Data Locality in MapReduce

Loris Marchal[1] Olivier Beaumont[2]

1: CNRS and ENS Lyon, France.

2: INRIA Bordeaux Sud-Ouest, France.

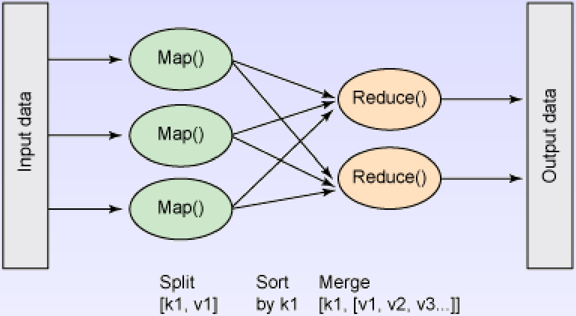
New Challenges in Scheduling Theory — March 2016

MapReduce basics

- Well known framework for data-processing on parallel clusters
- Popularized by Google, open source implementation: *Apache Hadoop*
- Breaks computation into small tasks, distributed on the processors
- Dynamic scheduler: handle failures and processor heterogeneity
- Centralized scheduler launches all tasks
- Users only have to write code for two functions:
  - Map: filters the data, produces intermediate results
  - Reduce: summarizes the information

- Large data files split into chunks that are scattered on the platform (e.g. using HDFS for Hadoop)
- Goal: process computation near the data, avoid large data transfers

MapReduce example



Textbook example: WordCount (count #occurrences of words in a text)

1. Text split in chunks scattered on local disks
2. Map: compute #occurrences of words in each chunk, produces results as <word,#occurrences> pairs
3. Sort and Shuffle: gather all pairs with same word on a single processor
4. Reduce: merges results for single word (sum #occurrences)

Other usages of MapReduce

- Several phases of Map and Reduce (tightly coupled applications)
- Only Map phase (independent tasks, divisible load scheduling)

MapReduce locality

Potential data transfer sources:

- Sort and Shuffle: data exchange between all processors
  - Depends on the applications (size and number of <key,value> pairs)
- Map task allocation: when a Map slot is available on a processor
  - choose a local chunk if any
  - otherwise choose any unprocessed chunk and transfer data

Replication during initial data distributions:

- To improve (data locality) and fault tolerance
- Optional, basic setting: 3 replicas
  - first, chunk placed on a disk
  - one copy sent to another disk of the same rack (local communication)
  - one copy sent to another rack

Objective of this study

Analyze the data locality of the Map phase:

1. estimate the volume of communication
2. estimate the load imbalance without communication

Using a simple model, to provide good estimates and measure the influence of key parameters:

- Replication factor
- Number of tasks and processors
- Task heterogeneity (to come)

Disclaimer: work in progress
Comments/contributions welcome!

Outline

Outline

Introduction & motivation

Related work

Volume of communication of the Map phase

Load imbalance without communication

Conclusion

Related work 1/2

MapReduce locality:

- Improvement Shuffle phase
- Few studies on the locality for the Map phase (mostly experimental)

Balls-into-bins:

- Random allocation of n balls in p bins:
  - For $n = p$, maximum load of $\log n / \log \log n$
  - Estimation of maximum load with high probability for $n \geq p$ [Raab & Steeger 2013]
- Choosing the least loaded among r candidates improves a lot
  - "Power of two choices" [Mitzenmacher 2001]
  - Maximum load $n/p + O(\log \log p)$ [Berenbrick et al. 2000]
- Adaptation for weighted balls [Berenbrick et al. 2008]

Related work 1/2

MapReduce locality:

- Improvement Shuffle phase
- Few studies on the locality for the Map phase (mostly experimental)

Balls-into-bins:

- Random allocation of n balls in p bins:
  - For $n = p$, maximum load of $\log n / \log \log n$
  - Estimation of maximum load with high probability for $n \geq p$ [Raab & Steeger 2013]
- Choosing the least loaded among r candidates improves a lot
  - "Power of two choices" [Mitzenmacher 2001]
  - Maximum load $n/p + O(\log \log p)$ [Berenbrick et al. 2000]
- Adaptation for weighted balls [Berenbrick et al. 2008]

Related work 2/2

Work-stealing:

- Independent tasks or tasks with precedence
- Steal part of a victim's task queue in time 1
- Distributed process (steal operations may fail)
- Bound on makespan using potential function [Tchiboukdjian, Gast & Trystram 2012]

Outline

Problem statement – MapReduce model

Data distribution:

- p processors, each with its own data storage (disk)
- n tasks (or chunks)
- r copies of each chunk distributed uniformly at random

Allocation strategy:

- whenever a processor is idle:
  - allocate a local task is possible
  - otherwise, allocate a random task, copy the data chunk
  - invalidate all other replicas of the chosen chunk

Cost model:

- Uniform chunk size (parameter of MapReduce)
- Uniform task durations

Question:

- Total volume of communication (in chunk number)

Problem statement – MapReduce model

Data distribution:

- p processors, each with its own data storage (disk)
- n tasks (or chunks)
- r copies of each chunk distributed uniformly at random

Allocation strategy:

- whenever a processor is idle:
  - allocate a local task is possible
  - otherwise, allocate a random task, copy the data chunk
  - invalidate all other replicas of the chosen chunk

Cost model:

- Uniform chunk size (parameter of MapReduce)
- Uniform task durations

Question:

- Total volume of communication (in chunk number)

Problem statement – MapReduce model

Data distribution:

- p processors, each with its own data storage (disk)
- n tasks (or chunks)
- r copies of each chunk distributed uniformly at random

Allocation strategy:

- whenever a processor is idle:
  - allocate a local task is possible
  - otherwise, allocate a random task, copy the data chunk
  - invalidate all other replicas of the chosen chunk

Cost model:

- Uniform chunk size (parameter of MapReduce)
- Uniform task durations

Question:

- Total volume of communication (in chunk number)

Problem statement – MapReduce model

Data distribution:

- p processors, each with its own data storage (disk)
- n tasks (or chunks)
- r copies of each chunk distributed uniformly at random

Allocation strategy:

- whenever a processor is idle:
  - allocate a local task is possible
  - otherwise, allocate a random task, copy the data chunk
  - invalidate all other replicas of the chosen chunk

Cost model:

- Uniform chunk size (parameter of MapReduce)
- Uniform task durations

Question:

- Total volume of communication (in chunk number)

Simple solution

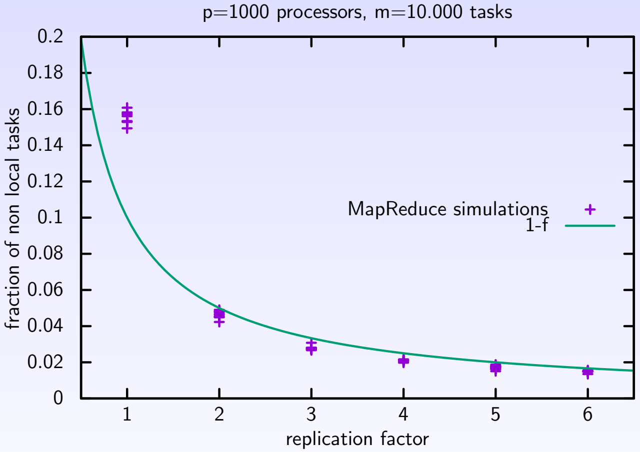
- Consider the system after k chunks have been allocated
- A processor i requests a new task
- Assumption: the remaining $r(n-k)$ replicas are uniformly distributed
- Probability that none of them reach i:

$$p_k = \left(1 - \frac{1}{p}\right)^{r(n-k)} = 1 - \frac{r(n-k)}{p} + o\left(\frac{1}{p}\right) = e^{-r(n-k)/p} + o\left(\frac{1}{p}\right)$$

- Fraction of non-local chunks:

$$f = \frac{1}{n}\sum_k p_k = \frac{p}{rn}(1 - e^{-rn/p})$$

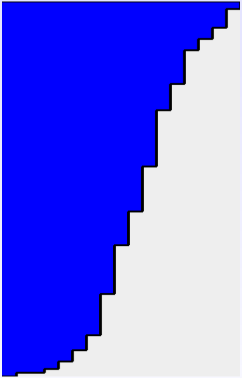
Simple solution - simulations



- Largely underestimates non-local tasks without replication ($r = 1$)
- Average accuracy with replication ($r > 1$)

Simple solution - questioning the assumption

Remaining chunks without replication:
(100 processors, 1000 tasks)

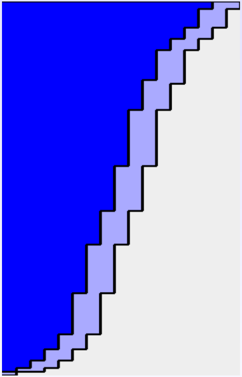


initial distribution (10 chunks/procs on average)

Non uniform distribution after some time ☹

Simple solution - questioning the assumption

Remaining chunks without replication:
(100 processors, 1000 tasks)

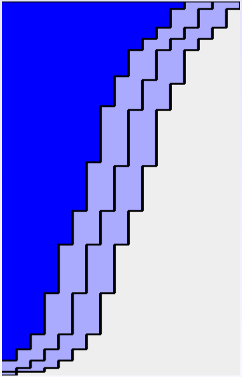


after 200 steps

Non uniform distribution after some time ☹

Simple solution - questioning the assumption

Remaining chunks without replication:
(100 processors, 1000 tasks)

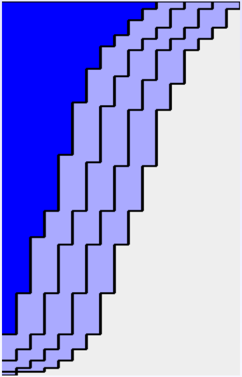


after 400 steps

Non uniform distribution after some time ☹

Simple solution - questioning the assumption

Remaining chunks without replication:
(100 processors, 1000 tasks)

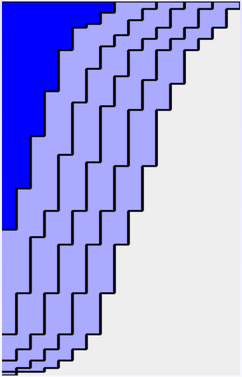


after 600 steps

Non uniform distribution after some time ☹

Simple solution - questioning the assumption

Remaining chunks without replication:
(100 processors, 1000 tasks)

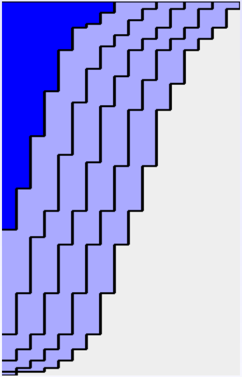


after 800 steps

Non uniform distribution after some time ☹

Simple solution - questioning the assumption

Remaining chunks without replication:
(100 processors, 1000 tasks)

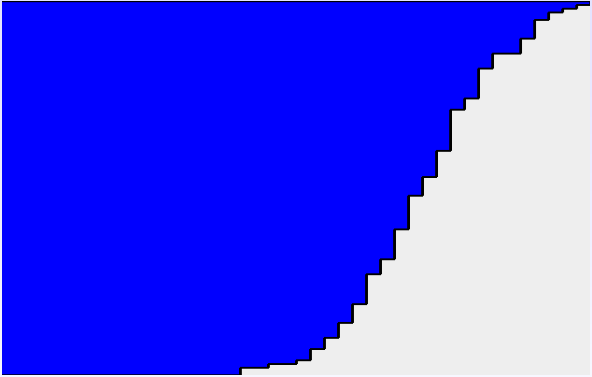


after 800 steps

Non uniform distribution after some time ☹

Simple solution - questioning the assumption

Remaining chunks with replication=3:
(100 processors, 1000 tasks)

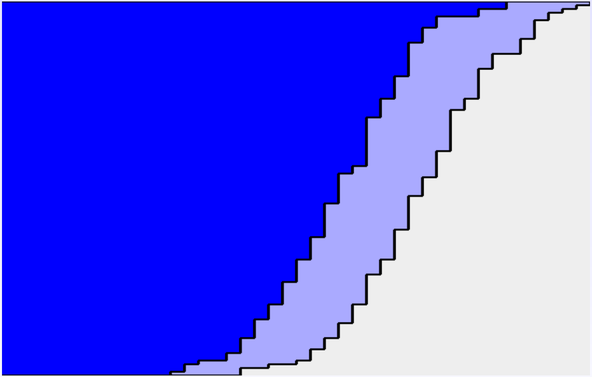


initial distribution (30 chunks/procs on average)

Uniform distribution for a large part of the execution?

Simple solution - questioning the assumption

Remaining chunks with replication=3:
(100 processors, 1000 tasks)

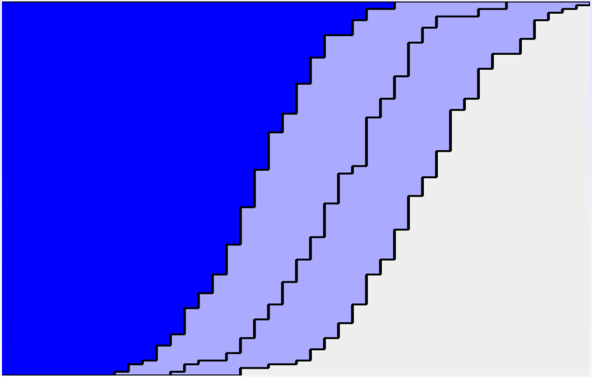


after 200 steps

Uniform distribution for a large part of the execution?

Simple solution - questioning the assumption

Remaining chunks with replication=3:
(100 processors, 1000 tasks)

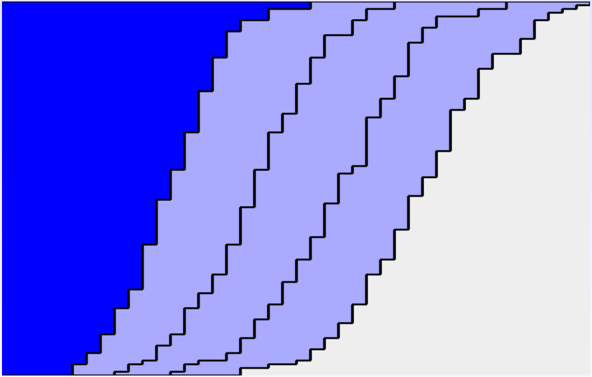


after 400 steps

Uniform distribution for a large part of the execution?

Simple solution - questioning the assumption

Remaining chunks with replication=3:
(100 processors, 1000 tasks)

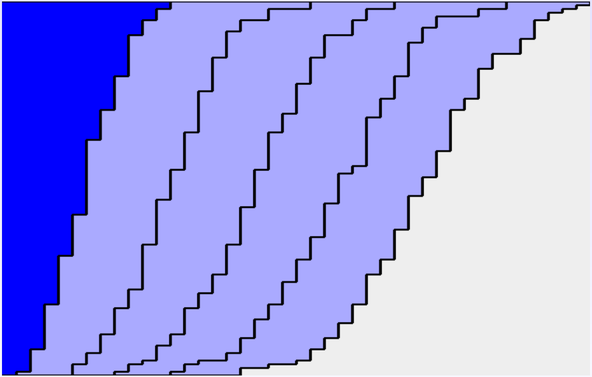


after 600 steps

Uniform distribution for a large part of the execution?

Simple solution - questioning the assumption

Remaining chunks with replication=3:
(100 processors, 1000 tasks)

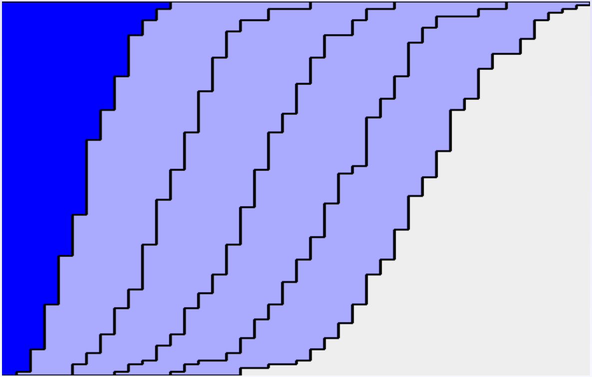


after 800 steps

Uniform distribution for a large part of the execution?

Simple solution - questioning the assumption

Remaining chunks with replication=3:
(100 processors, 1000 tasks)



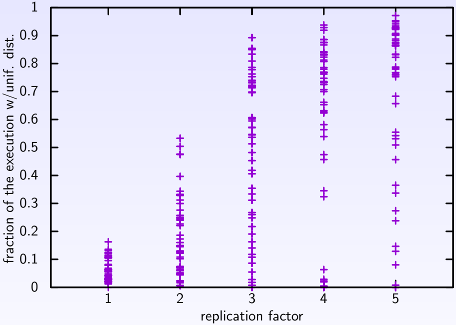
after 800 steps

Uniform distribution for a large part of the execution?

Simple solution - questioning the assumption

Assumption: after k steps, the remaining $r(n-k)$ replicas are uniformly distributed

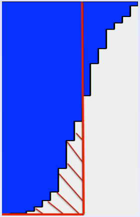
- χ^2 test to check if the distribution is uniform
- Fraction of the execution with a uniform distribution:



- For $r = 1$: non-uniform distribution for most of the execution
- For $r > 1$: uniform distribution in a majority of cases

Lower bound on communications without replication

- Consider n balls placed in p bins (initial distribution with $r = 1$)



- A processor with $k < n/p$ chunks will have to receive at least $k - n/p$ chunks
  - It may need more chunks if some of its chunks are used by other starving processors
  - Assume that we steal chunks only from overloaded processors
- Let N_k be the number of processors with exactly k chunks:

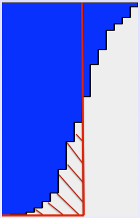
$$N_k = p \times \binom{n}{k} (1/p)^k (1 - 1/p)^{n-k}$$
$$= e^{-n/p} (n/p)^k / k! \text{ when } k \ll n, p$$

- Then, the communication volume is given by:

$$V = \sum_{k < n/p} (n/p - k) N_k = p e^{-n/p} \frac{(n/p)^{n/p+1}}{(n/p)!} \approx \sqrt{\frac{np}{2\pi}}$$

Lower bound on communications without replication

- Consider n balls placed in p bins (initial distribution with $r = 1$)
- A processor with $k < n/p$ chunks will have to receive at least $k - n/p$ chunks
  - It may need more chunks if some of its chunks are used by other starving processors
  - Assume that we steal chunks only from overloaded processors



- Let N_k be the number of processors with exactly k chunks:

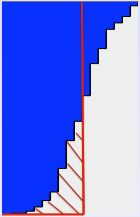
$$N_k = p \times \binom{n}{k} (1/p)^k (1 - 1/p)^{n-k}$$
$$= e^{-n/p} (n/p)^k / k! \text{ when } k \ll n, p$$

- Then, the communication volume is given by:

$$V = \sum_{k<n/p} (n/p - k) N_k = pe^{-n/p} \frac{(n/p)^{n/p+1}}{(n/p)!} \approx \sqrt{\frac{np}{2\pi}}$$

Lower bound on communications without replication

- Consider n balls placed in p bins (initial distribution with $r = 1$)
- A processor with $k < n/p$ chunks will have to receive at least $k - n/p$ chunks
  - It may need more chunks if some of its chunks are used by other starving processors
  - Assume that we steal chunks only from overloaded processors



- Let N_k be the number of processors with exactly k chunks:

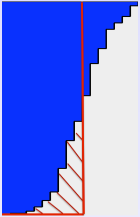
$$\begin{aligned} N_k &= p \times \binom{n}{k} (1/p)^k (1 - 1/p)^{n-k} \\ &= e^{-n/p} (n/p)^k / k! \text{ when } k \ll n, p \end{aligned}$$

- Then, the communication volume is given by:

$$V = \sum_{k<n/p} (n/p - k) N_k = p e^{-n/p} \frac{(n/p)^{n/p+1}}{(n/p)!} \approx \sqrt{\frac{np}{2\pi}}$$

Lower bound on communications without replication

- Consider n balls placed in p bins (initial distribution with $r = 1$)
- A processor with $k < n/p$ chunks will have to receive at least $k - n/p$ chunks
  - It may need more chunks if some of its chunks are used by other starving processors
  - Assume that we steal chunks only from overloaded processors



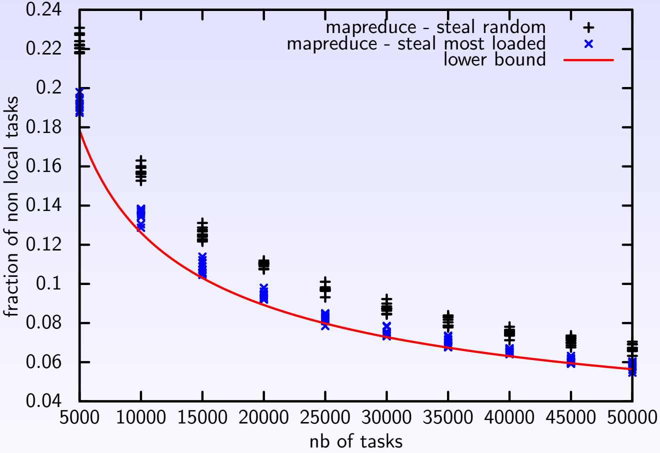
- Let N_k be the number of processors with exactly k chunks:

$$\begin{aligned} N_k &= p \times \binom{n}{k} (1/p)^k (1 - 1/p)^{n-k} \\ &= e^{-n/p} (n/p)^k / k! \text{ when } k \ll n, p \end{aligned}$$

- Then, the communication volume is given by:

$$V = \sum_{k<n/p} (n/p - k) N_k = p e^{-n/p} \frac{(n/p)^{n/p+1}}{(n/p)!} \approx \sqrt{\frac{np}{2\pi}}$$

Lower bound without replication – simulations



Outline

Estimate load imbalance without communication

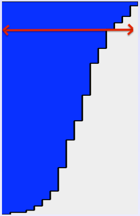
- Previous section: estimate communication done by MapReduce to mitigate load imbalance
- But load imbalance might be more desirable that large data exchange
- Objective: estimate the makespan without communication

Model:

- Similar data distribution (n chunks on p processors, r replicas of each chunk)
- Allocation mechanism:
  - When a processor is idle, allocate a task on local chunk (if any)
  - Invalidate other replicas of the chosen chunk
- Uniform or slightly heterogeneous task durations ($w_i \leq \frac{\sum w_i}{n \log n}$), unknown beforehand

Makespan without replication

- Without replication: each chunk is on a single processor
- Processor execution time = sum of chunk sizes
- Similar to the maximum load of a bin in balls-in-bins:



  - With identical tasks, when $n/\text{polylog}(n) \leq p \leq n \log n$:

$$M \sim \frac{\log p}{\log\left(\frac{p \log p}{n}\right)} \quad w.h.p.$$

  - For other cases, see [Raab & Steeger 2013], [Berenbrick 2008]

Makespan with replication – intuition

We build an analogy between:

- Modified MapReduce with replication r
- Balls-In-Bins distribution with r choices:
  - For each ball, select r bins at random
  - Allocate ball to the least loaded bin among them

In the following:

- Slightly different starting times of processors: t_i
- Initial load of bins i: t_i (same tie break at time 0)
- Set of random choices: $C_i = \{i_1, \dots, i_r\}$ used by both processes

Makespan with replication – analogy

Modified MapReduce:

- For each task:
  Place a copy of task T_i on processors with index in $C_i = \{i_1, \ldots, i_r\}$
- When a processor k becomes idle:
  Execute the available tasks with smaller index (if any)

NB: allocation with replication, load-balancing at runtime

Balls-In-Bins with multiple choices:

- For each ball: Place ball i in the least loaded bin with index in $C_i = \{i_1, \ldots, i_r\}$

NB: load-balancing during the allocation

Theorem.

The makespan of Modified MapReduce is equal to the maximum load of Balls-In-Bins with multiple choice

Makespan with replication – analogy

Modified MapReduce:

- For each task:
  Place a copy of task T_i on processors with index in
  $C_i = \{i_1, \dots, i_r\}$
- When a processor k becomes idle:
  Execute the available tasks with smaller index (if any)

NB: allocation with replication, load-balancing at runtime

Balls-In-Bins with multiple choices:

- For each ball: Place ball i in the least loaded bin with index in $C_i = \{i_1, \dots, i_r\}$

NB: load-balancing during the allocation

Theorem.

The makespan of Modified MapReduce is equal to the maximum load of Balls-In-Bins with multiple choice

Makespan with replication – analogy

Modified MapReduce:

- For each task:
  Place a copy of task T_i on processors with index in
  $C_i = \{i_1, \dots, i_r\}$
- When a processor k becomes idle:
  Execute the available tasks with smaller index (if any)

NB: allocation with replication, load-balancing at runtime

Balls-In-Bins with multiple choices:

- For each ball: Place ball i in the least loaded bin with index in
  $C_i = \{i_1, \dots, i_r\}$

NB: load-balancing during the allocation

Theorem.

The makespan of Modified MapReduce is equal to the maximum load of Balls-In-Bins with multiple choice

Makespan with replication – proof

Lemma.

Let $proc(i)$ be the processor executing task i and $bin(i)$ the bin containing ball i, then $proc(i) = bin(i)$.

Proof by induction:

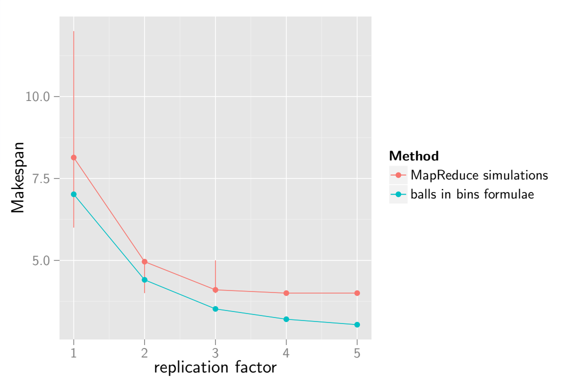
- First ball put on bin $k \in C_1$ with smallest t_k, same for first task
- Consider task/ball i:
  - When T_i starts, only tasks with smaller indexes already processed by processors of C_i
  - Completion time of such a processor k before starting T_i:
    $$C_k = \sum_{j<i, proc(j)=k} size(j))$$
  - Ball i considered after balls $1, \dots, i-1$, load of bin k at that time:
    $$L_k = \sum_{j<i, bin(j)=k} size(i)$$
  - Ball i put in bin $k \in C_i$ with smallest L_k
  - By induction, $C_k = L_k$

Makespan with replication – results

Maximum load using multiple choice ($r \geq 2$) at most:

$$\frac{n}{p} + \frac{\log \log n}{\log r} + \Theta(1) \text{ w.h.p.} \quad \text{[Berenbrick et al. 2000]}$$

Simulations with 200 processors and 400 (identical) tasks:



Outline

Conclusion

- Data locality analysis of the Map phase of MapReduce
- Task allocation mechanism with initial data placement: very simple and general
  - Volume of communication:
    - Simple formula accurate for $r \geq 2$ (missing formal proof)
    - Lower bound for $r = 1$
      $=$ exact volume for a variant of MapReduce (steal the most loaded)
  - Load imbalance without communication:
    - Makespan $=$ maximum load for multiple-choice balls-in-bins
- Key parameter: replication (both for comm. and makespan)
- Analogy: replication vs. "power of 2 choices" for balls-in-bins
- NB: cost of replication: large communication volume prior to the computation (best-effort, possibly for many computations)

Perspectives

Extensions: Better estimate the communication volume with replication:

- Use analogy with balls-into-bins with r choices (at most $2p$ holes [Berenbrick et al. 2000])?
- Use potential function (cf. [Tchiboukdjian et al. 2012])?
- Heterogeneous task durations

Long-term perspectives:

- More complex data dependences (2D, tasks sharing files)