On the Performance of MapReduce: A Stochastic Approach

Sarker Tanzir Ahmed and Dmitri Loguinov

Internet Research Lab
Department of Computer Science and Engineering
Texas A&M University

October 28, 2014
Agenda

- Introduction
- Background
- Disk I/O
- Merge Overhead
- Runtime
Introduction

• MapReduce is a popular programming model for cluster computing, big data processing
• Resource constraints and data properties dictate MapReduce performance
  - Specifically, RAM size and distribution of key frequency impact both the run-time and volume of disk I/O
• Existing literature is missing an accurate performance model for external-memory sorting/merging
• Common to assume a linear relationship between input size and processing overhead
  - This includes disk spill, number of comparison in sort/merge
• Open questions:
  – Is this dependency indeed linear with some constant factor converting input size to various metrics of interest?
  – Does the constant stay the same in the full design space?
  – Is the constant easy to obtain/estimate?

• **Our objective**: analyze a shared-memory MapReduce system with a single host for all computation
  – Multiple CPU cores provide parallelism
  – Data distribution is only through disks, not network

• Due to limited space, we analyze only the *external merge-sort* as the underlying algorithm
Agenda

• Introduction
• Background
• Disk I/O
• Merge Overhead
• Runtime
Background

MapReduce platform

Read input

(key, value) pairs

Sort

sorted runs on disk

Merge

User functions

Map

Reduce

Reduce

output
Agenda

• Introduction
• Background
• Disk I/O
• Merge Overhead
• Conclusion
MapReduce Disk I/O

- Input is a stream of length $T$
  - Entries are key-value pairs, each $K+D$ bytes
  - At time step $t$, one pair is processed by MapReduce
  - Keys belong to a finite set $V$ of size $n$
  - Each key $v$ is repeated $I(v)$ times

- Disk I/O consists of:
  - Input with $T$ pairs (some duplicate)
  - Output with $n$ unique pairs
  - Sorted runs of size $L$

- Total disk overhead is $W = (K+D)(T + n + 2L)$
  - Our first goal is to derive $L$
MapReduce Disk I/O (2)

- Suppose RAM can hold \( m \) key-value pairs
- Let \( S_t \) denote the seen set at time \( t \) and \( \epsilon_t = t/T \) be the fraction of input processed by \( t \)
  - Then, \( k = \lceil T/m \rceil \) is the number of sorted runs, where each contains \( E[|S_m|] \) pairs on average

- **Theorem 1**: The expected size of the seen set at \( t \) is:
  \[
  E[|S_t|] = n - nE[(1 - \epsilon_t)^I] 
  \]

- **Theorem 2**: Disk spill \( L \) of a merge-sort MapReduce is:
  \[
  L = nk(K + D) \left( 1 - E[(1 - \epsilon_m)^I] \right) 
  \]

- Total I/O overhead is thus:
  \[
  W = n(K + D) \left\{ E[I] + 1 + 2k \left( 1 - E[(1 - \epsilon_m)^I] \right) \right\} 
  \]
MapReduce Disk I/O (3)

- Quite a complex function of $m$ and $I$
- Verification on real graphs
  - IRLbot host graph (640M nodes, 6.8B edges, 55 GB)
  - WebBase web graph (667M nodes, 4.2B edges, 33 GB)
Agenda

• Introduction
• Background
• Disk I/O
• Merge Overhead
• Runtime
MapReduce Merge Overhead

• Selection tree for merging sorted runs, where each internal node
  - Executes binary comparison
  - Applies reduce operation

• De-duplication makes upper nodes perform less work than lower

• Theorem 3: The number of comparisons in a binary selection tree with $k$ leaf nodes is:

$$C_k = nk \sum_{i=1}^{d} \frac{1}{2^{i-1}} \left(1 - E\left[(1 - \epsilon_{im})^I\right]\right)$$
Merge Rate Evaluation

• Comparison with naïve model (no de-duplication):
  \[ \hat{C}_k = n \log_2 k \cdot \left(1 - E[(1 - \epsilon_m)^T]\right) \]

• Comparison overhead and merge rate \( \gamma_m \) dictated by \( k \)
  - Higher \( k \) implies more de-duplication
Agenda

• Introduction
• Background
• Disk I/O
• Merge Overhead
• Runtime
MapReduce Runtime

- Total runtime is:

\[ T = \frac{W}{\rho} + \frac{T}{\delta} + \frac{L}{\gamma m}, \]

- Slight discrepancy due to disk seeking during merge.
Thank you!
Questions?