On Data Skewness, Stragglers, and MapReduce Progress Indicators

Emilio Coppa and Irene Finocchi

Progress analysis helps users understand the program execution and can shed light on abnormal behaviors:
- remaining time?
- slow/stalled computations?
- load imbalance?
- algorithmic inefficiencies?

Another example: MetMat - a sparse matrix multiplication library

Skewness and stragglers in MapReduce
- Partitioning skewness: keys unfairly partitioned among tasks
- Shuffle 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 program assumption: running time depends linearly on the input size

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

- no linear progress assumption
- can be based on dynamically collected fine-grained profile data
- exploits machine learning techniques to predict remaining running time

Efficient implementation based on data streaming algorithms

Wolff's eye view of our prediction model

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

Two-complementary techniques

Combining nearest neighbors and curve fitting

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

1. Nearest neighbor
2. Curve fitting

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

- Nearest neighbor
- Curve fitting

Nearest neighbor is not always applicable, but hard to tune in practice (unstable, noisy profiles)

Nearest neighbor is not always applicable, but hard to tune in practice (unstable, noisy profiles)

Implementation ingredients

1. Map workers send map profile to application master
2. 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

Experimental results

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

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

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

Accuracy: NearestFit vs state-of-art indicators

Space and time overhead of NearestFit

An operational view of NearestFit

1. Map workers send map profile 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 profile:
- top-k keys with largest set of values
- cumulative summary of remaining keys and their sizes

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

Implemented on top of Hadoop 2.6.0

On Data Skewness, Stragglers, and MapReduce Progress Indicators

Emilio Coppa and Irene Finocchi

Progress analysis helps users understand the program execution and can shed light on abnormal behaviors:
- remaining time?
- slow/stalled computations?
- load imbalance?
- algorithmic inefficiencies?

Another example: MetMat - a sparse matrix multiplication library

Skewness and stragglers in MapReduce
- Partitioning skewness: keys unfairly partitioned among tasks
- Shuffle 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 program assumption: running time depends linearly on the input size

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

- no linear progress assumption
- can be based on dynamically collected fine-grained profile data
- exploits machine learning techniques to predict remaining running time

Efficient implementation based on data streaming algorithms

Wolff's eye view of our prediction model

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

Two-complementary techniques

Combining nearest neighbors and curve fitting

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

1. Nearest neighbor
2. Curve fitting

Nearest neighbor is not always applicable, but hard to tune in practice (unstable, noisy profiles)

Nearest neighbor is not always applicable, but hard to tune in practice (unstable, noisy profiles)

Implementation ingredients

1. Map workers send map profile 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

Experimental results

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

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

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

Accuracy: NearestFit vs state-of-art indicators

Space and time overhead of NearestFit

An operational view of NearestFit

1. Map workers send map profile 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 profile:
- top-k keys with largest set of values
- cumulative summary of remaining keys and their sizes

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

Implemented on top of Hadoop 2.6.0