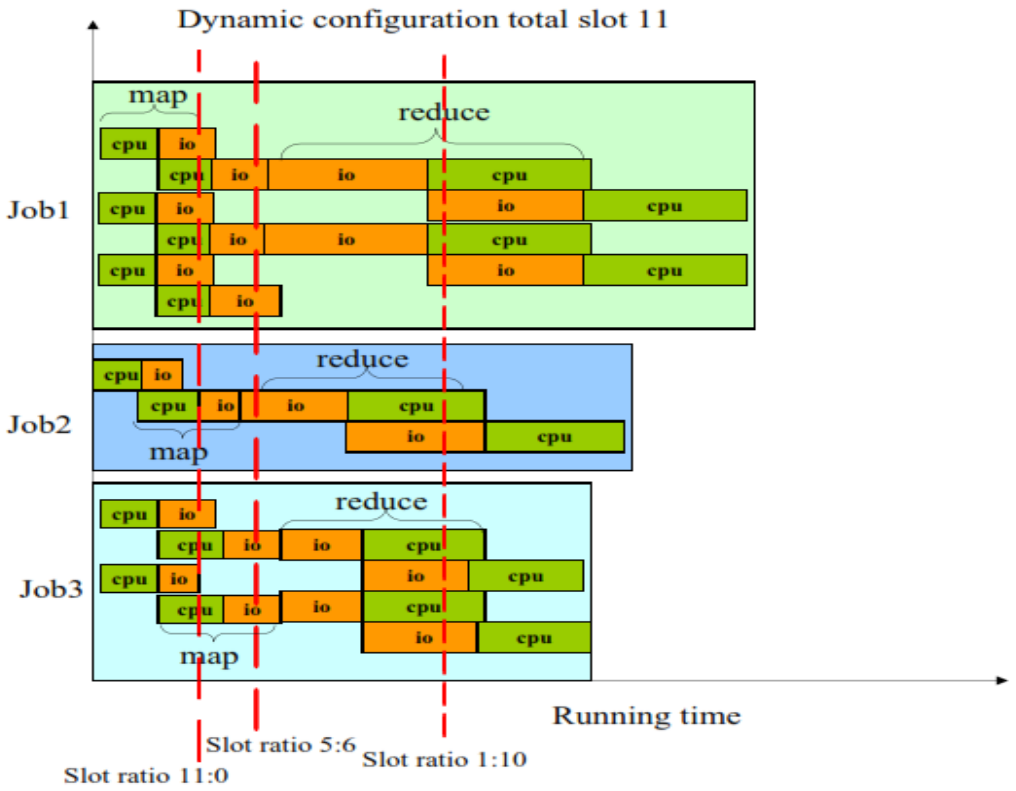


Figure2: Job execution situation on a single node in new version Hadoop the red dash dotted line stands for three arbitrary time point in the execution process.

Dynamic Scheduling Model

- Separate resource usage within a phase into two periods:

  CPU period and IO period. Use advanced scheduling to launch a task at a proper point so that one task's sub-operation can overlapped with the other in case their resource usage is complementary.

- Collect the system load and the status of each job at run time to

  allocate resource dynamically. So that the number of slot is not the same and can be modified according to system load.

Dynamic Resource Allocation

- **Reduce Slot Hoarding Problem**

  The job will hold any reduce slots it receives during this until its maps finish.

- **Resources Allocation unbalance Problem**

  The requirement for slots varies along with job proceeding. Obviously, static slot configuration can’t adapt to these requirements

- **Our Solution: Dynamic Resource Allocation**

  We will allocate resource according to the cluster load and all jobs run-time status.

Dynamic Resource Allocation

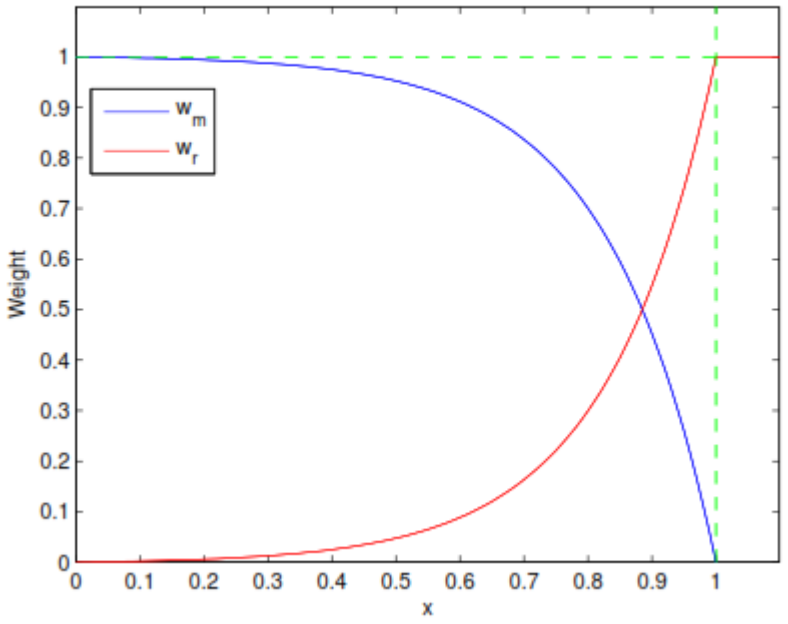


Figure3: The dynamic weight of map phase and reduce phase with the changing of job status.

$W_m + W_r = 1$

✓ w_m : the weight of map phase;

✓ w_r : the weight of reduce phase.

Suppose: the percentage of map phase completion is x.

✓ F_{task} is the number of finished map tasks;

✓ T_{task} is the total number of map tasks in the job.

Then: $x = F_{task}/T_{task}$ $(0 <= x <= 1)$

We defined the w_m and w_r as bellow:

$$w_m = 1 - \frac{1}{e^6 - 1}\left(e^{6x} - 1\right)$$

$$w_r = 1 - w_m = \frac{1}{e^6 - 1}\left(e^{6x} - 1\right)$$

Dynamic Resource Allocation

- **If only one job in the cluster:**
  - ✓ The number of slots in the cluster is *R*;
  - ✓ There are R_m slots use for map phase;
  - ✓ There are R_r slots use for reduce phase.

Then we get:

$$R_m = \frac{w_m R}{w_m + w_r} = w_m R = (1 - \frac{e^{6x} - 1}{e^6 - 1})R$$

$$R_r = \frac{w_r R}{w_m + w_r} = w_r R = \frac{(e^{6x} - 1)}{e^6 - 1} R$$

Dynamic Resource Allocation

- **If lots of users submit jobs:**
  - ✓ There are *n* jobs running in the cluster;
  - ✓ Each job *i* has a weight w_i;
  - ✓ The resource for job *i* is R_i, the map phase gets resource R_{im}, the reduce phase gets resource R_{ir}.

So we can get:

$$R_i = \frac{w_i}{\sum_{i=1}^{n} w_i} R$$

$$R_{im} = \frac{w_{im} R_i}{w_{im} + w_{ir}} = w_{im} R_i = (1 - \frac{e^{6x_i} - 1}{e^6 - 1}) \frac{w_i}{\sum_{j=1}^{n} w_j} R$$

$$R_{ir} = \frac{w_{ir} R_i}{w_{im} + w_{ir}} = w_{ir} R_i = (\frac{e^{6x_i} - 1}{e^6 - 1}) \frac{w_i}{\sum_{j=1}^{n} w_j} R$$

Dynamic Resource Allocation

- **In the cluster:**
  - ✓ There are totally R_M resources allocated for map slots;
  - ✓ There are totally R_R resources allocated for reduce slots;

Then we can get:

$$R_M = \sum_{i=1}^{n} R_{im} = \sum_{i=1}^{n} (1 - \frac{e^{6x_i} - 1}{e^6 - 1}) \frac{w_i}{\sum_{j=1}^{n} w_j} R$$

$$R_R = \sum_{i=1}^{n} R_{ir} = \sum_{i=1}^{n} (\frac{e^{6x_i} - 1}{e^6 - 1}) \frac{w_i}{\sum_{j=1}^{n} w_j} R$$

RESOURCE USAGE PIPELINE

- **Resource Usage Unbalance problem**
  - ✓ io.size.mb configuration conflict in map phase
  - ✓ Obvious resource usage in reduce phase
  - ✓ resource usage unbalance problem in a single node

- **Our Solution: Resource Usage Pipeline**
  - ✓ Dynamic Buffer Enlargement in Map Phase
  - ✓ Dynamic Slot Request of Map Task
  - ✓ Dynamic Slot Request of Reduce Task

RESOURCE USAGE PIPELINE

- **Dynamic Buffer Enlargement Logic in Map Phase**
  - ✓ Map task allocates a kvbuffer according to the default io.sort.mb value to hold output (key, value) pairs.
  - ✓ We used a dynamic buffer instead of the static buffer in raw Hadoop.
  - ✓ The method used to calculate how much free memory **x** is needed.

$$pfs : tfs = abs : x$$

    - ➢ *tfs*: total file size;
    - ➢ *pfs*: processed file size;
    - ➢ *abs*: allocated buffer size.
    - ➢ *afm*: available free JVM memory

  - ✓ If $x < (afm * threshold)$, then kvbuffer can be expanded to *x*, else we will set a flag to indicate map task to handle the remaining output in the way the raw version Hadoop does.

RESOURCE USAGE PIPELINE

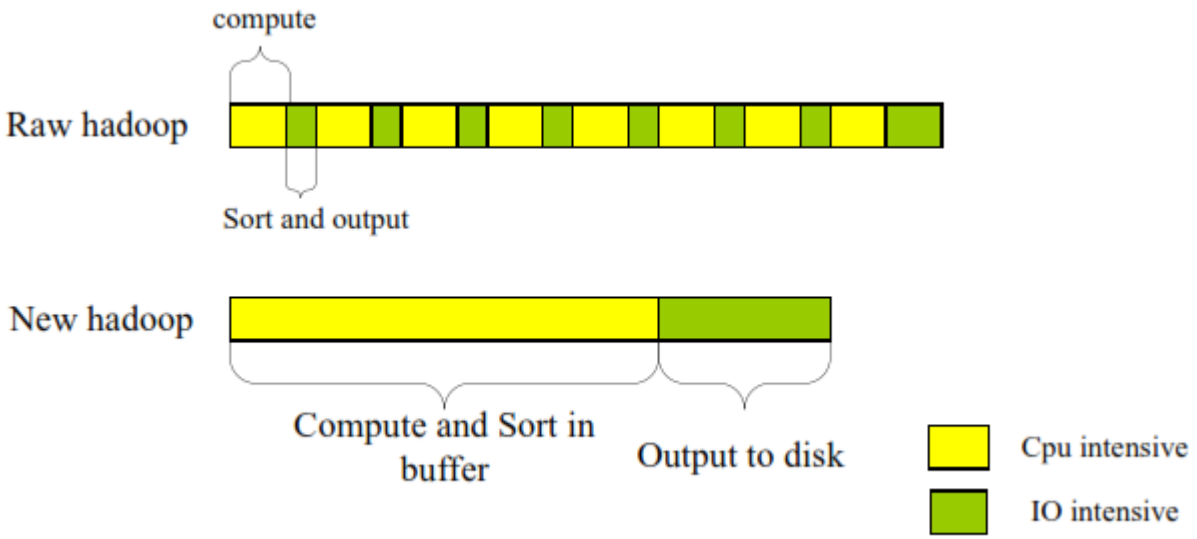


Figure4: A map task process analysis from resource usage perspective.

RESOURCE USAGE PIPELINE

- **Dynamic Slot Request of Map Task**
  - ✓ Additional map slot is needed , when map task enters into the IO-intensive period then the task tracker can ask for a new map task
  - ✓ Two different resource usage periods can be overlapped
  - ✓ mapred.tasktracker.map.tasks.maximum is the sum of normal map slot and additional map slot

RESOURCE USAGE PIPELINE

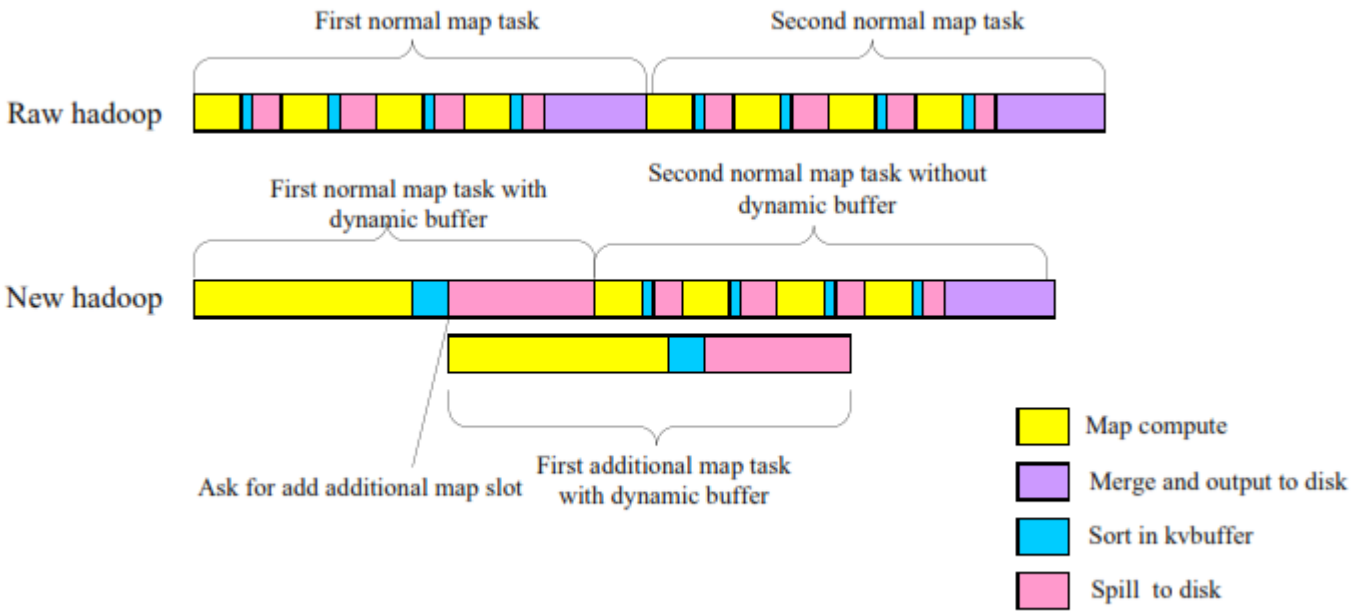
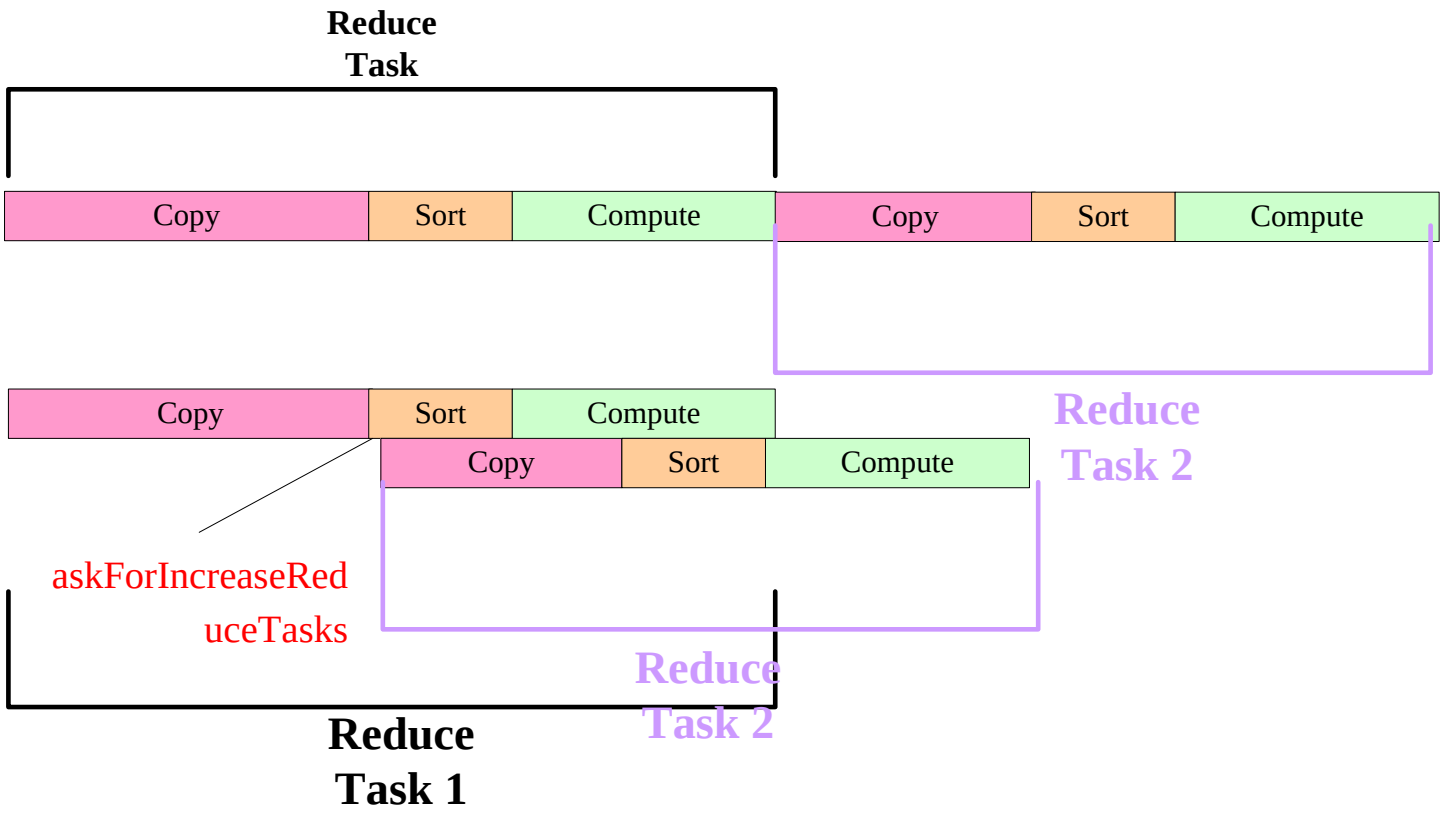


Figure5: Dynamic slot request

RESOURCE USAGE PIPELINE



RESOURCE USAGE PIPELINE

- **Dynamic Slot Request of Reduce Task**
  - ✓ Shuffle period and sort-compute period which provides the possibility to implement the resource utilization pipeline
  - ✓ Three subparameters:

    mapred.tasktracker.reduce.toal.tasks

    - ➢ The total running reduce tasks is not greater than the maximum reduce tasks at any time.

    mapred.tasktracker.reduce.shuffle.tasks

    - ➢ Running shuffling tasks is not greater than maximumshuffling tasks at any time.

    mapred.tasktracker.reduce.compute.tasks

    - ➢ Running sort-computing tasks is not greater than maximum sort-computing tasks at any time.

RESOURCE USAGE PIPELINE

Guideline for parameters selection

- ✓ A formula applying to both maximum shuffling tasks slot and maximum sort-computing tasks slot.
- ✓ Suppose:
  - Avg_x is the average running time in raw version;
  - $Gain_x$ is to be the total gain of a job in reduce phase if mapred.tasktracker.reduce.tasks.maximum set to be *x*;
  - $Lose_x$ to be the total lose of a job.

So we get ($m < n$):

$$Gain_m = (Avg_n - Avg_m) * P/n$$

$$Lose_m = (P/m - P/n) * Avg_m$$

- ✓ In a reduce phase:

If $Gain_m > Lose_m \Rightarrow Avg_n/n > Avg_m/m$

Then we can get: set mapred.tasktracker.reduce.tasks.maximum to be m is better than n and vice versa.

Evaluation

- **Definition:**
  - ✓ Throughput (T) : the number of jobs finished in unit time. If we finished n jobs in time t, then we get: $T = n / t$
  - ✓ The throughput increased by I ($n_{raw} = n_{new}$):

$$I = \frac{T_{new} - T_{raw}}{T_{raw}} = \frac{n_{new}/t_{new} - n_{raw}/t_{raw}}{n_{raw}/t_{raw}} = \frac{t_{raw}}{t_{new}} - 1$$

  - ✓ Suppose: wall time of job i is t_1 in the raw version, and t_2 in the new version, The percentage of wall time is reduced by r_i. Then:

$$r_i = \frac{t_1 - t_2}{t_1}$$

  - ✓ The average wall time gain for all jobs in the workload is r_{ave}

$$r_{ave} = \frac{1}{n} \sum_{i=1}^{n} r_i$$

Evaluation

- **Environment**
  - ✓ The cluster is configured in one rack;
  - ✓ Operating system is CentOS release 5.3, Linux version 2.6.18-128.el5;
  - ✓ Apache Hadoop version 0.20.2;
  - ✓ JDK version 1.6.0_14.

Table 1: Hardware configuration list

#nodes	11
#CPU in each node	4
#core in each CPU	1
CPU	AMD 1.8GHz
Memory in each node	5.9G
Disk in each node	186G
Network	Gigabit

Evaluation

- **Microbenchmark**

- ✓ A data input set of 27G using TextWriter;
- ✓ Run a monsterquery job including 200 map tasks and 100 reduce tasks with 128mb block size;
- ✓ Using Hadoop-0.20.2 and FairScheduler as a comparison;

Evaluation

- **Impact of Map slot and job type on performance**

Table 2: CPU utility percent in map phase

	user CPU	sys CPU	iowait CPU	idle CPU
raw version	87.69	6.43	0.81	5.06
new version	93.71	4.00	0.22	2.07

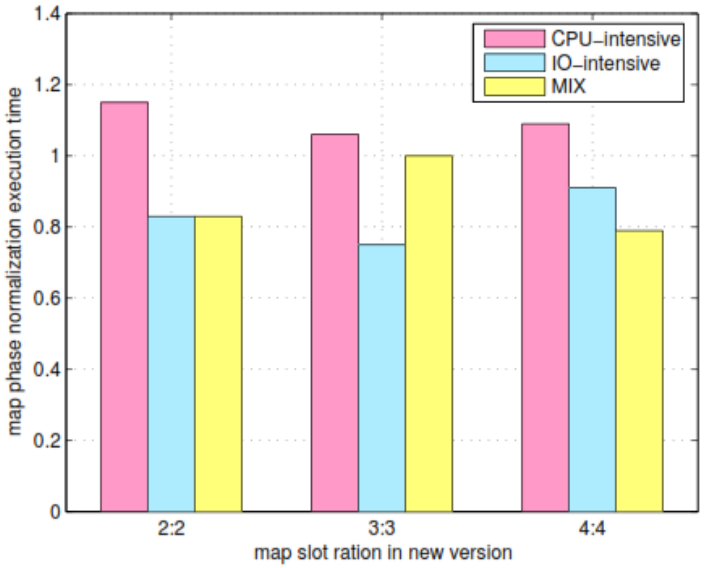


Figure6: impact of job type and map slot ratio on performance compared to raw version whose map slot is set to 4

Evaluation

- **Memory Resource in Map phase**

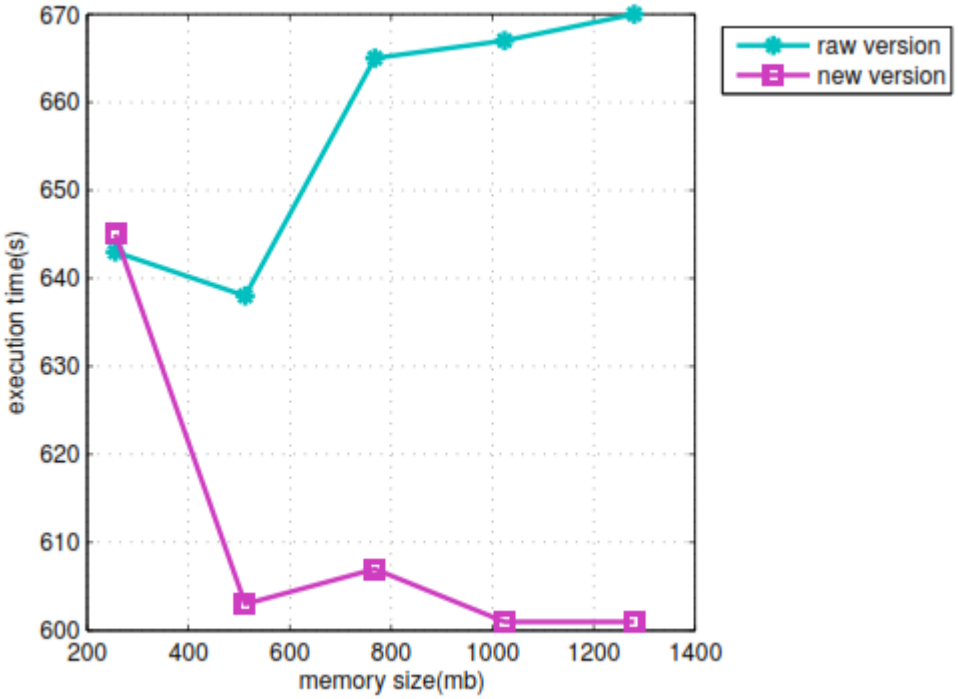


Figure7: Effect with memory size change ranging from 256 to 1280

Evaluation

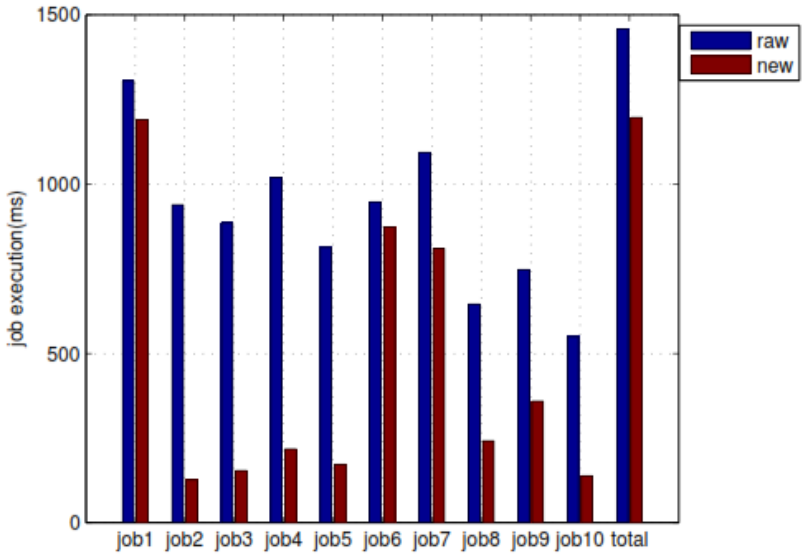
- **Macrobenchmark**
- ✓ Using 10 jobs with different input size and type;
- ✓ Using Hadoop-0.20.2 and FairScheduler as a comparison;
- ✓ We submit each job by a time interval to simulate the real environment, because different users will submit jobs at different times. The time interval is 1 min.

Table 7: The macrobenchmark

Job size	Big	Middle	Small
input	25G	10G	2.5G
type	CPU/IO	CPU/IO/Mix	CPU/IO/Mix
number	2	3	5

Evaluation

- **Job execution time:**



- ✓ We can get:

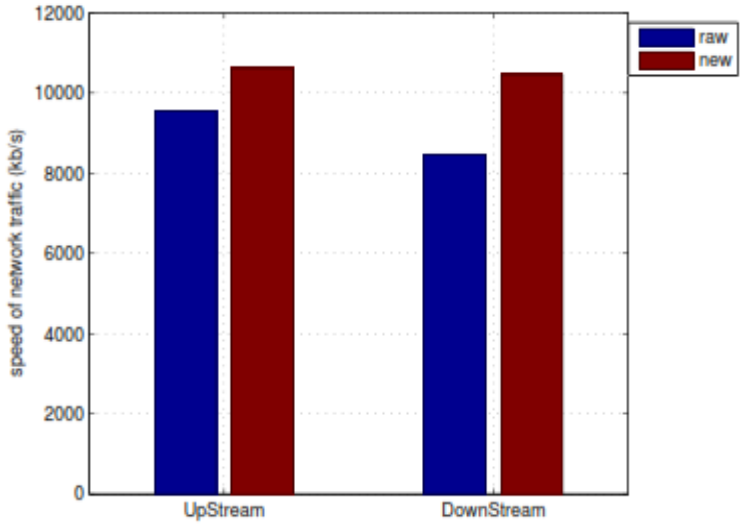
$$I = \frac{t_{raw}}{t_{new}} - 1 = \frac{1457}{1197} - 1 = 21.72\%$$

$$r_{ave} = \frac{1}{10}\sum_{i=1}^{10} r_i = 55.83\%$$

Evaluation

Table 8: CPU utility in the dynamic resource split test

	user CPU	sys CPU	iowait CPU	idle CPU
raw version	70.82	10.16	10.87	8.15
new version	83.75	10.57	4.26	1.42



Compare to raw Hadoop:

CPU:

- *userCPU* is 12.93% higher;
- *iowaitCPU* is 6.61% less;
- *idleCPU* is 6.73% less.

Net I/O :

- upstream speed is increased by 11.3%
- downstream speed is increased by 23.5%.


Thank You !
LOGO