Optimization Techniques for "Scaling Down" Hadoop on Multi-Core, Shared-Memory Systems

K. Ashwin Kumar, Jonathan Gluck, Amol Deshpande, Jimmy Lin

EDBT 2014

Motivation

Hadoop - Assumption

Data to be analyzed does not fit in single machine memory

This assumption needs to be re-examined!

- Single server today has more cores and large memory
  - Production clusters (at Microsoft and Yahoo) have median job input sizes under 14GB
  - 90% jobs on Facebook cluster have input sizes under 100GB[1]
- Increasing trend toward data mining and machine learning
  - More refined and smaller datasets

[1] Rowstron, Antony, et al. "Nobody ever got fired for using Hadoop on a cluster." HotCDP. 2012.

Motivation

Problem

- How can we achieve efficient in-memory data analytics on single machine?
- Do we go back to multi-threaded programming?
  - Coding and optimizations depend on applications

Our Solution

- "Scale down" Hadoop to run on shared-memory machines
- HONE: 'H'adoop 'ONE'
- API compatibility with Hadoop

Why not Hadoop Pseudo-Distributed Mode (PDM)?

Different types of overheads

- Multi-process overheads
- IO overheads
- Framework overhead
- Hadoop PDM on RAM disk shows negligible benefit

Related Work

System	Pros	Cons
Phoenix[2]	First multi-core map reduce implementation, fast, written in C	Not Hadoop compatible
Metis[3]	Claims improvement over Phoenix	Not Hadoop compatible
M3R[4]	Hadoop compatible, in-memory cluster analytics	Optimized for scale-out, not evaluated for single machine
Spark[5]	In-memory cluster analytics	Not Hadoop compatible, optimized for scale-out

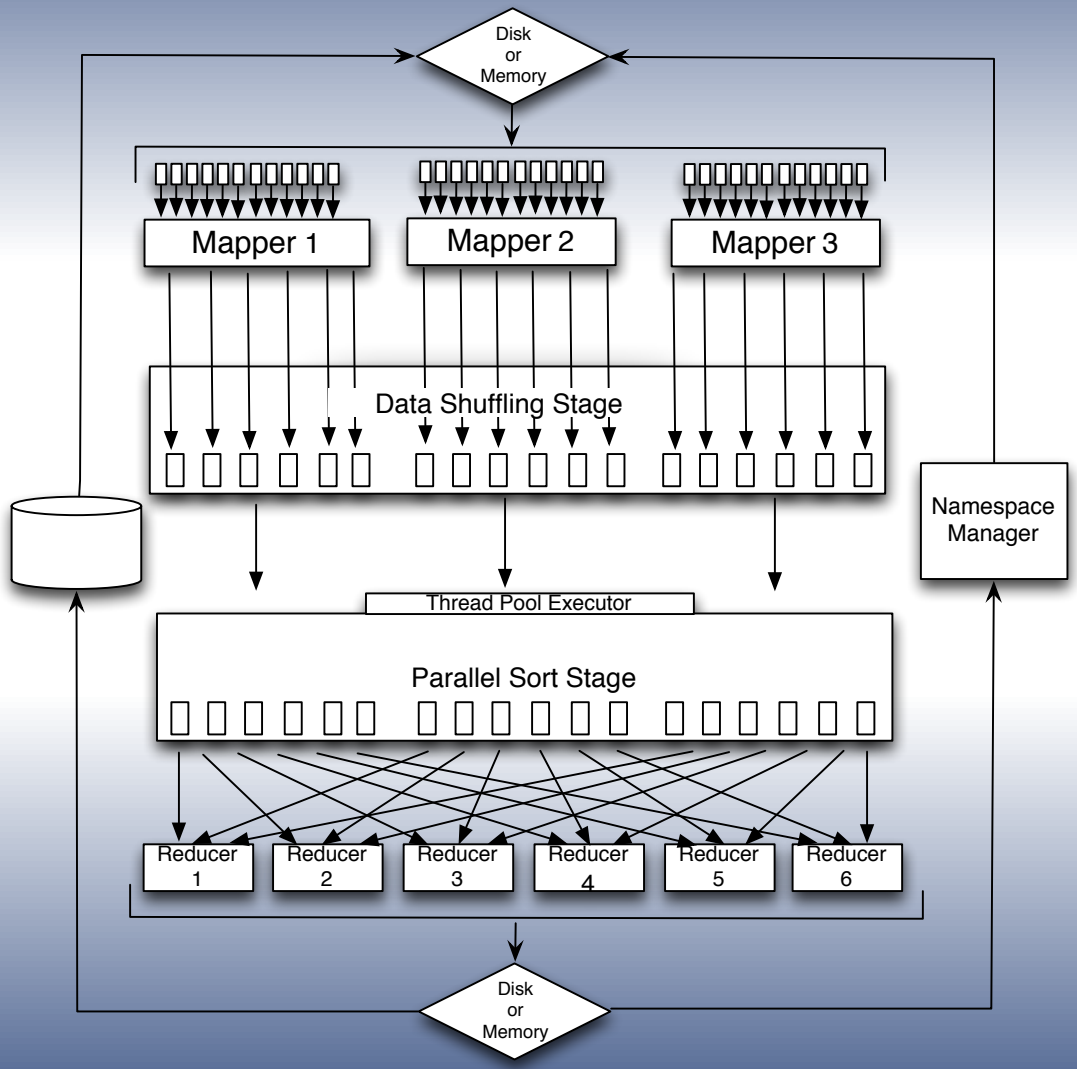
[2] Ranger, Colby, et al. "Evaluating mapreduce for multi-core and multiprocessor systems." HPCA 2007.

[3] Mao, Yandong, et al. "Optimizing MapReduce for multicore architectures." CSAIL, MIT, Tech. Rep. 2010.

[4] Shinnar, Avraham, et al. "M3R: increased performance for in-memory Hadoop jobs." VLDB 2012.

[5] Zaharia, Matei, et al. "Spark: cluster computing with working sets." HotCloud 2010.

Hone: System Architecture



Technical Contributions

1. Memory Overheads
2. Data Shuffling: Techniques and Tradeoffs
3. Dealing with Hadoop's disk-based Readers and Writers
4. Dealing with GC Overheads

Memory Overheads

Data structure	Data (bytes)	Fixed overhead (bytes)	Per entry overhead (bytes)	Total (bytes)
int[]={1}	4	16	4	24
String (6 chars)	6	48	8	64
ArrayList<Integer>	4	80	12	96
TreeMap<String, Integer>	7	48	105	160

Table : Memory used by Java objects

Our approach

- Use primitive data structures extensively (ex: byte arrays)
- Operations performed on byte arrays (ex: read, write and sorting), reuse objects

Data Shuffling: Techniques

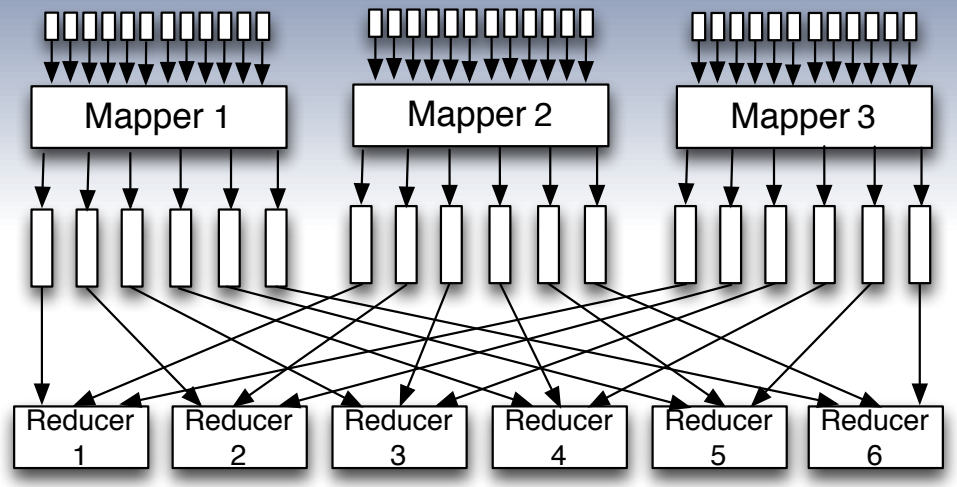


Figure : Pull

Pull model

- **Cons:** More active intermediate streams in memory in the mapper stage results in GC overhead
- **Pros:** No locking contention

Data Shuffling: Techniques

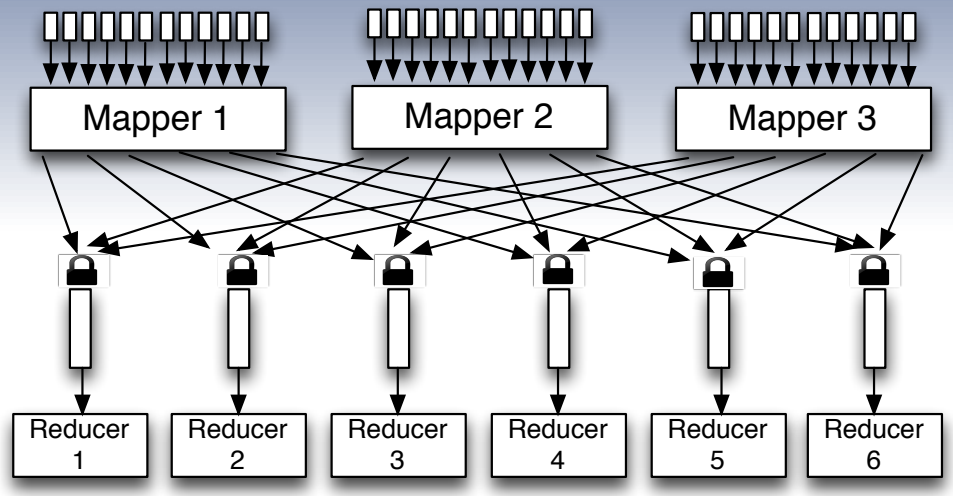


Figure : Push

Push model

- **Cons:** High synchronization overheads for skewed intermediate key distributions
- **Pros:** Creates fewer intermediate data structures

Data Shuffling: Techniques

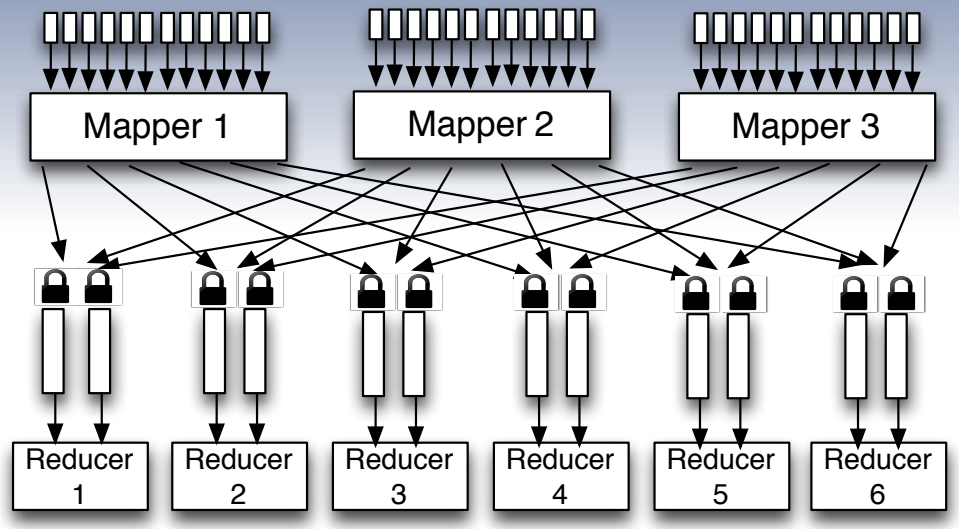


Figure : Hybrid (K=2)

Hybrid model

- **Cons:** Need to set the value of K correctly
- **Pros:** Lower synchronization overheads than Push, fewer intermediate data structures than Pull

Effect of Data Shuffling on Sorting

- Pull approach takes maximum advantage of parallelism in sort stage
  - Intermediate data is in $m * r$ streams
- In Push and Hybrid approaches, intermediate data is in r and $K * r$ streams respectively ($K << m$)
  - Decreased parallelism in the sorting stage making it a bottleneck

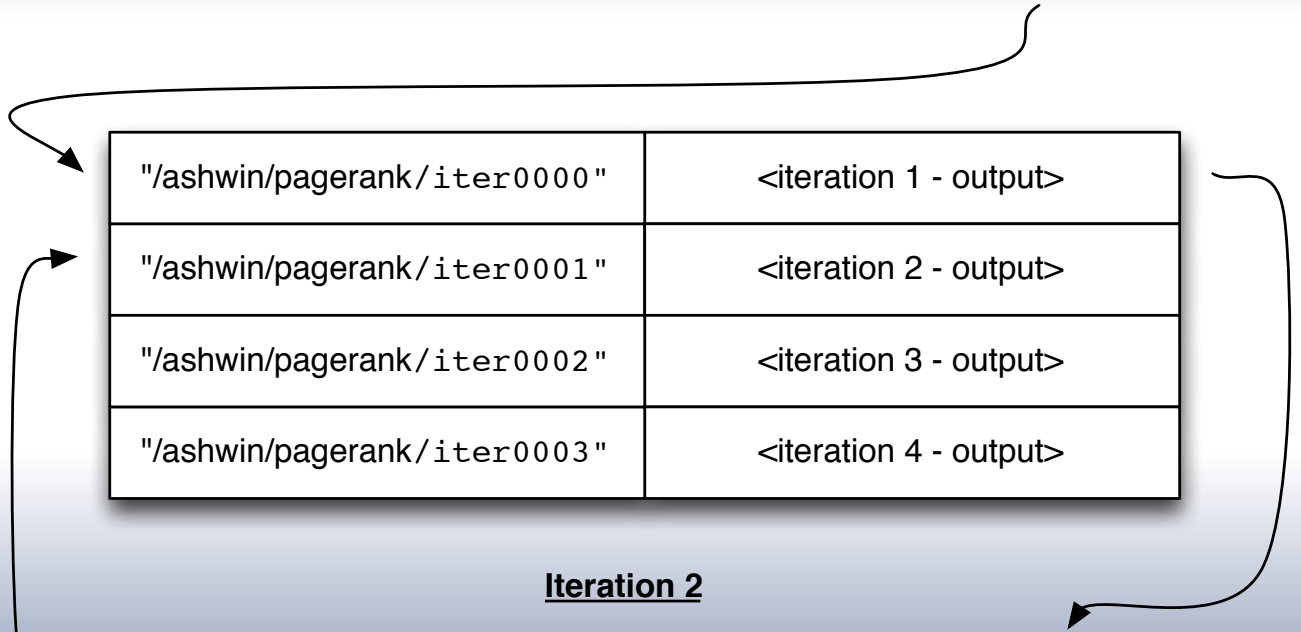
Our solution

- On-the-fly splitting of intermediate streams
- Split size as a parameter to tune

Dealing with Hadoop's disk-based Readers and Writers

Iteration 1

FileOutputFormat.setOutputPath(conf, new Path("/ashwin/pagerank/iter0000"));

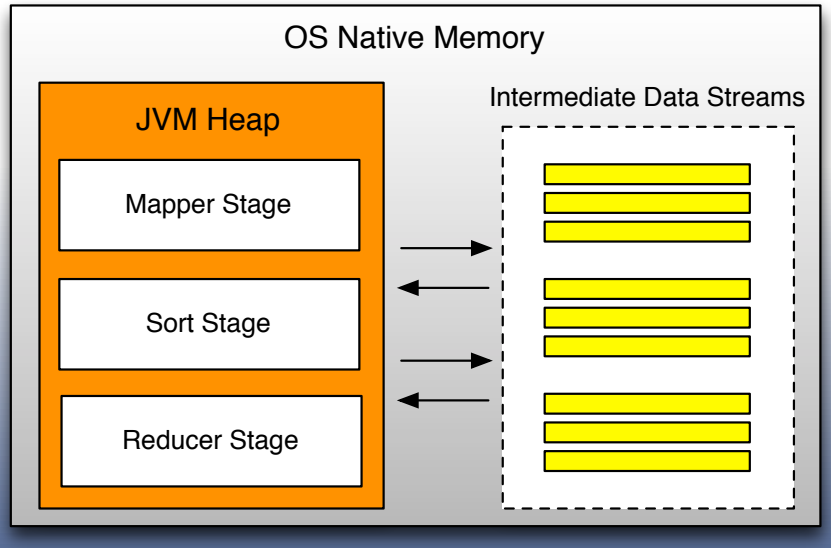


Garbage Collection Overheads

Two common ways to control GC overheads:

- Minimize object creation, reuse objects
  - May not be possible for certain applications
- Fine-tune JVM parameters
  - Too many interdependent parameters
  - Application-dependent

Our solution



Applications

Application	Dataset	Small	Medium	Large
Word Count (WC)	Wiki articles	128MB, 256MB, 512MB	1GB, 2GB	4GB, 8GB, 16GB
K-means (KM)	3D points	12M, 25M	51M, 102M	204M, 398M
Inverted Indexing (II)	Wiki articles	128MB, 256MB, 512MB	1GB, 2GB	4GB, 8GB, 16GB
PageRank (PR)	Wiki graph	0.4M, 0.8M	1.8M, 3.5M	7.2M
LDA	TREC docs	125MB, 256MB	512MB	1GB

Comparison

We compare Hone with

- Hadoop PDM (YARN 2.0.3)
- Hadoop 16-node cluster (CDH4 YARN)
- Phoenix2 (C++)
- Phoenix++ (C++)
- Spark (0.8.0)

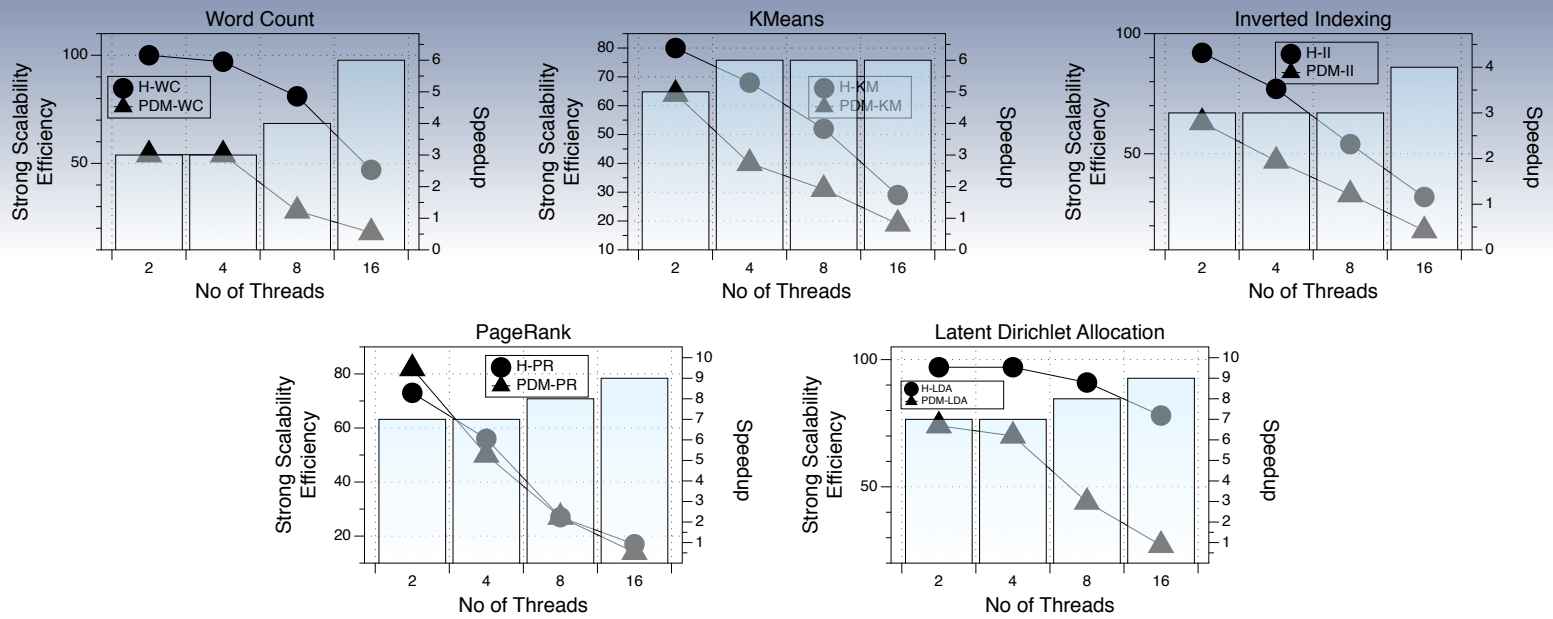
Single machine configuration:

- Dual Intel Xeon quad-core processors (E5620 2.4 GHz)
- 128GB RAM

Cluster configuration:

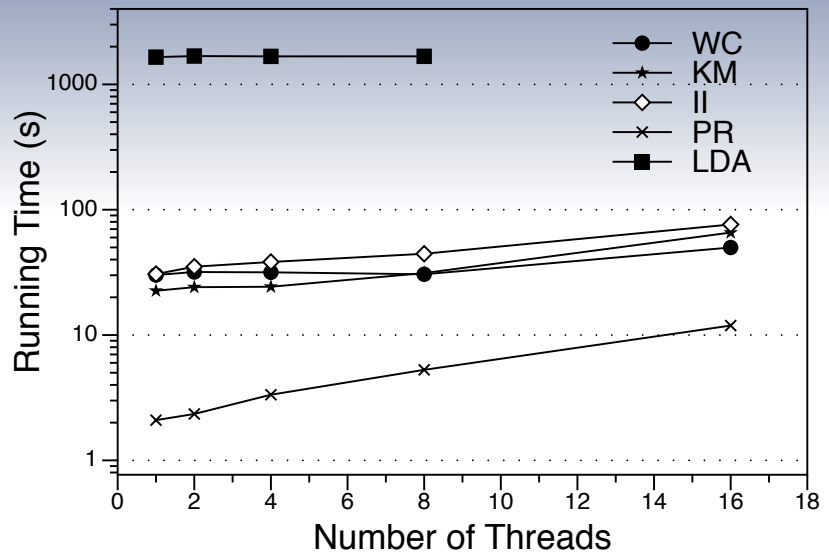
- 16 compute nodes, each has two quad-core Xeon E5520 processors
- 24GB RAM, three 2TB disks

Strong Scalability Analysis



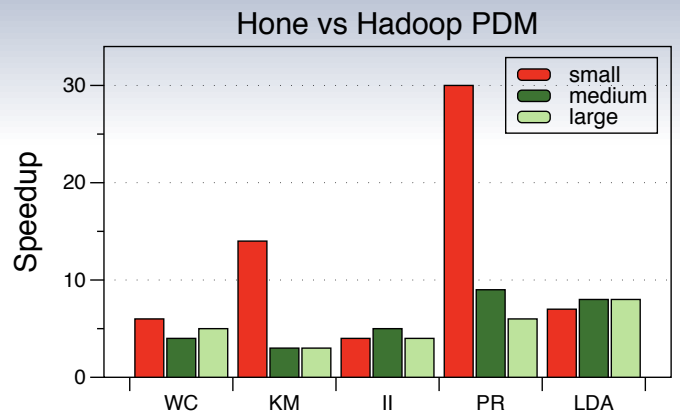
- Strong scalability efficiency: $(\frac{t_1}{N.t_N}) \times 100$
- Ideal curve: linear at 100%
- Dipping curves indicate various overheads
- Hone shows better SSE and speedup over Hadoop PDM

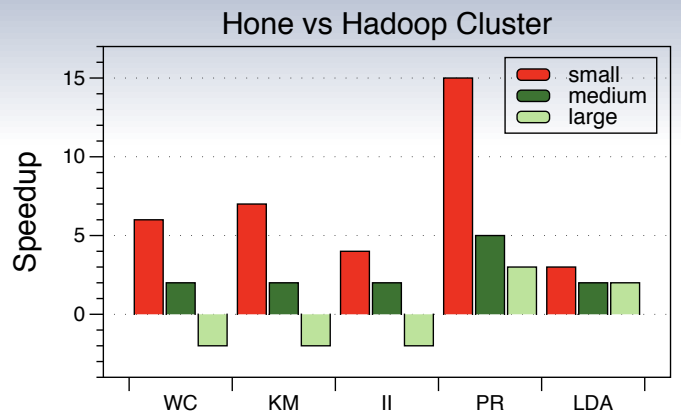
Weak Scalability Analysis



- Ideal curve: Flat line
- Non-flat lines indicate various overheads
- Hone exhibits relatively good weak scaling

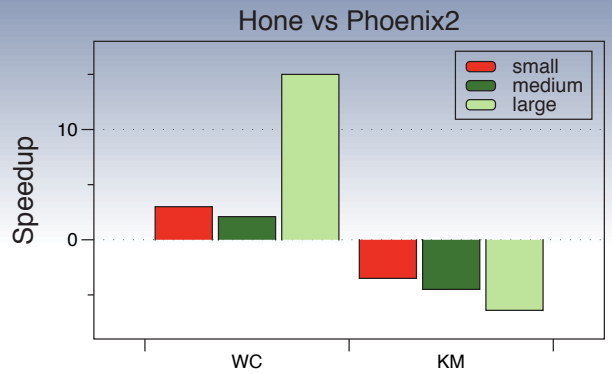
Performance Analysis

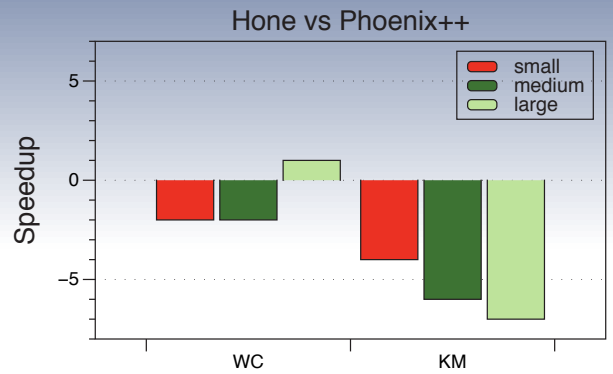


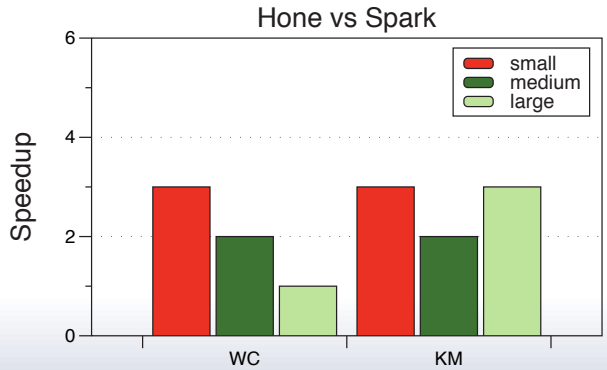


- Hone performs consistently better than Hadoop PDM
- Hone performs better than Hadoop cluster for small and medium size datasets

Performance Analysis







- Hone is mostly slower than Phoenix (C,C++ based) systems
- Hone consistently performs better than Spark

Summary

- Hone is the first MapReduce implementation that is both Hadoop API compatible and optimized for scale-up architectures
- Proposed and evaluated different approaches to implementing the data shuffling stage in MapReduce
- Discussed key challenges in implementing Hone on the JVM, how we addressed them
- Extensive experimentation on real world datasets
- Developed synthetic workload generator for evaluating Hone

Thanks & Questions?

I am graduating!

- Currently in job market
- Do you have a job to offer?
- Email: ashwin@cs.umd.edu