Real-time Scheduling of Skewed MapReduce Jobs in Heterogeneous Environments

Nikos Zacheilas, Vana Kalogeraki

Department of Informatics

Athens University of Economics and Business

Introduction

- Big Data era has arrived!
  - Facebook processes daily more than 500 TB of data
  - Twitter users generate 500M tweets per day
  - Dublin's city operational center receives over 100 bus GPS traces per minute
- Wide range of domains
  - Traffic monitoring
  - Inventory management
  - Healthcare infrastructures
- More data than we can handle with traditional approaches (e.g. relational databases)
- Novel frameworks were proposed
  - Batch processing
    - Google's MapReduce
    - IBM's BigInsights
    - Microsoft's Dryad
  - Stream processing
    - Storm
    - IBM's Infosphere Streams

The MapReduce Model

- MapReduce [Dean@OSDI2004] was proposed as a powerful and cost-effective approach for massive scale batch processing
- Popularized via its open source implementation, Hadoop, is used by some of the major computer companies:
  - Yahoo!
  - Twitter
  - Facebook
- Intense processing jobs are broken into smaller tasks
- Two stages of processing *map* and *reduce*

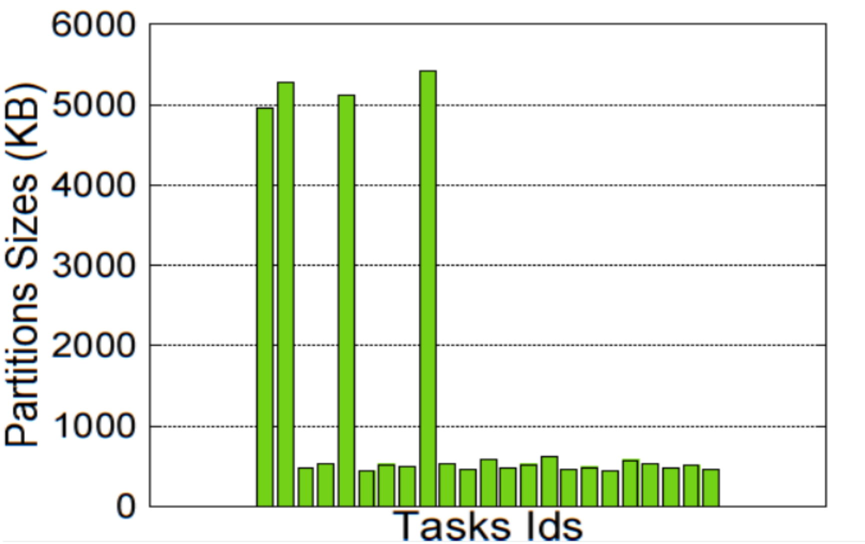
$$\text{map}(k_1, v_1) \rightarrow [k_2, v_2]$$

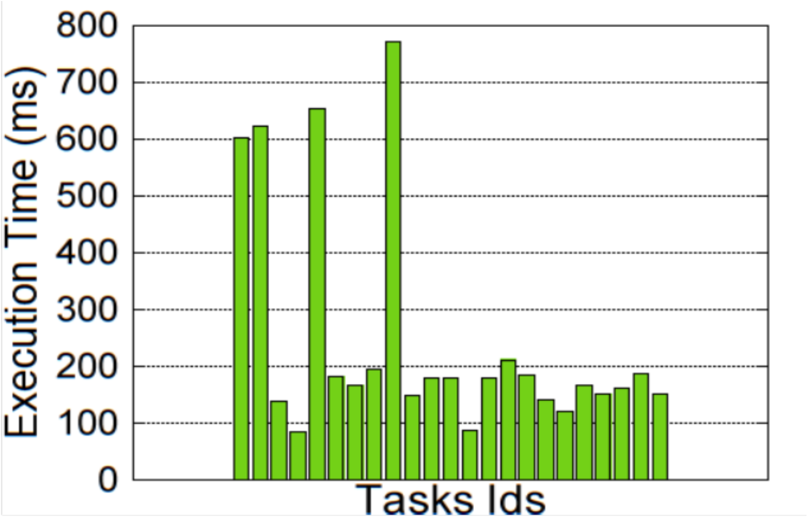
$$\text{reduce}(k_2, [v_2]) \rightarrow [k_3, v_3]$$

- All $[k_2, v_2]$ intermediate pairs assigned to the same *reduce* task are called a reduce task's *partition*

Processing Big Data with MapReduce Challenges

- Load imbalances due to skewed data
- Heterogeneous environments with heterogeneous processing capabilities
- Real time response requirements
  - 95% of Facebook's MapReduce jobs have average execution time of 30 seconds [Chen@MASCOTS2011]





Youtube social graph application

Problem

Question: How can we efficiently schedule the execution of multiple MapReduce jobs with real-time response requirements?

Challenges:

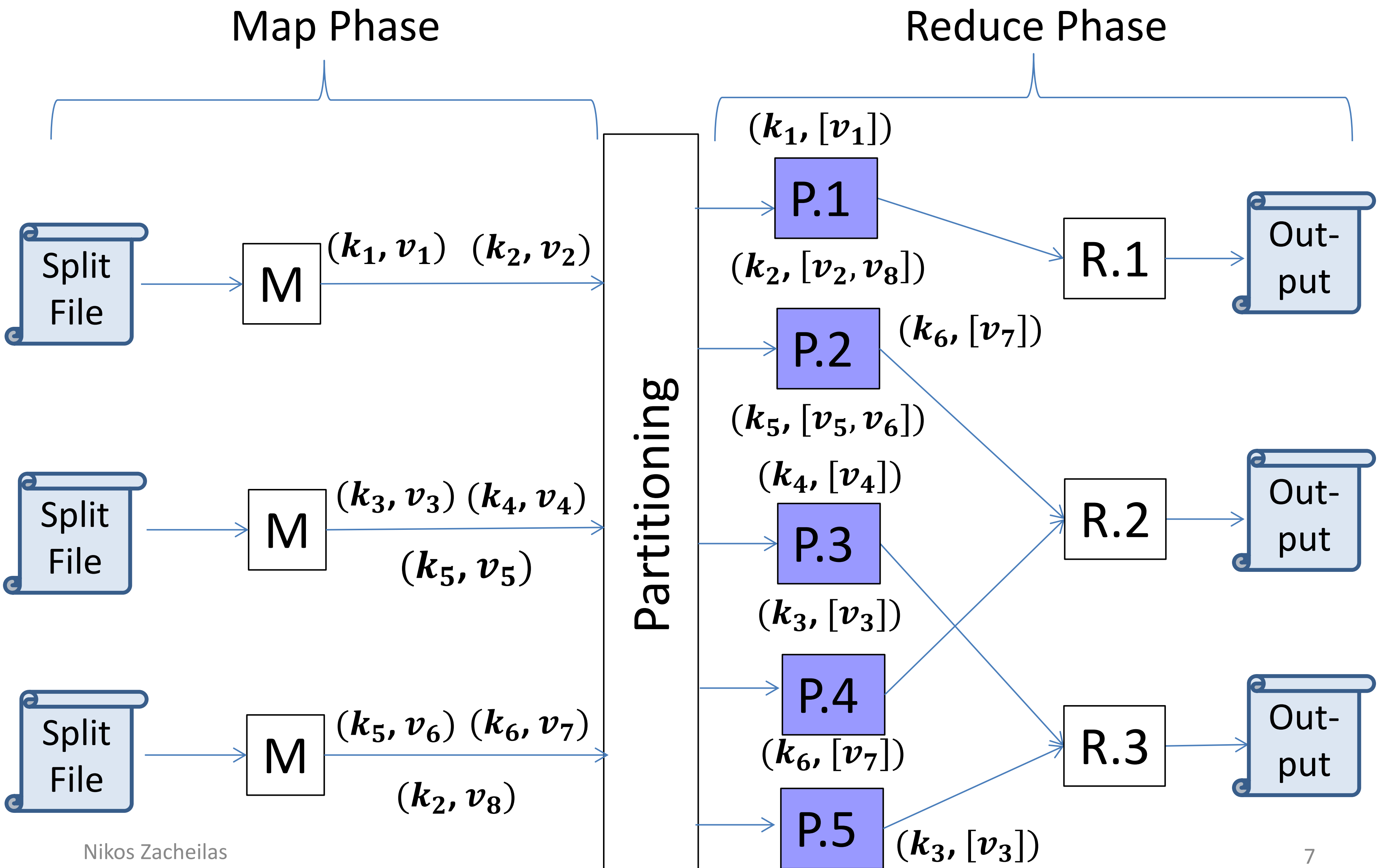
- Maximize the probability of meeting end-to-end real-time response requirements
- Effectively handle skewed data
- Identify overloaded nodes
- Deal with heterogeneous environments

DynamicShare System

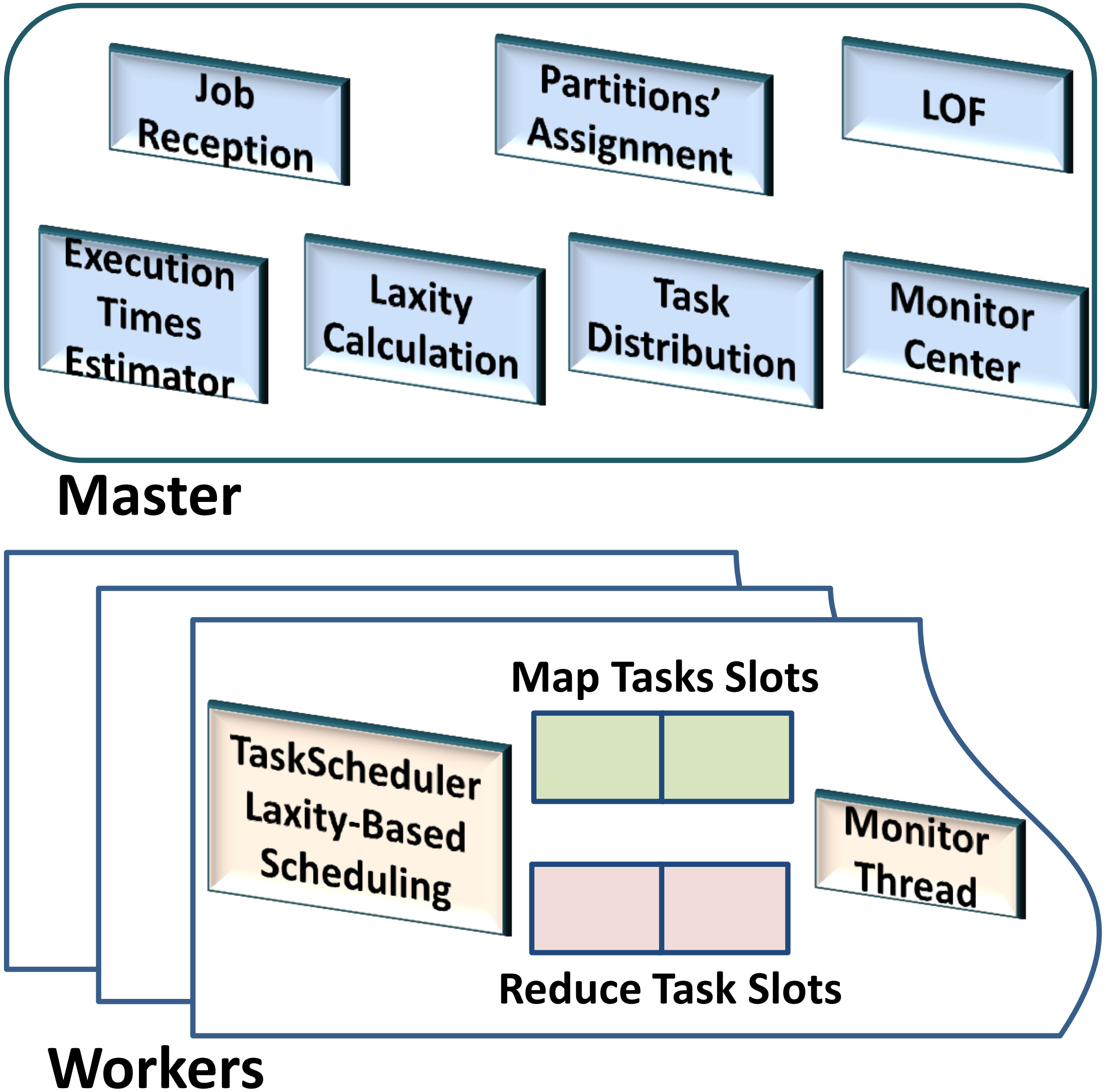
We propose DynamicShare a novel MapReduce framework for heterogeneous environments. Our approach makes the following contributions:

- New jobs' execution times estimation model based on *non-parametric regression*
- Distributed least laxity first scheduling of jobs' tasks to meet end-to-end demands
- Early identification of overloaded nodes through Local Outlier Factor algorithm
- Handling data skewness with two approaches:
  - Simple partitions' assignment
  - Count-Min Sketch assignment

The MapReduce Model



DynamicShare Architecture



- DynamicShare comprises a single Master and multiple Worker nodes
- Master node
  - responsible for assigning map and reduce tasks to Workers under skewness and real-time criteria
  - monitor jobs performance
- Worker nodes
  - execute map/reduce tasks
  - report task progress

System Model

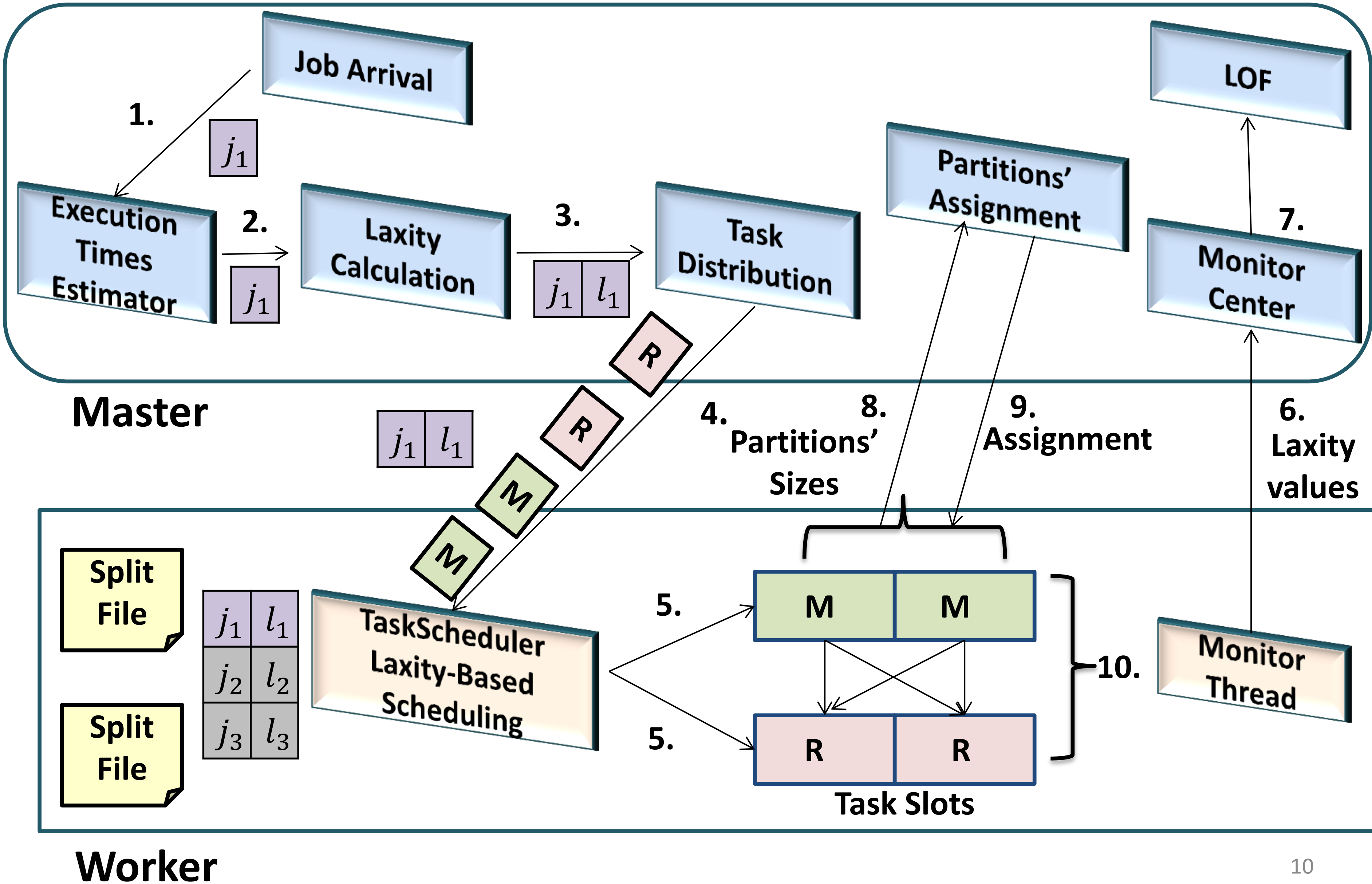
Each submitted job j comprises a sequence of invocations of *map* and *reduce* tasks. Each job j is characterized by:

- $Deadline_j$ is the time interval, starting at job initialization, within which job j must be completed
- $Proj\_exec\_time_j$: the estimated amount of time required for the job to complete. Estimation is given by the following Equation: $Proj\_exec\_time_j = \max\{m_{i,t}, \dots, m_{k,t}\} + \max\{r_{z,t}, \dots, r_{l,t}\}$
- $Laxity_j$: the difference between $Deadline_j$ and $Proj\_exec\_time_j$, considered a metric of urgency for job
- $split\_size_j$: the size of a split file

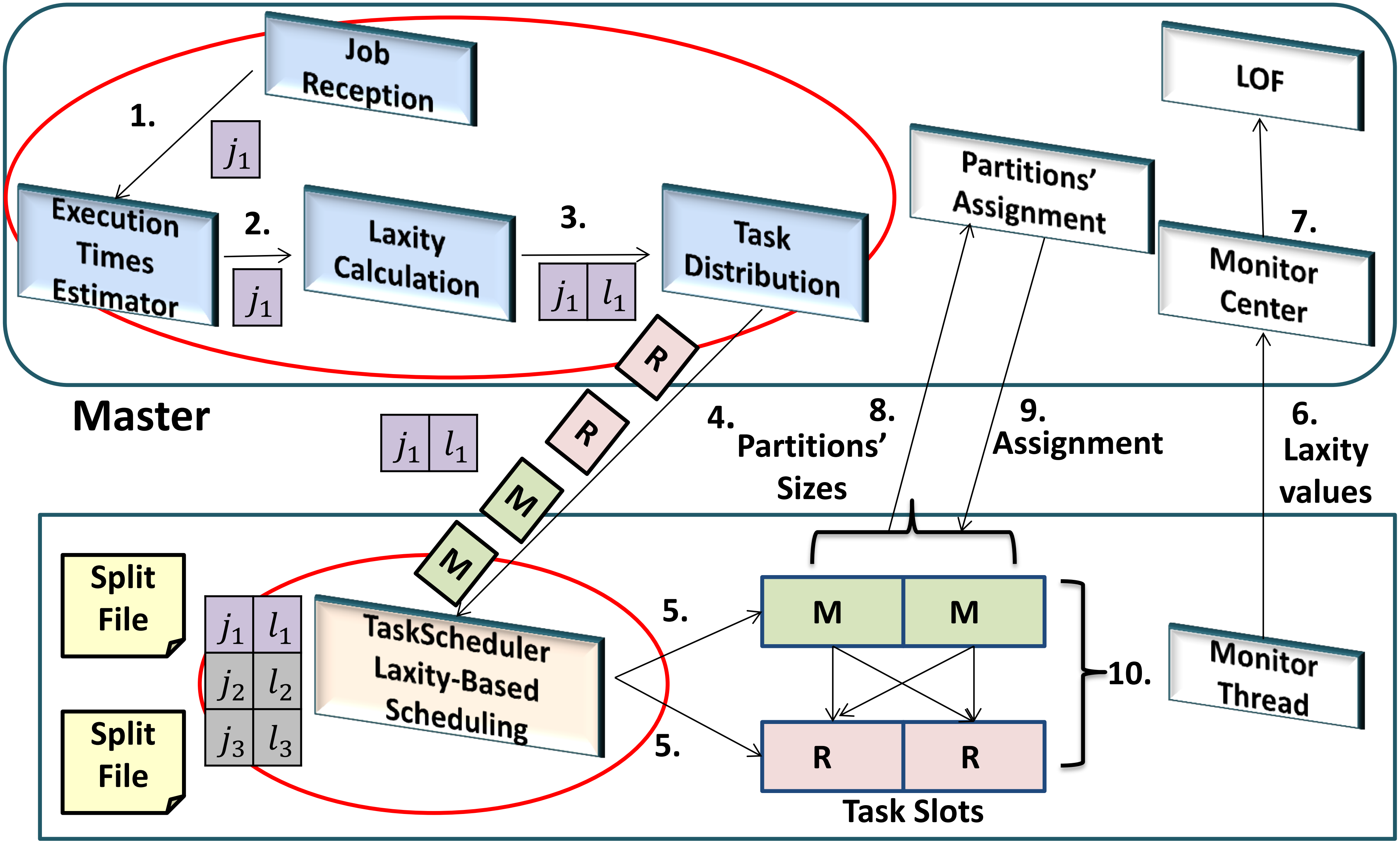
Each task t of job j has the following parameters:

- $m_{i,t}$, $r_{i,t}$: estimated execution times of map and reduce tasks in Worker i
- $cpu_{i,t}, memory_{i,t}$: average CPU and memory usage of task t in Worker i

DynamicShare: How it works?



Task Scheduling

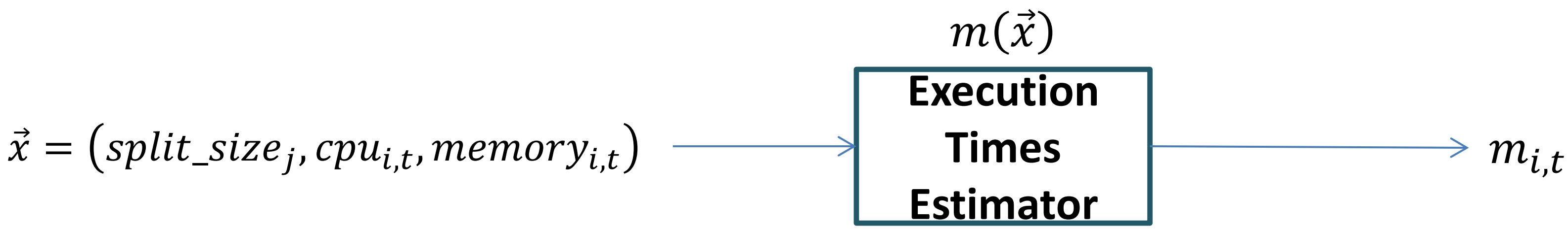


Task Scheduling

- Given the $Deadline_j$ and $Proj\_exec\_time_j$ for job j, we compute the $Laxity_j$ value with the following formula
$$Laxity_j = Deadline_j - Proj\_exec\_time_j$$
- Least laxity scheduling is a dynamic algorithm that allow us to compensate for queueing delays experienced by the tasks executing at the nodes
- TaskScheduler sorts jobs' tasks based on the $Laxity_j$ values. Tasks of jobs with the smaller *laxity* values will be closer to the head of the queue
- Scheduling decisions are made when:
  1. New tasks are assigned to the TaskScheduler's
  2. Tasks finish or miss their deadlines

Estimating Task's Execution Time

- Current solutions such as building job profiles or using debug runs are not adequate
- Works well for homogeneous environments
- What happens though in heterogeneous environments where multiple applications may share the same resources?
- Need to take into account the resource requirements (e.g., CPU, memory usage) of newly submitted tasks



- Approximate $m(\vec{x})$ function
  - Parametric regression considers the functional form known
  - Non-parametric regression makes no assumption (**data-driven technique**)

Estimating Task's Execution Time

$$\hat{m}(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} W_i\,(\vec{x}) \cdot y_i$$

$\vec{x}$ → **Execution Times Estimator** → $m_{i,t}$

Past runs

Vector	Execution Time
$\vec{x}_1$	y_1
…	…
$\vec{x}_n$	y_n

Non-parametric Regression

$$\hat{m}(\vec{x}) = \frac{1}{k}\sum_{i=1}^{k} W_i\,(\vec{x}) \cdot y_i$$

$\vec{x}$ → **Execution Times Estimator** → $m_{i,t}$

Past runs

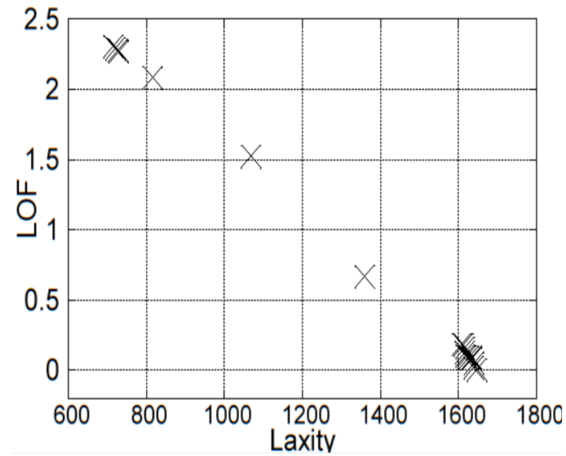
Use k closest in Euclidean distance past runs

Vector	Execution Time
$\vec{x}_1$	y_1
…	…
$\vec{x}_n$	y_n

k-Nearest Neighbor (k-NN) Smoothing

Identifying Overloaded Nodes

- Due to the dynamic behavior of the jobs Workers performance may change rapidly. Need to quickly detect overloaded Workers
- We consider overloaded nodes those that are assigned more tasks than their processing capabilities
- **Key Observation**: *Laxity* values of these tasks will be left behind in relation to the tasks running in different nodes
- **Solution:** Applied Local Outlier Factor algorithm (**LOF**) on the *laxity* values of the tasks of the same job that run on different Workers
- $LOF_l(lax_A) := \frac{\sum_{lax_B \in N_l(lax_A)} lrd_l(lax_B)}{|N_l(lax_A)| * lrd_l(lax_A)}$
- Compares reachability density of a point with each neighbors



Handling Skewed Data

In our system two types of skew frequently occur:

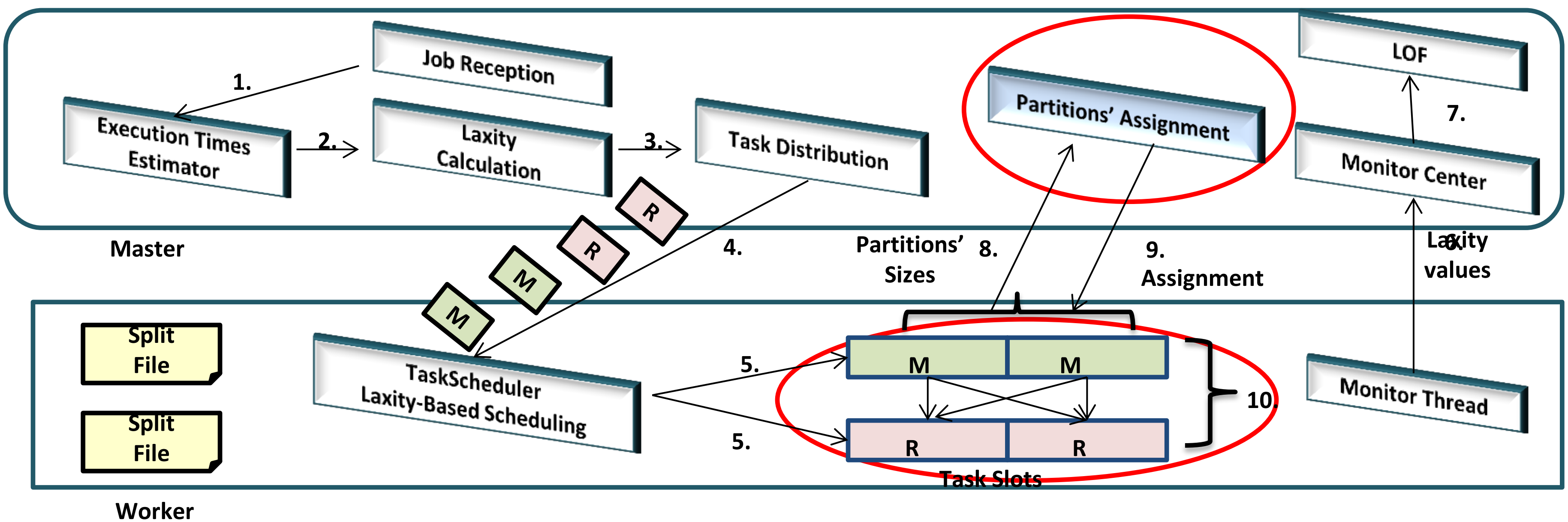
- *Skewed Key Frequencies*
- *Skewed Tuple Sizes*

Idea: Use more *partitions* than the original MapReduce

Problem: How to assign *partitions* to the *reduce* tasks in order to minimize the reduce phase execution time?

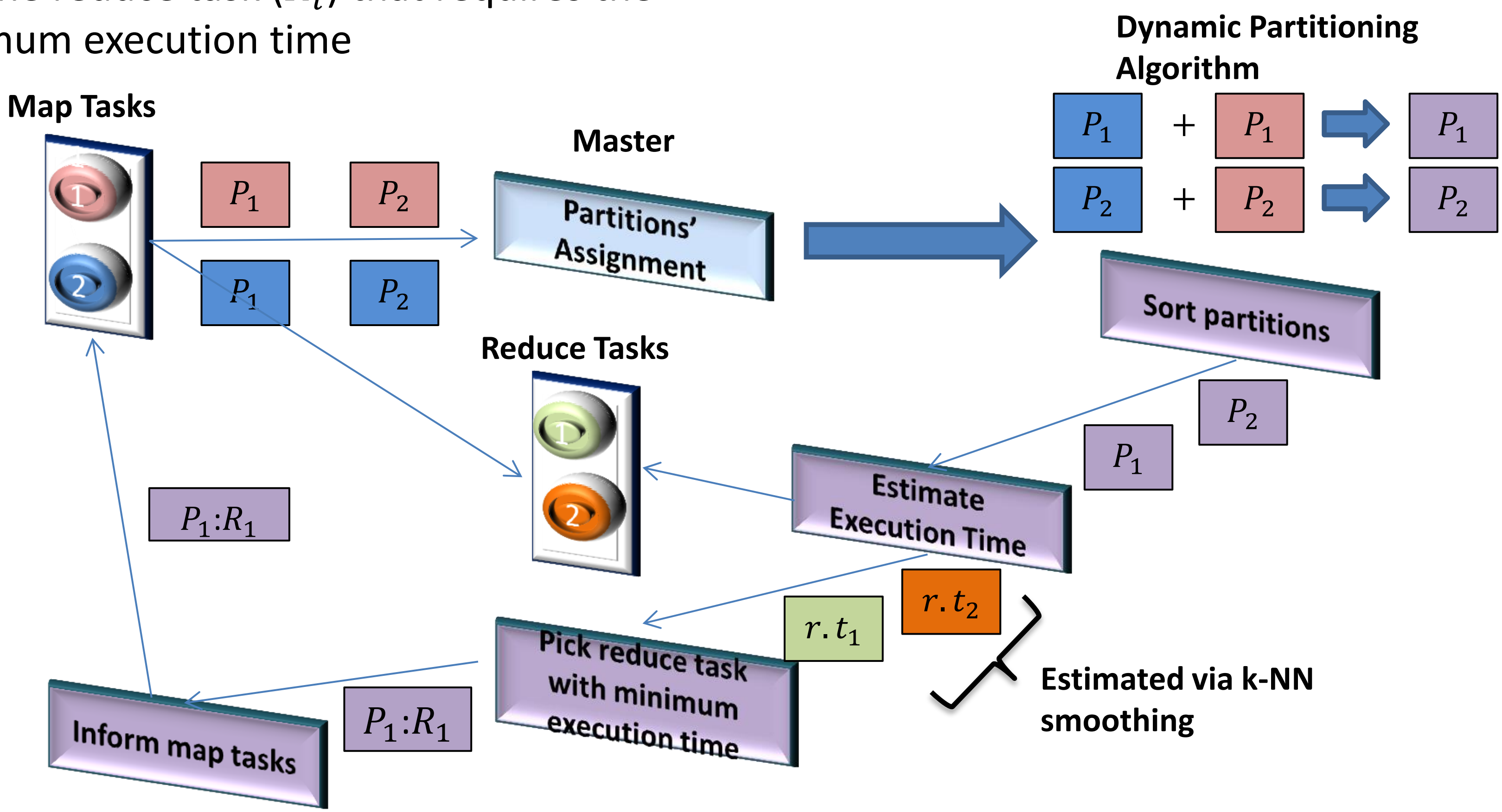
Exploit two approaches:

- **Simple Partitions' Assignment**
- **Count Min Sketch Assignment**



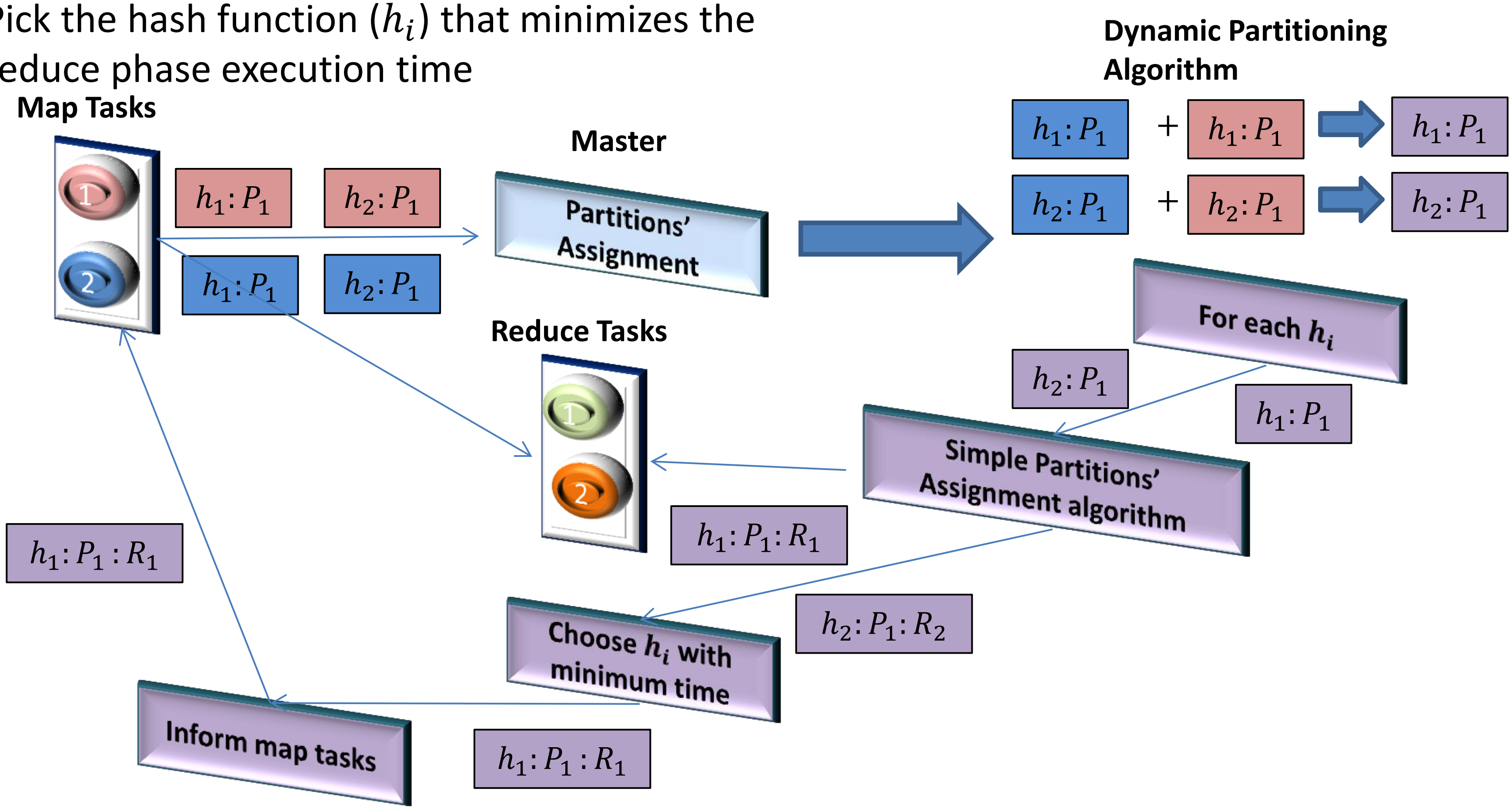
Simple Partitions' Assignment

1. Calculate partitions sizes (P_i)
2. Sort partition sizes
3. Estimate the execution times ($r.t_i$) of assigning each partition to the available reduce tasks
4. Pick the reduce task (R_i) that requires the minimum execution time



Count-Min Sketch Assignment

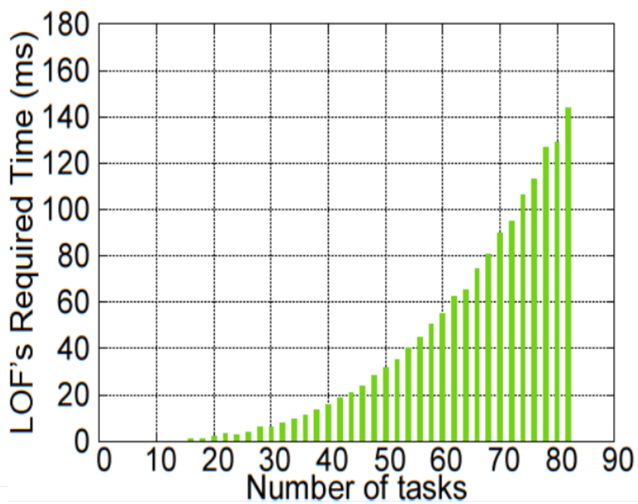
1. Calculate partitions' sizes (P_i) for each hash function (h_i)
2. For each hash function apply Simple Partitions Assignment algorithm
3. Pick the hash function (h_i) that minimizes the reduce phase execution time

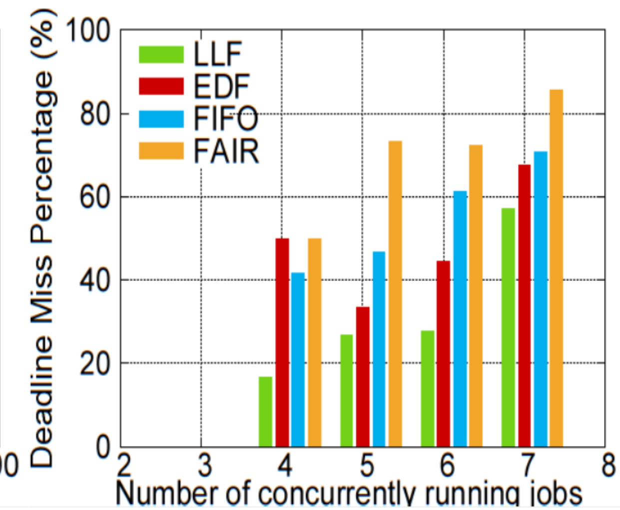


Implementation

- We implemented and evaluated DynamicShare on Planetlab. Fourteen nodes were used with 82 processing cores. One dedicated node was the Master and the others used as Workers
- Two MapReduce jobs were issued:
  - A Twitter friendship request query on 2GB of available tweets. 59 map and 23 reduce tasks were used
  - A Youtube friends counting application for a 39MB Youtube social graph. Again 59 map and 23 reduce tasks were used
- Compared our scheduling proposal with:
  - Earliest Deadline First (EDF)
  - FIFO
  - FAIR
- Our partitioning algorithms were compared to:
  - Load Balance [Gufler@CLOSER2011]
  - Hadoop
  - Skewtune [Kwon@SIGMOD2012]

Experiments



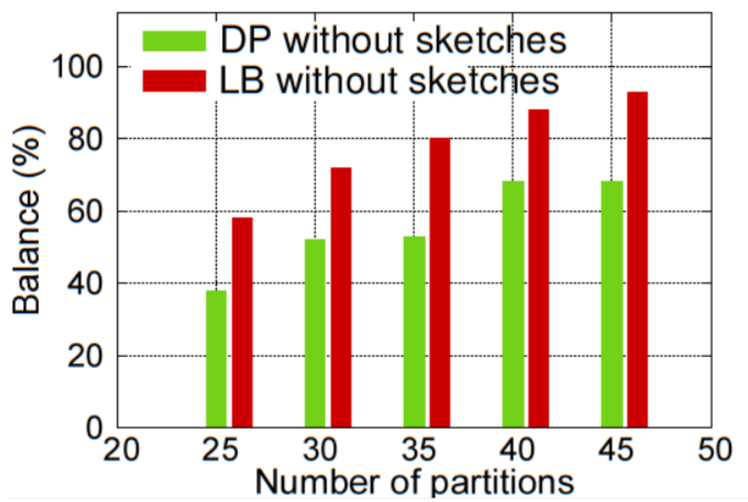


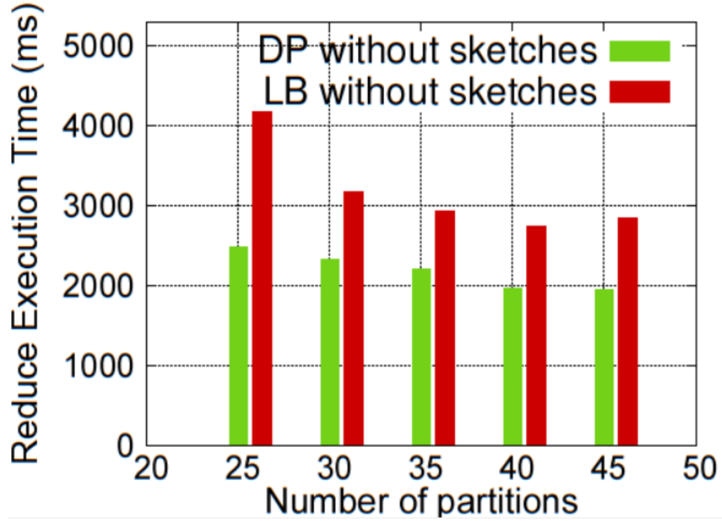
- **k-NN Smoothing Performance**
- Initially when not enough data are available, the estimated value is larger than the actual
- Better prediction when more past runs are used

- **LOF Execution time**
- LOF depends on the number of tasks used by a job
- Even for great number of tasks the algorithm is capable of detecting outliers in respectable amount of time

- **Deadline misses comparison**
- LLF maintains the percentage of deadline misses at the lowest possible level
- Takes into account the current system conditions for the assignment

Experiments

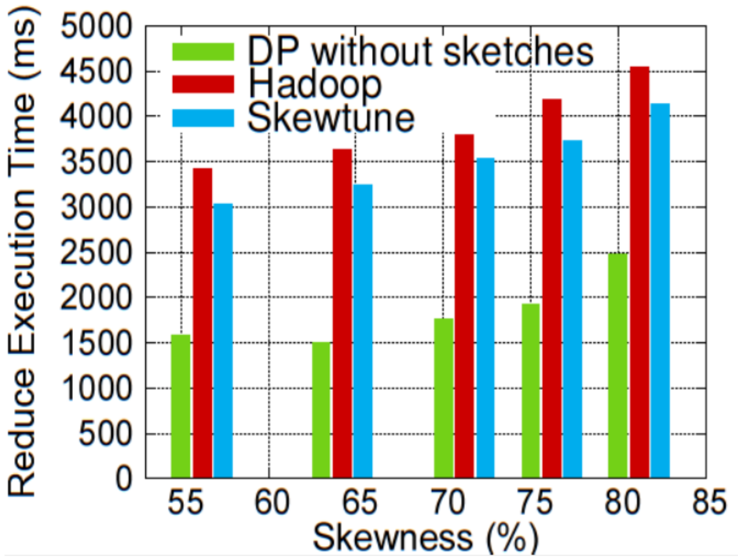


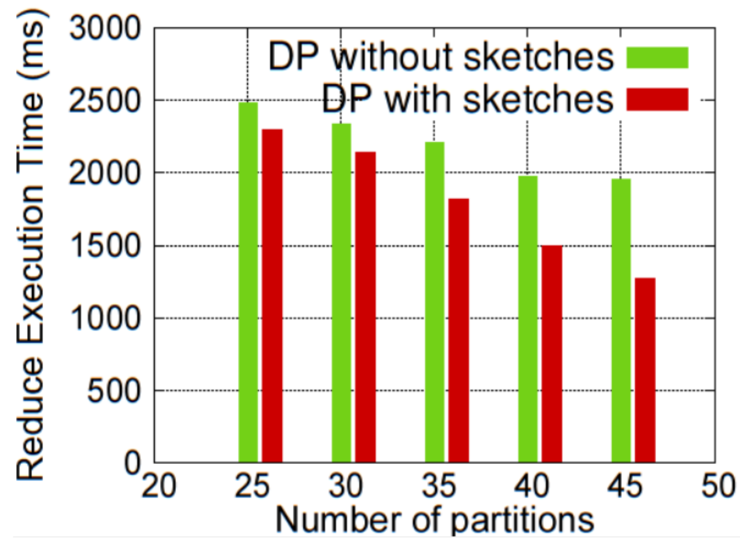


- **Comparing LB with DP in regards to achieved balance**
- LB has better results because it considers a fair distribution of the partitions to the available reduce tasks
- DP does not consider balance in the assignment

- **Comparing DP with LB in regards to achieved execution time**
- Balance is not the correct approach for heterogeneous environments
- DP's opportunistic assignment exploits high performance nodes by assigning extra partitions

Experiments





- **Comparing DP with Skewtune and Hadoop partitioning**
- Hadoop leads to the execution of large partitions to slow nodes
- Skewtune repartitioning cost is prohibitive for short jobs
- DP does an appropriate one time assignment
- Similar results were observed in Youtube job

- **Comparing DP with and without sketches**
- DP with sketches achieves better results than DP without sketches, because more partitions assignments are possible
- However the overhead of the algorithm is not negligible. When sketches are applied DP requires approximately 200 ms while without sketches only 80 ms

Conclusions and Future Work

- We proposed a new framework for handling MapReduce jobs with real-time constraints in highly heterogeneous environments using:
  - *non-parametric* regression for estimating tasks' execution times
  - *Least Laxity First* scheduling of jobs' tasks in the available slots
  - *Local Outlier Factor* for detecting overloaded nodes
  - Dynamic Partitioning algorithms for handling skewed data
- Evaluated our proposal in Planetlab, and the results point out that our system achieves its goals
- Future work:
  - Dynamically decide the number of partitions and examine the trade-off between the reduce phase execution time and the two partitioning algorithms

Thank you

Questions??