CSC 258/458

MapReduce: A Programming Model for Large-Scale Distributed Computation

University of Rochester
Department of Computer Science
Shantonu Hossain

April 18, 2011

Outline

- Motivation
- MapReduce Overview
- A Detailed Example
- Implications in Parallel/Distributed Computing
- Available Implementations
- Usage Examples
- Performance
- Summary

How big is www?

- http://www.worldwidewebsize.com/
- ~35 billion pages
  - 35billion X 20KB = 700 tera-bytes of data
- Average read rate 60MB/s
  - 270 months of read all the data!

How to process the data?

- Most of the data processing are embarrassingly parallel tasks!
  - Distribute the task across cluster of commodity machines (e.g. MPI)
- Challenges
  - Co-ordination and communication among nodes
  - Handle fault-tolerance
  - Recover from failure
  - Debug
  - Difficult to write codes!

What do we need?

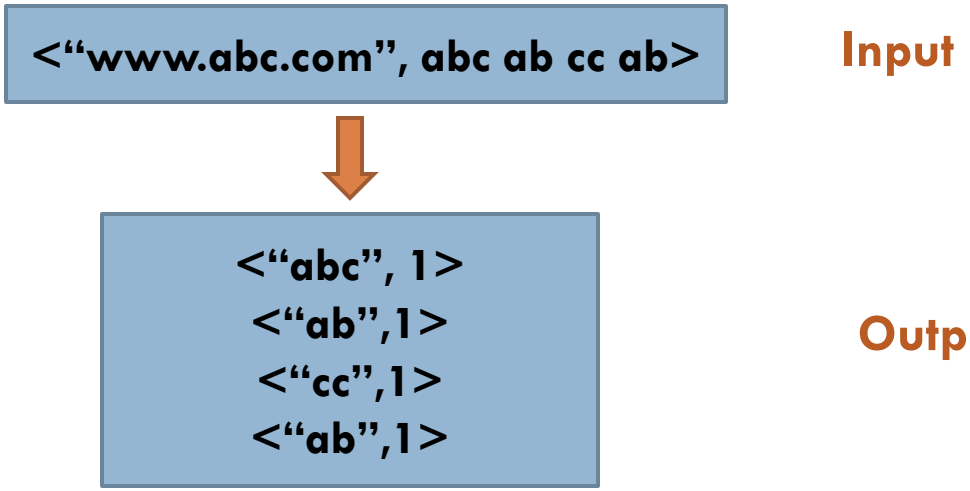
- A programming model that allows users to specify parallelism at high level
- An efficient run-time system that handles
  - Low-level mapping
  - Synchronization
  - Resource management
  - Fault tolerance and other issues
- Solution: MapReduce

MapReduce: Overview

- A programming model for large-scale data-parallel applications introduced by Google
- Aimed to process many tera-bytes of data on clusters with thousands of machines
- Programmers have to specify two primary methods
  - **Map:** processes input data and generates an intermediate key/value pairs
  - **Reduce:** aggregates and merges all intermediate values associated with the same key

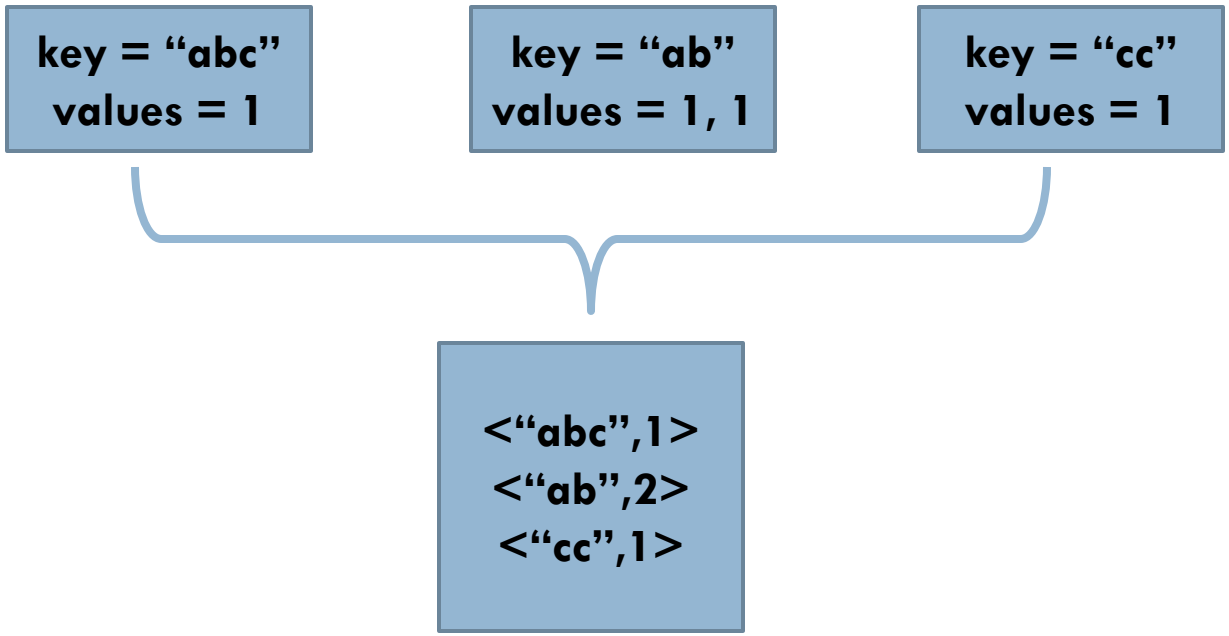
Example: Count words in web pages

- Input: files with one URL per record
- Output: count of occurrences of individual word
- Steps:
  1) Specify a Map function
     - Input: <key= webpage url, value=contents>
     - Output: <key=word, value=partialCount>*



2) Specify Reduce function : collects partial sum and compute total sum of each individual word

- Input: <key=word, value=partialCount*>
- Output: <key=word, value=totalCount>*



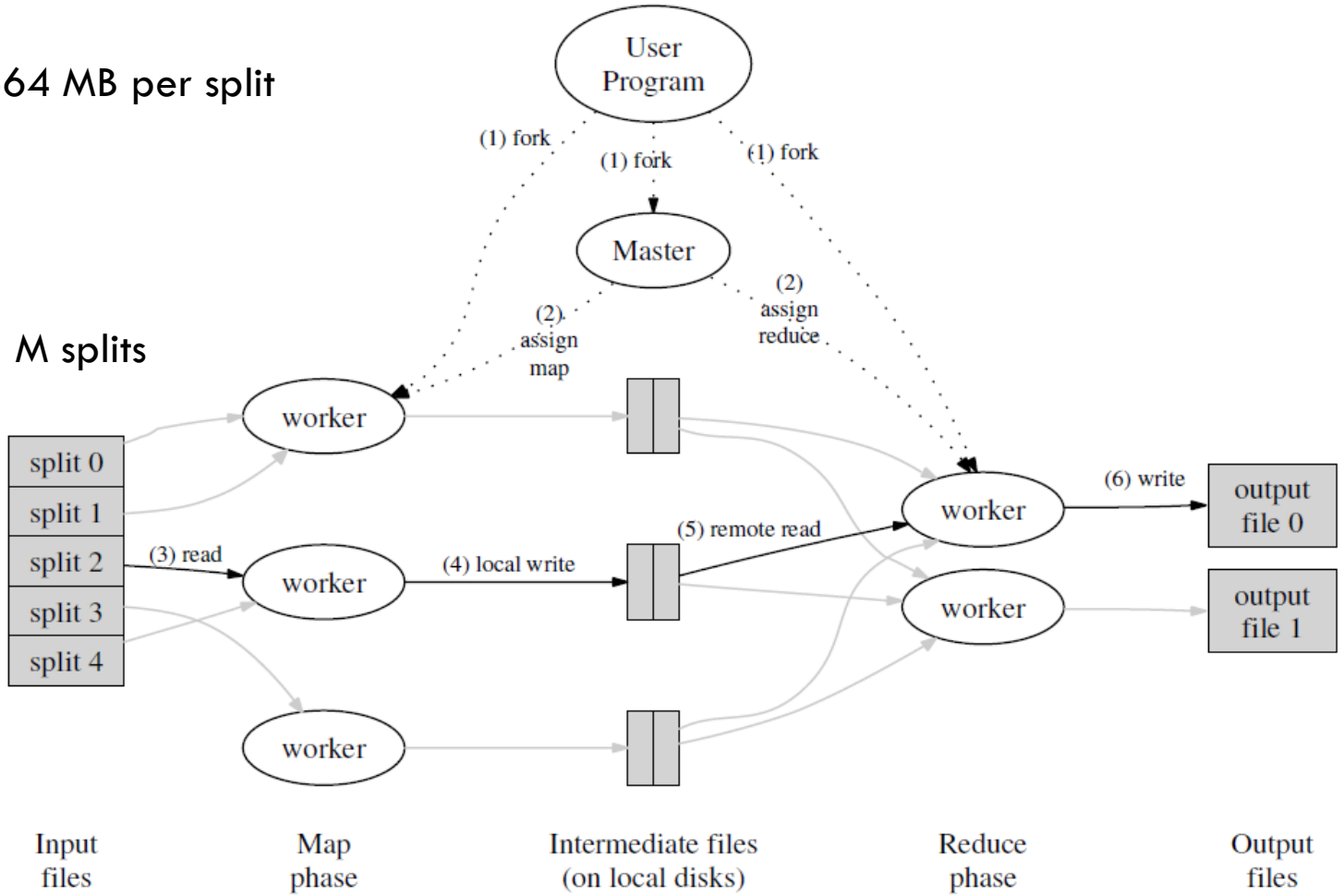
Example code: Count word

```
void map(String key, String value):
// key: webpage url
// value: webpage contents
for each word w in value:
  EmitIntermediate(w, "1");

void reduce(String key, Iterator partialCounts):
// key: a word
// partialCounts: a list of aggregated partial
  counts
int result = 0;
for each pc in partialCounts:
  result += ParseInt(pc);
Emit(AsString(result));
```

How MapReduce Works?

16MB-64 MB per split



Ref: J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, OSDI, 2004

Issues of Distributed Computing

- Fault tolerance
  - Worker failure
    - Master pings worker periodically
    - Keep tracks of states for each map reduce states
    - Reschedule failed workers
  - Master failure
    - Periodic check-pointing of master data structures
- Data Locality
  - GFS stores several copies of each input split on different machines
  - Master schedules tasks by taking location information into account

Key Features

- Simplicity of the model
  - Programmers specifies few simple methods that focuses on the functionality not on parallelism
  - Code is generic and portable across systems
- Scalability
  - Scales easily for large number of clusters with thousands of machines
- Applicability to a large variety of problems
  - Computation intensive

Implications on Parallel/Distributed Computing

- Allows programmers to write parallel codes easily and utilize large distributed systems by
  - Providing a new abstraction to write codes that are automatically parallelized.
  - By hiding all messy details of
    - Data distribution
    - Load balancing
    - Co-ordination and communication
- Applies to multi-core/multi-processor systems which are great candidates for data-parallel applications
  - Distributed clusters are replaced by large number of cores with shared memory system

Available Implementations: for Clusters

- MapReduce Hadoop
  - An open source implementation by Apache
  - Stores intermediate results of computation in local disks
  - Doesn't support configuring 'Map' tasks over multiple iterations
- CGL-MapReduce
  - A streaming based open source implementation by Indiana University at Bloomington
  - Intermediate results are directly transferred from 'Map' to 'Reduce' tasks.
  - Supports both single and iterative MapReduce computation

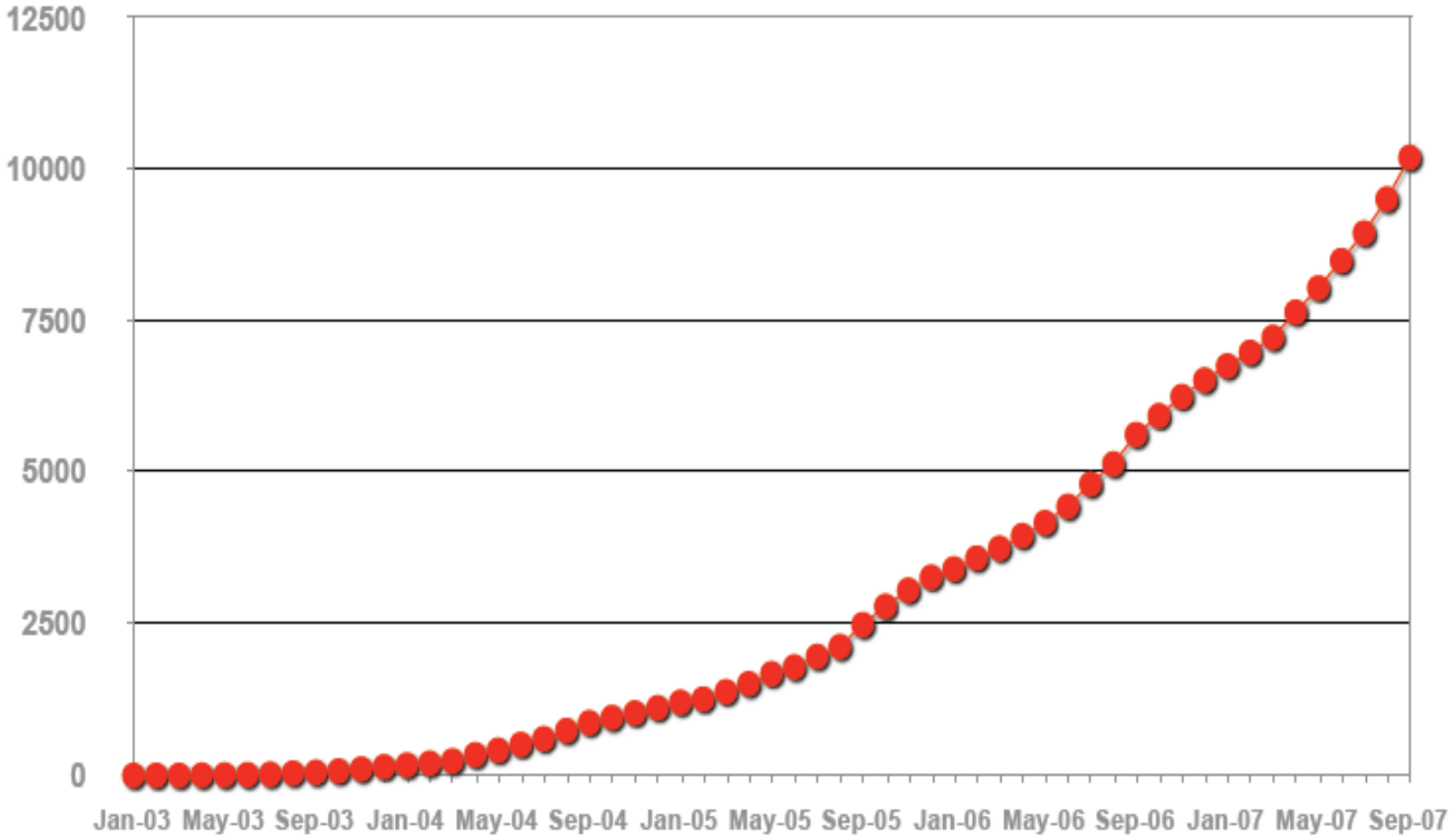
Available Implementations: for Multi-cores

- **Phoenix**
  - A MapReduce implementation in C/C++ for shared-memory systems by Stanford University
  - Stores intermediate key/value pairs in a matrix
    - Map puts result in the row of input split
    - Reduce processes an entire column at a time
- **Metis**
  - An in-memory MapReduce library in C optimized for multi-cores by MIT
  - Uses hash table with b-tree in each entry to store intermediate results
  - Involves several optimizations
    - Lock free scheme to schedule map and reduce work
    - immutable strings for fast key comparisons
    - Scalable Streamflow memory allocator

Usage Examples

- MapReduce has been used within Google for
  - completely regenerate Google's index of the world wide web
  - Clustering problems for Google News
  - Extraction of data used to produce reports of popular queries
  - Large scale graph computations
- Can be applied to wide range of applications such as
  - Distributed grep
  - Distributed sort
  - Document clustering
  - Inverted index
  - Large-scale machine learning algorithms

Usage Statics at Google



Ref: PACT 06' Keynote slides by Jeff Dean, Google, Inc.

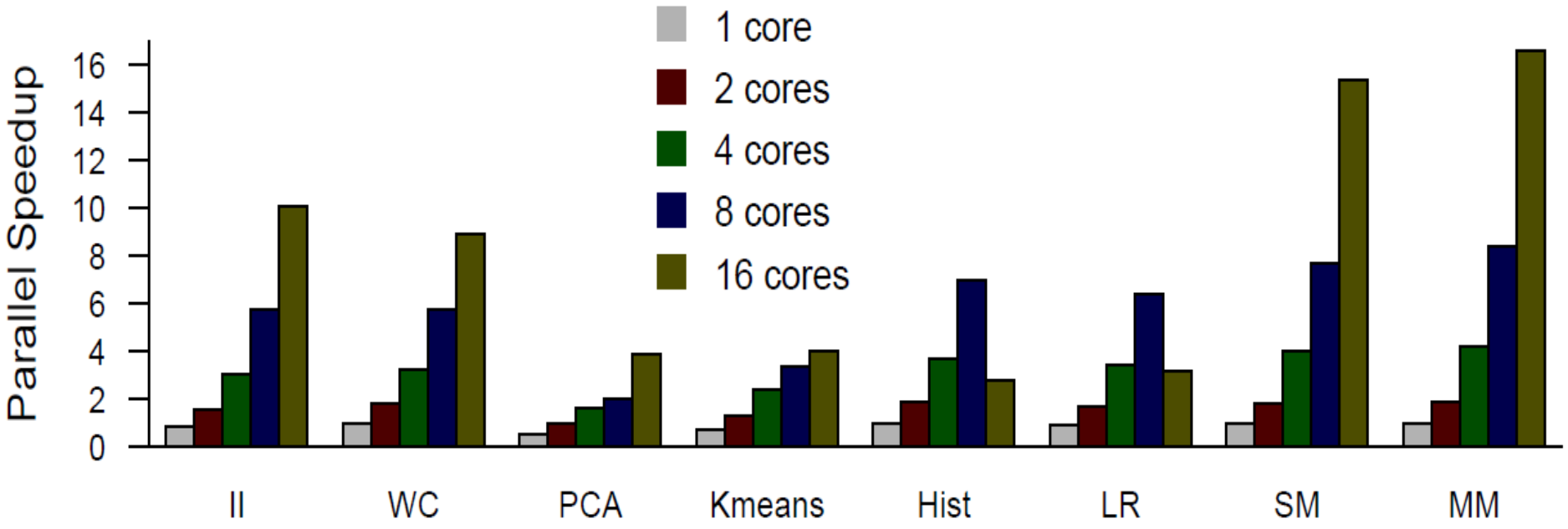
Candidate Workloads

- I/O intensive
  - Performance is limited by the I/O bandwidth
  - The overhead induced by the MapReduce implementations has negligible effect on the overall computation
  - Example: High Energy Physics data analysis, process remote sensing data
- Memory intensive
  - Performance is limited by memory access
  - Example: Successive Over Relaxation (SOR)
- CPU intensive
  - Performance is limited by the amount of computation
  - Example: K-means algorithm

Performance Evaluation

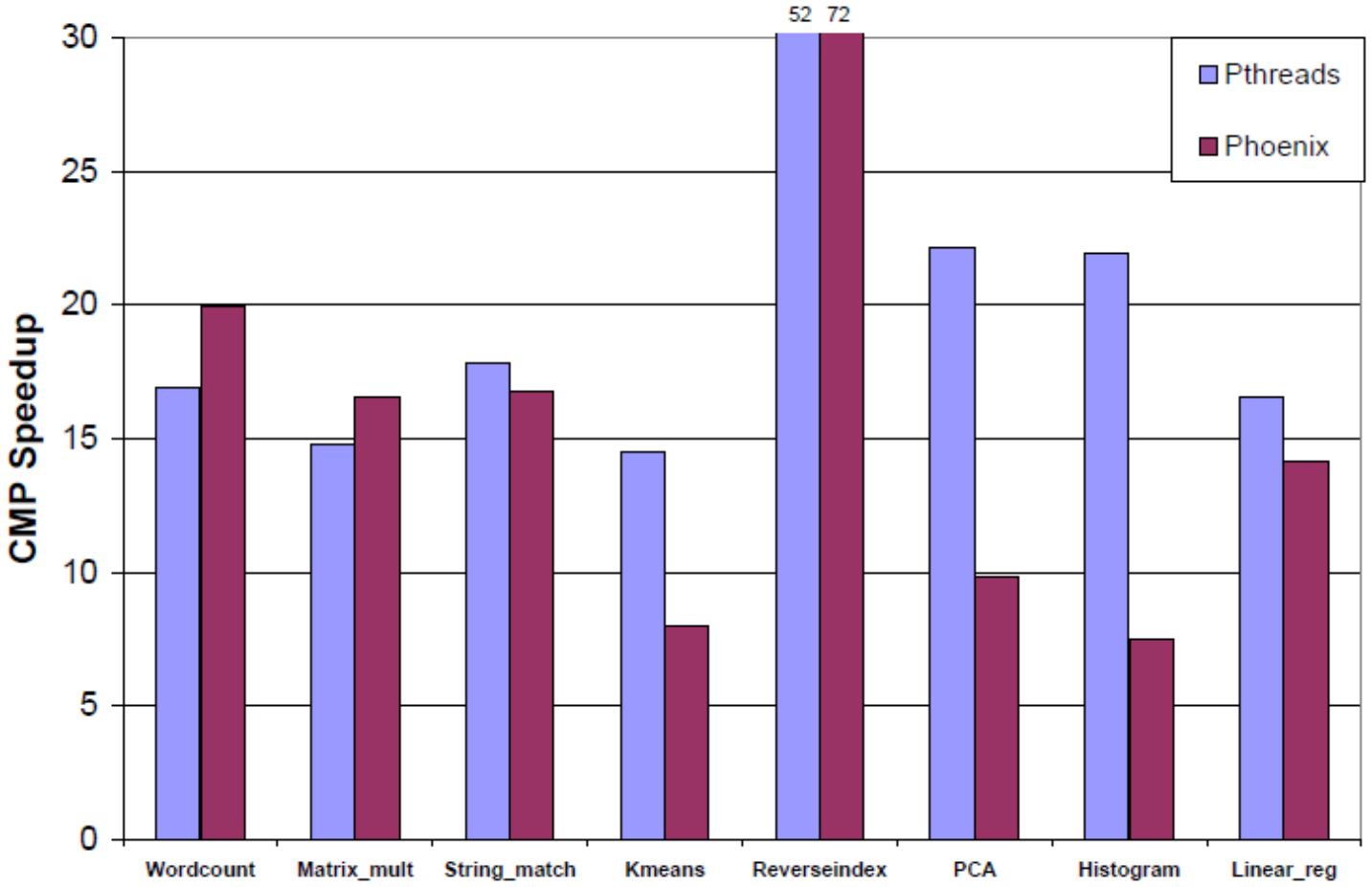
- Applications used
  - Word count (WC)
  - Matrix Multiply (MM)
  - Inverted index (II)
  - K-Means clustering(Kmeans)
  - String matching (SM)
  - PCA
  - Histogram (Hist)
  - Linear regression (LR)

Speedup (Metis)



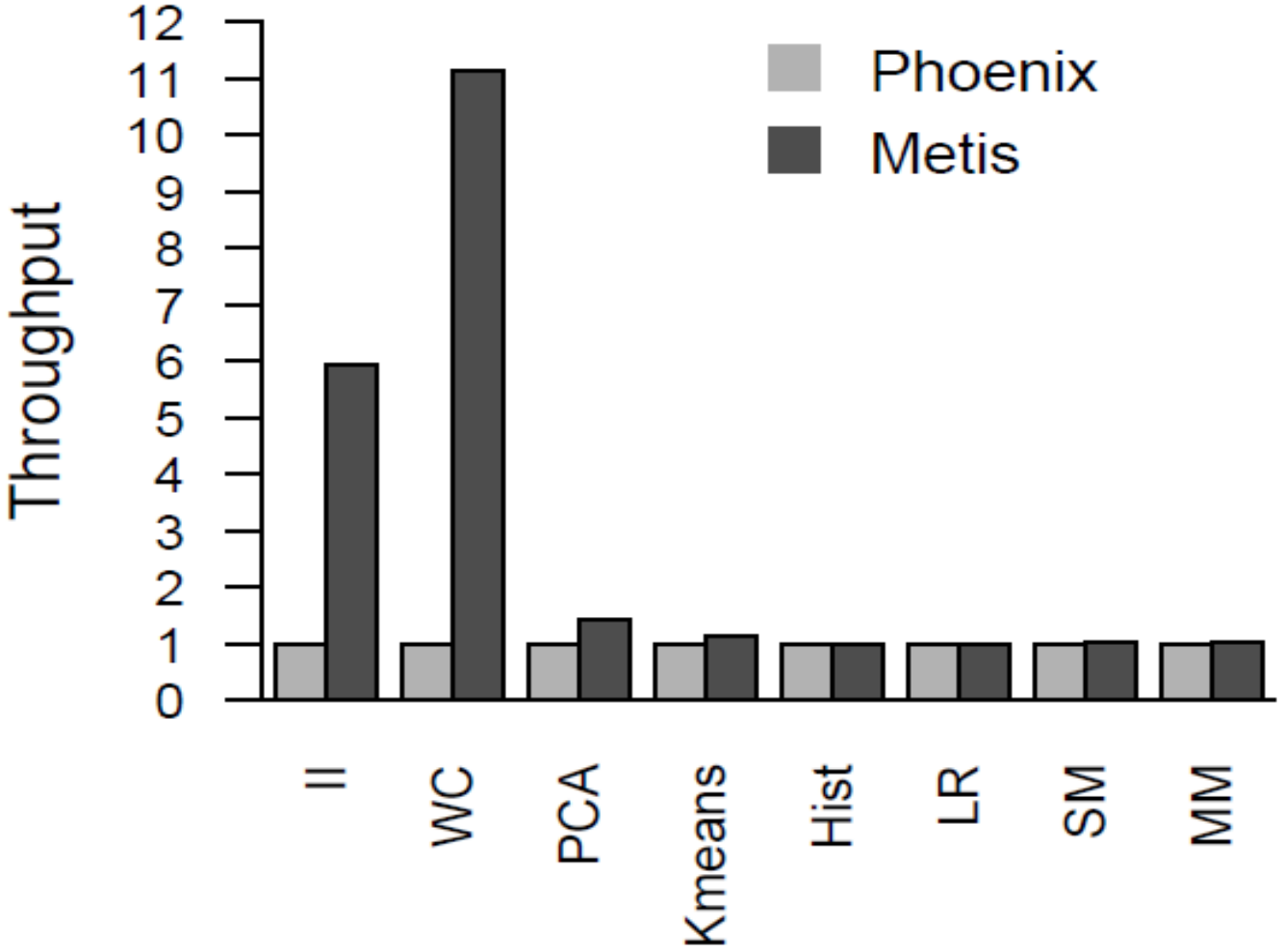
Ref: Y. Mao, R. Morris, and F. Kaashoek. Optimizing MapReduce for Multicore Architectures. Technical Report, MIT, 2010

Comparison to Pthreads (Phoenix)



Ref: C. Ranger , R. Raghuraman , A. Penmetsa , G. Bradski , C. Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems, HPCA, 2007.

Metis Vs Phoenix



Ref: Y. Mao, R. Morris, and F. Kaashoek. Optimizing MapReduce for Multicore Architectures. Technical Report, MIT, 2010

Summary

- MapReduce is a programming model for data-parallel applications
- Simplifies parallel programming
  - Programmers write code in functional style
  - The runtime library hides the burden of synchronization and coordination from programmers
- Many real-world large scale tasks can be expressed in MapReduce model
- Applicable for both distributed clusters and multi-core systems

References

- J. Dean and J. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In the Proc. of the 6th Symp. on Operating Systems Design and Implementation, Dec 2004.
- Wikipedia article on MapReduce, http://en.wikipedia.org/wiki/MapReduce
- C. Ranger , R. Raghuraman , A. Penmetsa , G. Bradski , C. Kozyrakis. Evaluating MapReduce for Multi-core and Multiprocessor Systems. In the Proc. of the 13th International Symposium on High-Performance Computer Architecture, Feb 2007.
- Y. Mao, R. Morris, and F. Kaashoek. Optimizing MapReduce for Multicore Architectures. Technical Report MIT-CSAIL-TR-2010-020, MIT, 2010.

- PACT 06' Keynote slides by Jeff Dean, Google, Inc.
- J. Ekanayake, S. Pallickara, and G. Fox. MapReduce for data intensive scientific analyses. In *IEEE Fourth International Conference on eScience, 2008*