On Data Skewness, Stragglers, and MapReduce Progress Indicators

Emilio Coppa and Irene Finocchi

Sapienza Università di Roma

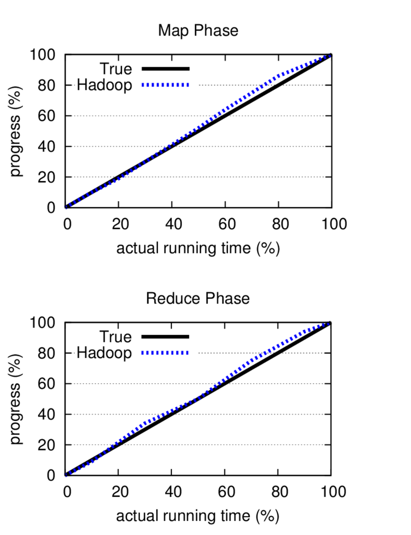
Progress analysis helps users understand the program execution and can shed light on abnormal behaviors:

- remaining time?
- load unbalancing?
- slow/stalled computations?
- algorithmic inefficiencies?

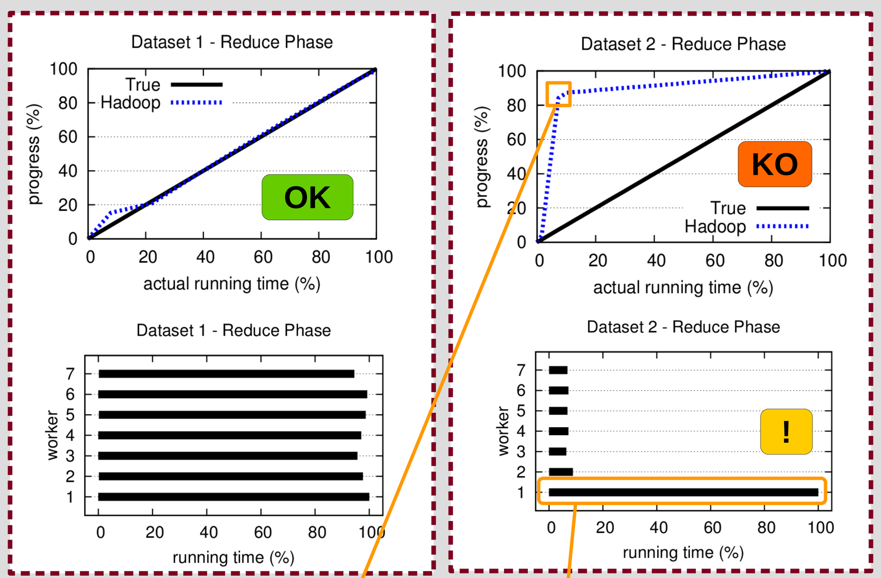
Example: Apache Hadoop progress indicator

Benchmark: WordCount Map: ~20mins
Dataset: Wikipedia dump Reduce: ~10mins

```
[...]
11:30:01: Running job: job_201503151102_0002
11:30:02: map 0% reduce 0%
11:32:26: map 10% reduce 0%
11:34:48: map 19% reduce 0%
11:36:32: map 30% reduce 0%
11:38:26: map 41% reduce 0%
11:40:08: map 52% reduce 0%
11:42:39: map 64% reduce 0%
11:44:14: map 75% reduce 0%
11:46:25: map 86% reduce 0%
11:50:01: map 100% reduce 0%
11:51:01: map 100% reduce 10%
11:52:57: map 100% reduce 34%
11:55:02: map 100% reduce 52%
11:57:07: map 100% reduce 73%
11:58:49: map 100% reduce 91%
11:59:54: map 100% reduce 100%
13:59:59: Job complete: job_201503151102_0002
[...]
```



Another example:
MatMul - a sparse matrix multiplication library



After ~4 mins:
- true progress: 10%
- true remaining time: 36 mins
- estimated progress: 85%
- estimated remaining time: ~1 min

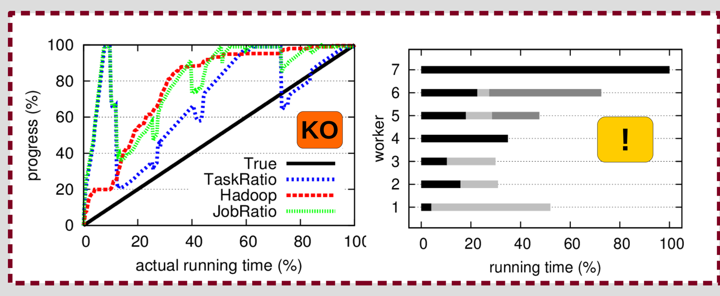
Straggler: a task which takes much longer to complete than the other ones

Same benchmark, different datasets:
very different progress prediction accuracy. Why?

Skewness and stragglers in MapReduce

- *Partitioning skewness*: keys unfairly partitioned among tasks
- *Shuffle data skewness*: few key groups much larger than others
- *Computational skewness*: data skewness + superlinear reduce functions

State-of-art progress indicators do not deal with computational skewness



Linear progress assumption: running time depends linearly on the input size

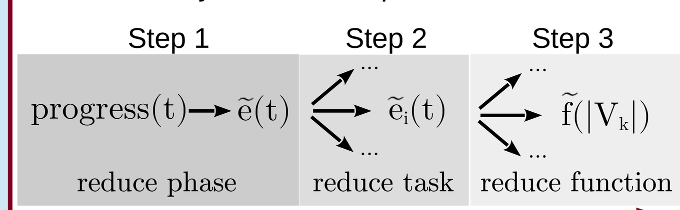
Computational skewness common in practice: e.g., computing clustering coefficients in social networks (power-law degree distribution)

Our contribution

Design and implementation of **NearestFit**, a novel progress indicator especially well-suited for long-running applications.

Our approach:
- no linear progress assumption
- predictions based on dynamically collected fine-grained profile data
- exploit machine learning techniques to predict remaining running time
- efficient implementation based on data streaming algorithms

Bird's eye view of our prediction model



(Step 1) Reduce progress at time t

$$progress(t) = \frac{t - t_{start}}{\tilde{e}(t) - t_{start}} \times 100$$

estimated end time; phase start time

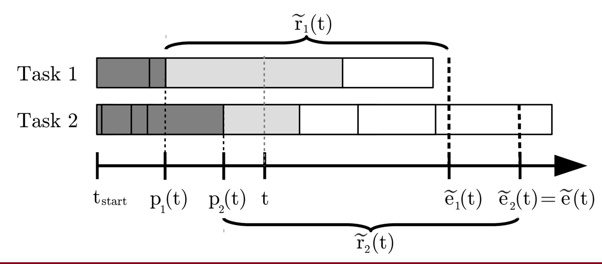
(Step 2) Estimated end time

$$\tilde{e}(t) = \max_{reduce\ tasks\ i} \tilde{e}_i(t)$$

estimated end time of task i

$$\tilde{e}_i(t) = p_i(t) + \tilde{r}_i(t)$$

time of last completed reduce execution in task i; estimated remaining time of task i



(Step 3) Remaining time of task i

$$r_i(t) = \sum_{\text{unprocessed key } k} f(k, V_k)$$

exact cost model for the running time of reduce functions: unknown in general

k: any unprocessed key assigned to task i
V_k: set of values associated with k

Our assumption: running time function of input size

$$|V_{k_1}| \approx |V_{k_2}| \Rightarrow f(k_1, V_{k_1}) \approx f(k_2, V_{k_2})$$

Then:

$$\tilde{r}_i(t) = \sum_{\text{unprocessed key } k} \tilde{f}(|V_k|)$$

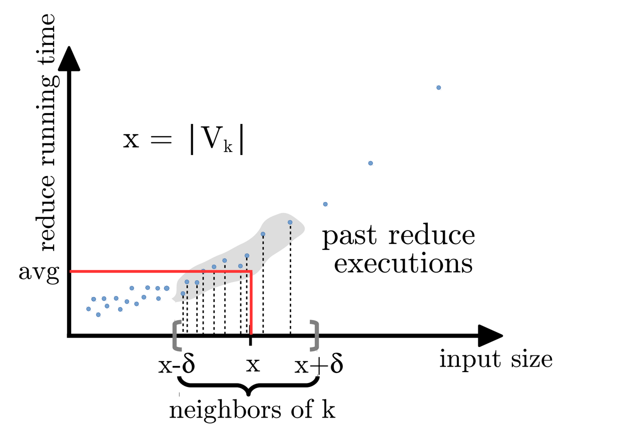
approximate cost model for the running time of reduce functions

How to predict reduce running time for key group (k, V_k)?

Two complementary techniques

Technique 1: δ-nearest neighbor regression

$\tilde{f}(|V_k|)$ = average of the running times observed in the δ-neighborhood of k

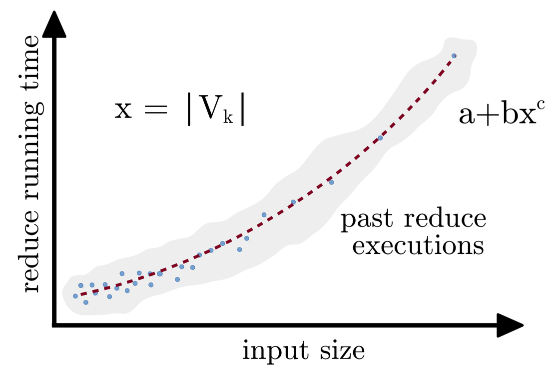


Rather accurate, but δ-neighborhood could be empty (especially for stragglers)

Technique 2: curve fitting

Find a mathematical model (parameters a, b, and c):

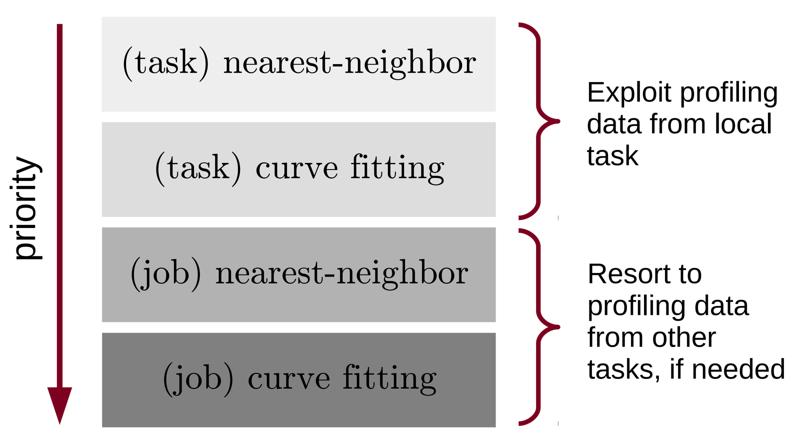
$$\tilde{f}(|V_k|) = a + b \cdot |V_k|^c$$



Potentially always applicable, but hard to tune in practice (unstable, noisy profiles)

Combining nearest neighbors and curve fitting

Combination of the two techniques overcomes their drawbacks while retaining their advantages:



Key insights:
- nearest neighbor, if applicable, more accurate than curve fitting
- prioritize task-local profiling data: VMs can exhibit vastly different performance even on homogeneous clusters
- if not enough profiling data available from task i, resort to profiles from other tasks (job-level profiles)

Implementation ingredients

(1) Characterization of reduce task inputs

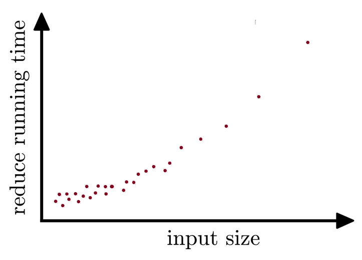
Which is the distribution of key group sizes for a given task?

Obtained by profiling map tasks

(2) Information about past executions of reduce functions:

Which are the input sizes and running times of terminated executions?

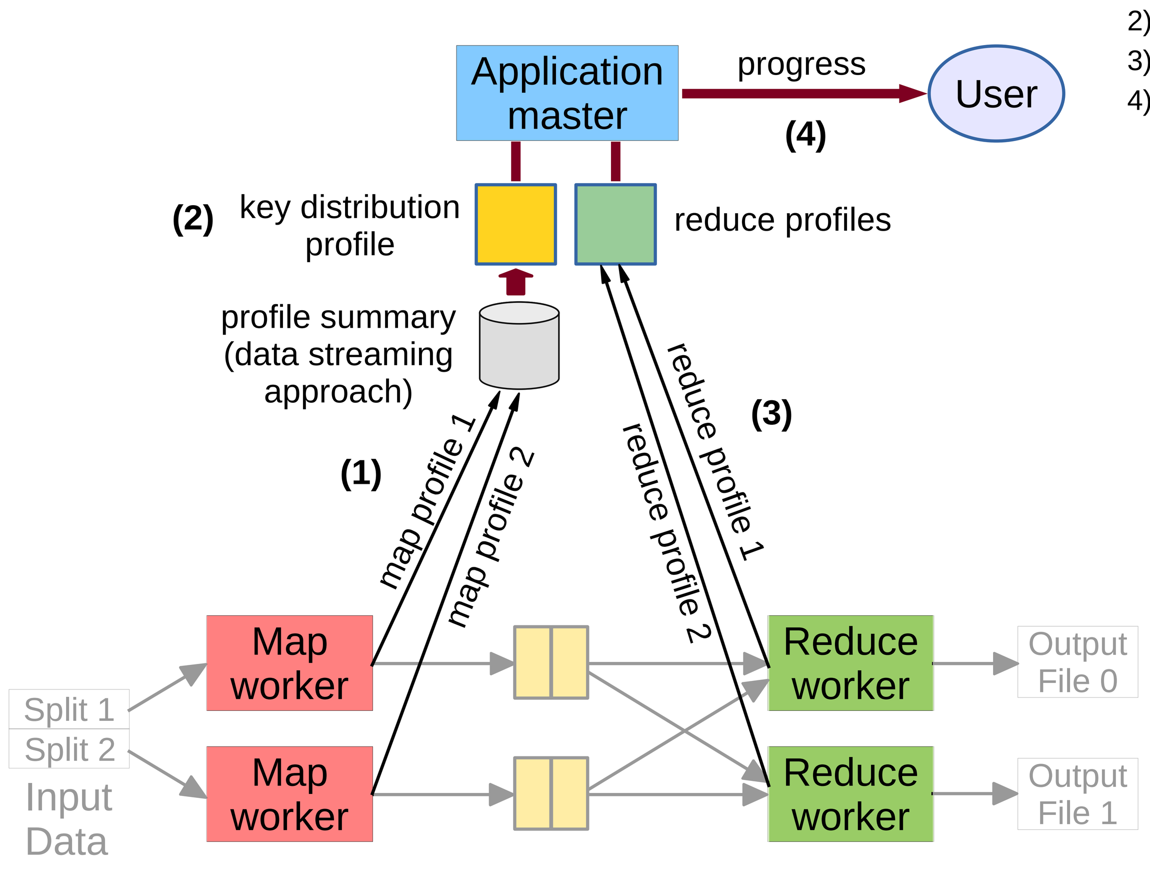
Obtained by profiling reduce tasks



Massive amounts of fine-grained profile data: non negligible time and space overheads!

NearestFit exploits space and time efficient data streaming algorithms to approximate some of the quantities required by the theoretical model

An operational view of NearestFit



1) Map workers send *map profiles* to application master
2) Using map profiles, application master builds a *key distribution profile*
3) Reduce workers periodically send *reduce profiles* to application master
4) Using key distribution and reduce profiles, application master estimates progress

Map profiles:
top-k keys with largest sets of values
+
cumulative summary of remaining keys and their sizes

Key distribution profile:
approximate top-k keys with largest set of values among all reduce tasks
+
cumulative summary of remaining keys and their sizes

Reduce profiles:
Running times and key group sizes of past executions of the reduce function

Implemented on top of Hadoop 2.6.0

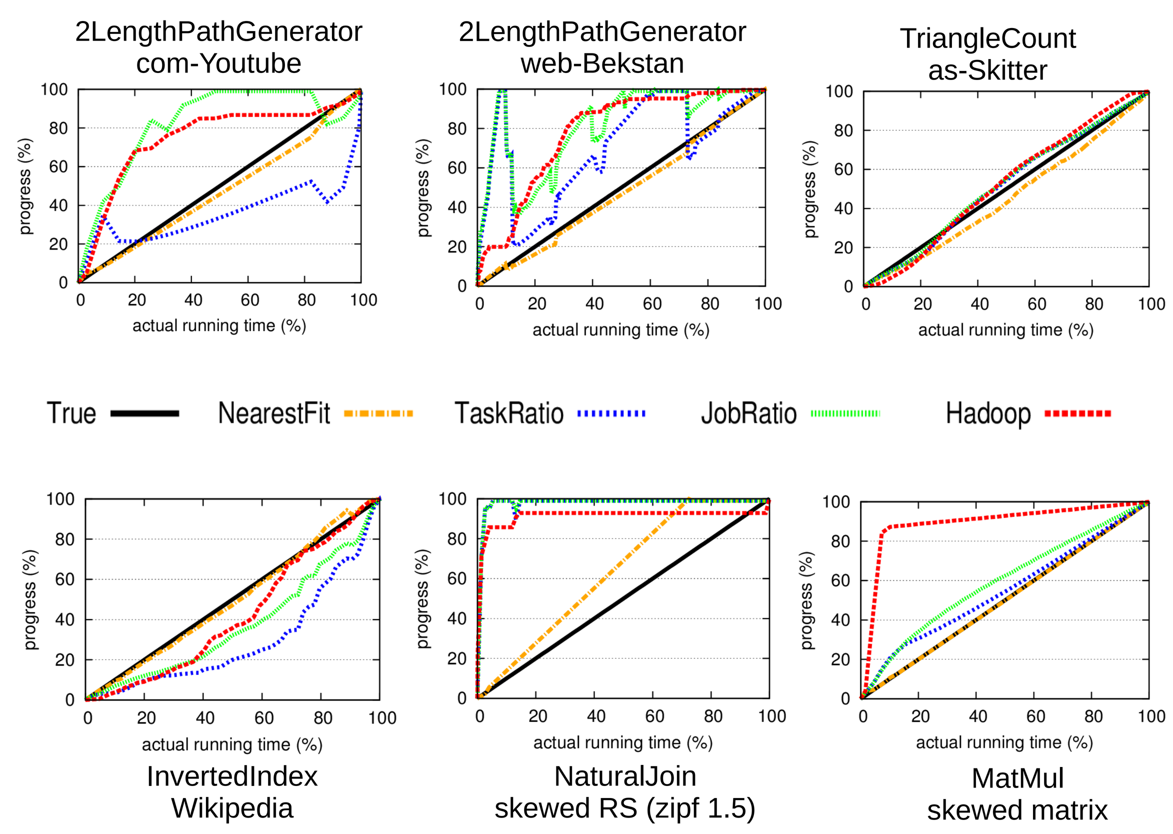
Experimental results

Applications: text processing (WordCount, InvertedIndex), graph computations (2LengthPathGenerator, TriangleCount), numerical analysis (MatrixMultiplication), database processing (NaturalJoin).

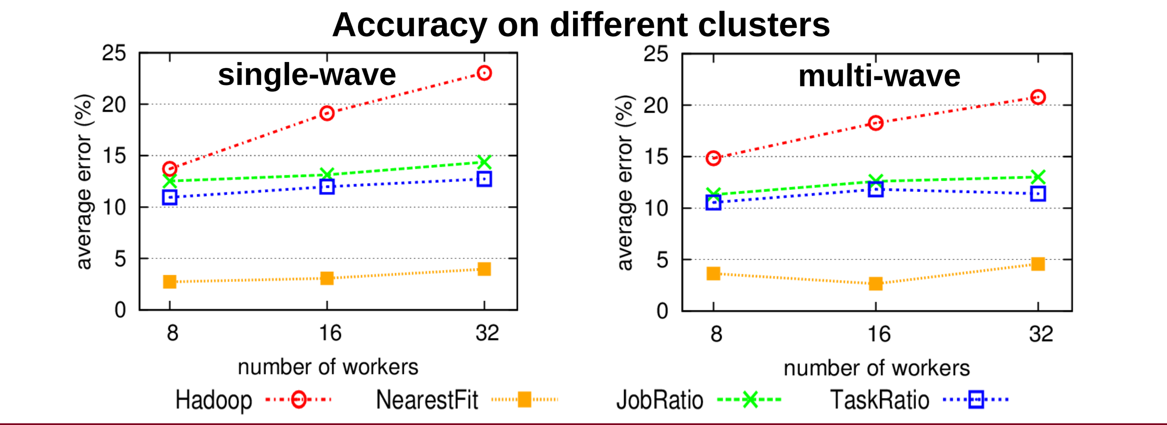
Datasets: Wikipedia dump, 6 social networks (SNAP project), 2 sparse matrices (uniform/skewed value distribution), and 5 skewed relations (zipf distribution).

Platform: 8/16/32 m1.xlarge instances from Amazon Web Services

Accuracy: NearestFit vs state-of-art indicators



Accuracy on different clusters



Space and time overhead of NearestFit