Subsetting and Characterizing Big Data Workloads from BigDataBench

Xinhui Tian

Institute of Computing Technology, Chinese Academy of Sciences

BigDataBench Tutorial
ASPLOS 2016 Atlanta, GA

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY

Overview

- Subsetting Big Data Workloads from BigDataBench

- Characterizing Big Data Workloads on modern processors

Overview

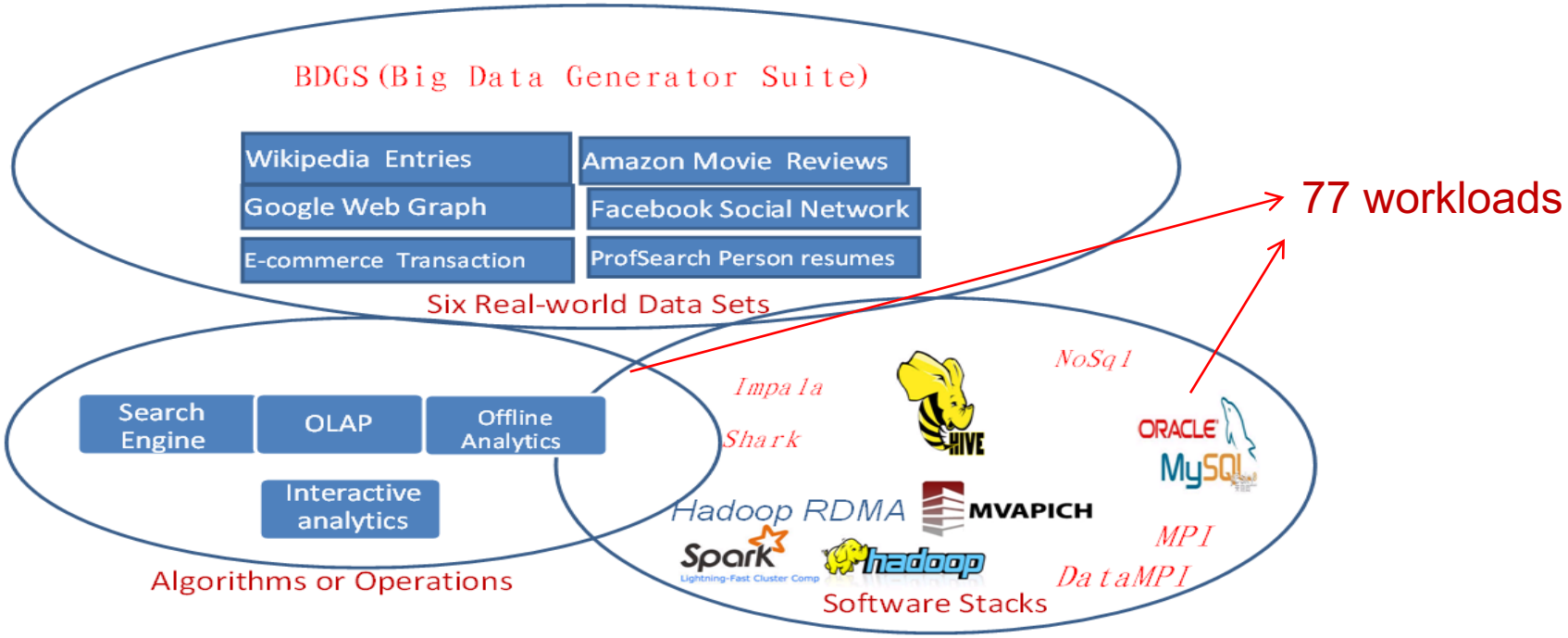
- Subsetting Big Data Workloads from BigDataBench

- Characterizing Big Data Workloads on modern processors

Challenges in Understanding Big Data Apps

- A huge number of representative workloads
  - Hard to thoroughly understand behaviors
  - Prohibitively expensive for simulation-based research
- Having many software stacks aggravates the challenge

Revisit BigDataBench 3.0



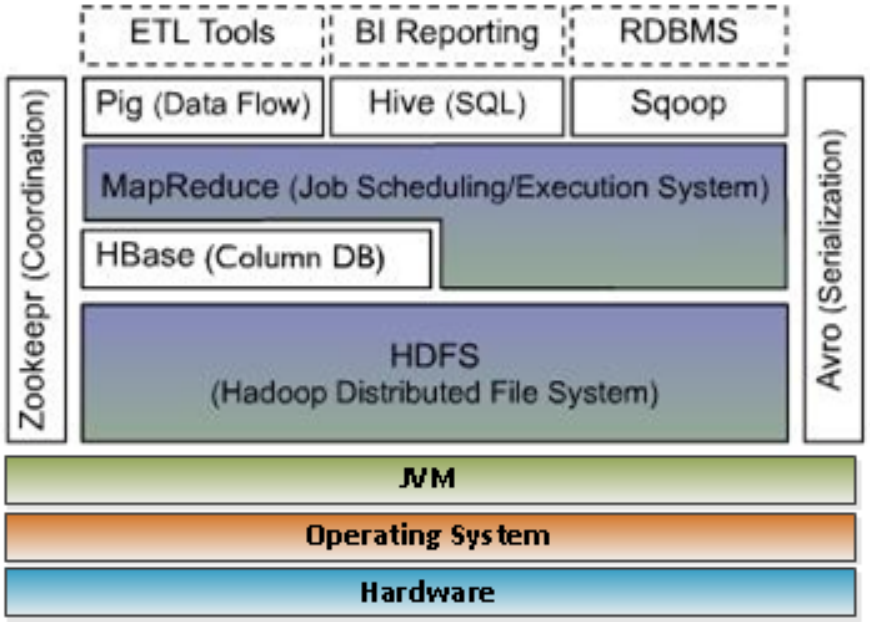
- Include multiple software stacks
- Too time-consuming to run and analyze them all

Why do we consider different software stacks?

- Software stacks have significant impact on workload behaviors--even greater than benchmark algorithms [1]
  - Deep software stacks
  - Integrated mechanisms

Easy to write a big data app

App code << software stack



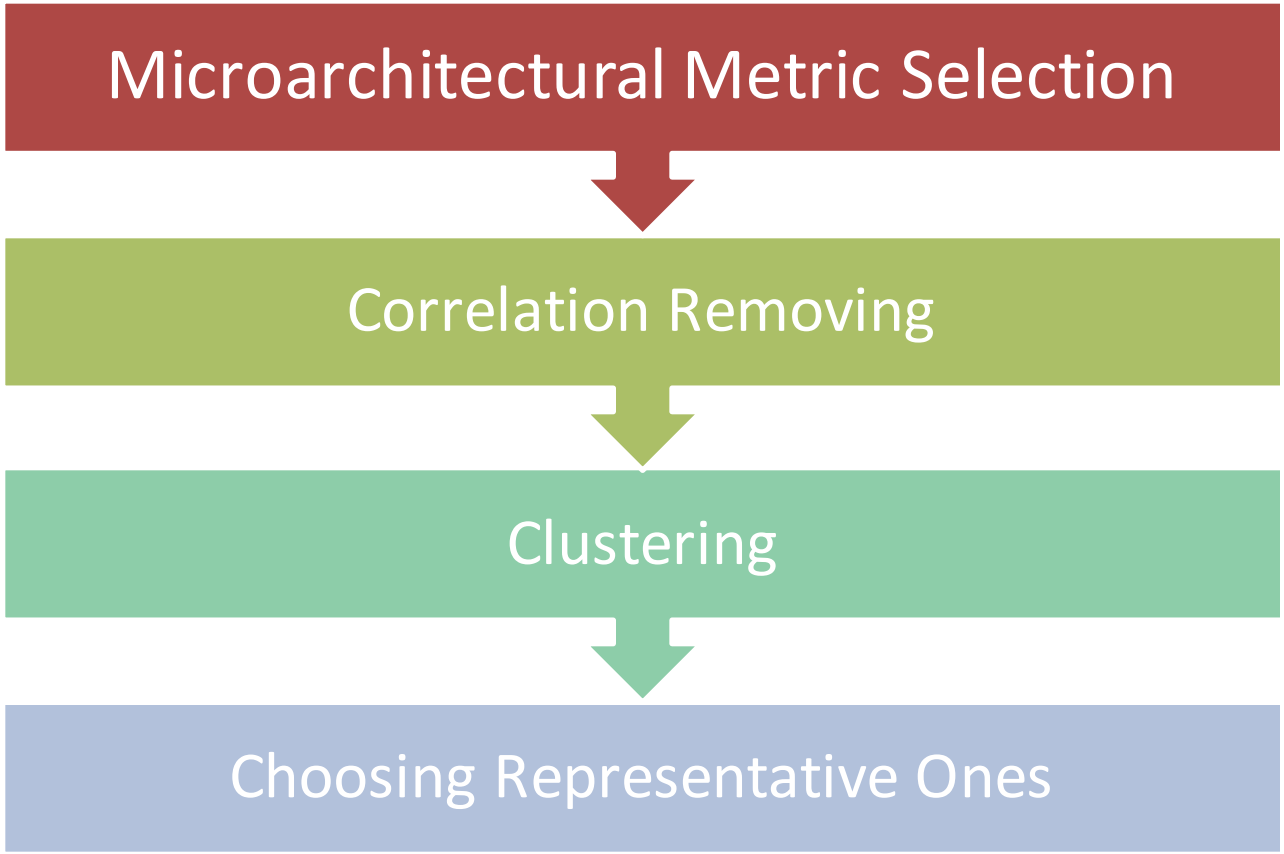
[1]Jia et al. Characterizing and Subsetting Big Data Workloads. IISWC 2014

Goals

- Find a way to downsize BigDataBench 3.0
  - BigDataBench suite contains 77 workloads
  - Should be shrunk to a manageable number
- Reduce the evaluation time of the Big Data research
  - Especially for architecture research using simulator

Subsetting Methodology

--From a view of microarchitecture



Metric Selection

- 45 total metrics, including:
  - Instruction Mix
  - Cache Behavior
  - TLB Behavior
  - Branch Execution
  - Pipeline Behavior
  - Offcore Requests and Snoop Responses
  - Parallelism
  - Operation Intensity
- PMCs accessed via *perf*
- Hard to analyze 77 workloads with 45 metrics

Correlation

- Many program characteristics (metrics) are correlated
  - e.g. long latency cache misses => pipeline stalls
- Correlated data can skew analysis
  - May overemphasize a particular property's importance

PCA

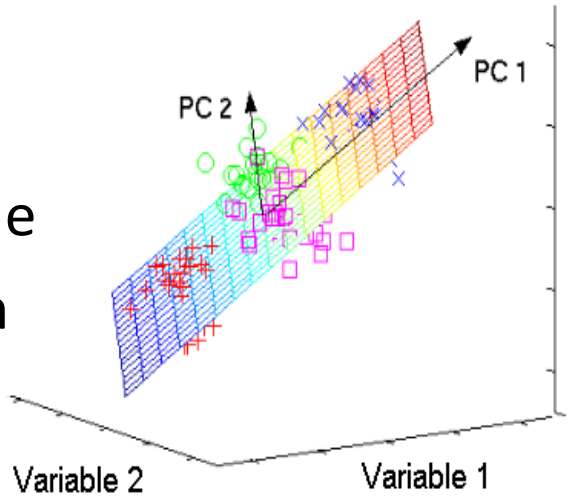
- Use PCA (Principal Components Analysis) to eliminate correlated data.
  - PCA: A statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated

- Principal Components Analysis (PCA) computes Principal Component (Y_i):
  - Yis that are linear combinations of original metrics xis called PCs (Principal Components):
    - $Y_i = a_{i1}x_1 + a_{i2}x_2 + \ldots a_{ip}x_p \; ; \; i=1..p$

PC properties

- PCs are derived in decreasing order of importance, and they are orthogonal
  - First principal component is the direction of greatest variability (covariance) in the data
  - Second is the next orthogonal (uncorrelated) direction of greatest variability
- Keep PCs with eigenvalues >=1
  - Data is ensured to be uncorrelated while capturing most of the original information

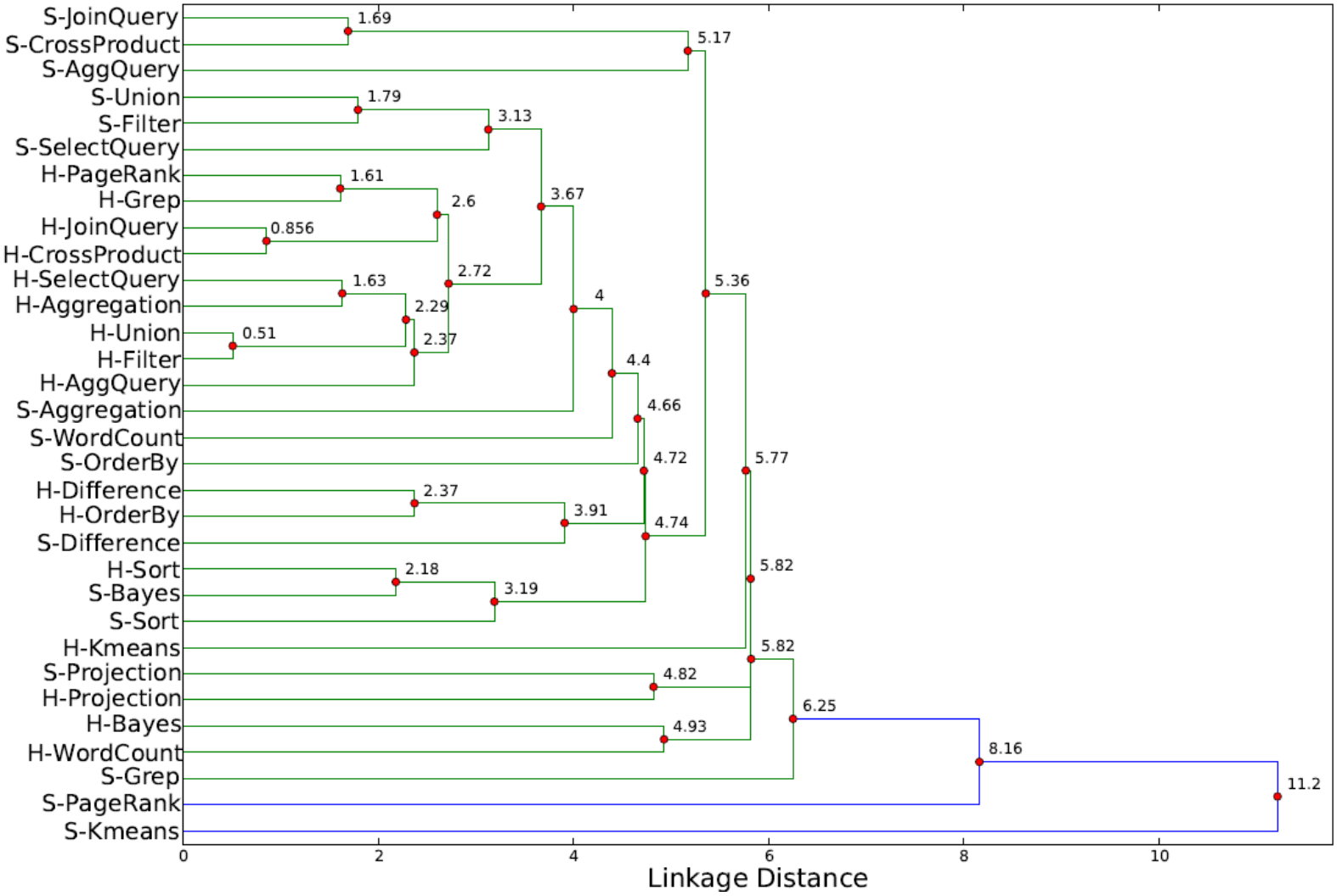
(Kaiser criterion)



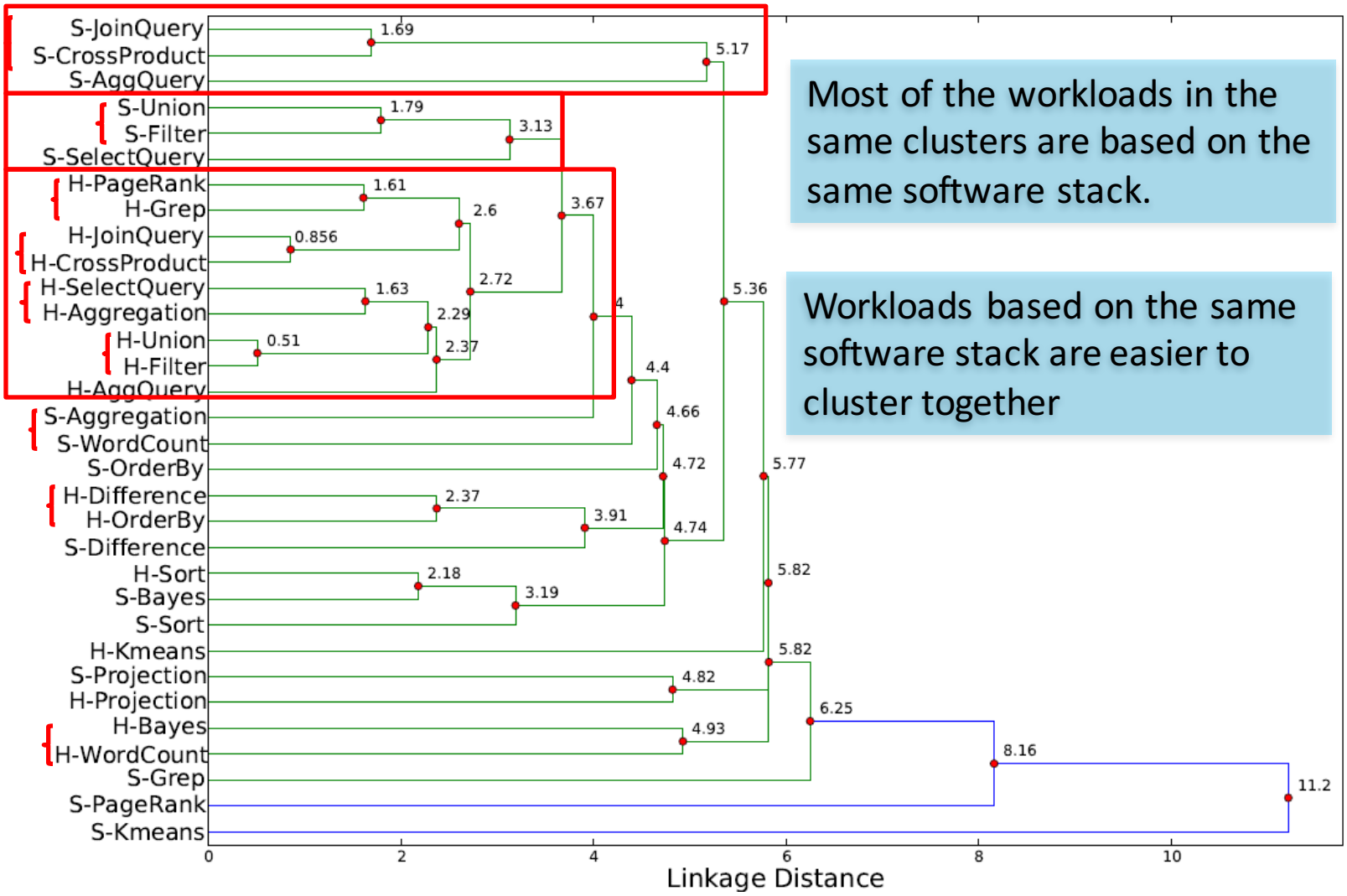
Measuring Similarity

- Choose principal components
  - Kaiser criterion: keep PCs with eigenvalues >=1
  - 8 PCs chosen retain 91.12% variance
- Quantitatively show similarity
  - Hierarchical clustering
  - Dendrograms

The Dendrogram



Observations

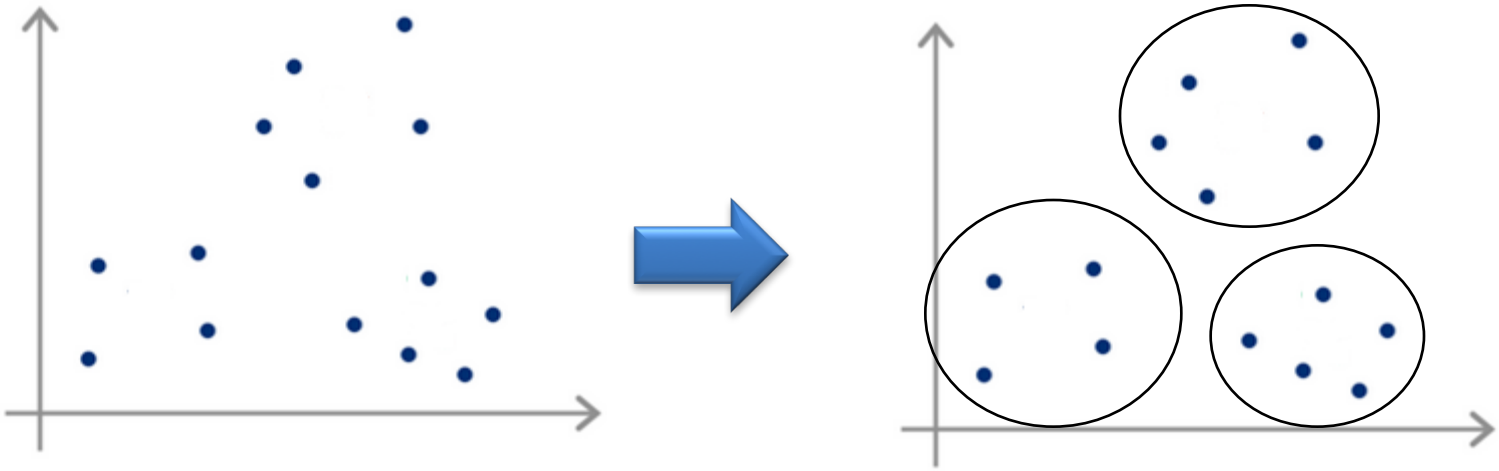


Most of the workloads in the same clusters are based on the same software stack.

Workloads based on the same software stack are easier to cluster together

Subsetting Method: Clustering

- Use K-means algorithm to partition workloads into K clusters.



- The problem: how to chose K?

BIC

- Use Bayesian Information Criterion (BIC) to choose proper K value
  - Measures how well the clustering fits the data set
  - Larger BIC scores are better
  - We choose the K with highest BIC scores

$$BIC(D,K) = l(D|K) - \frac{p_j}{2} log(R)$$

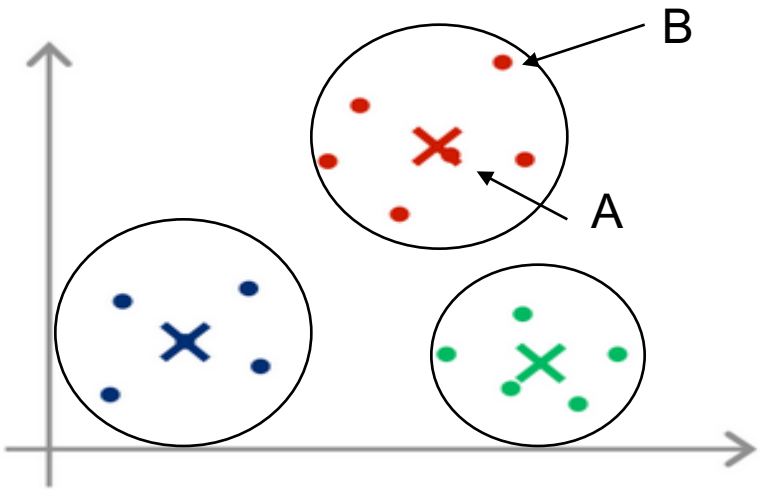
$$l(D|K) = \sum_{i=1}^{K} (-\frac{R_i}{2} log(2\pi) - \frac{R_i \cdot d}{2} log(\sigma^2) - \frac{R_i - K}{2} + R_i log R_i - R_i log R)$$

$$\sigma^2 = \frac{1}{R-K} \sum_i (x_i - \mu(i))^2$$

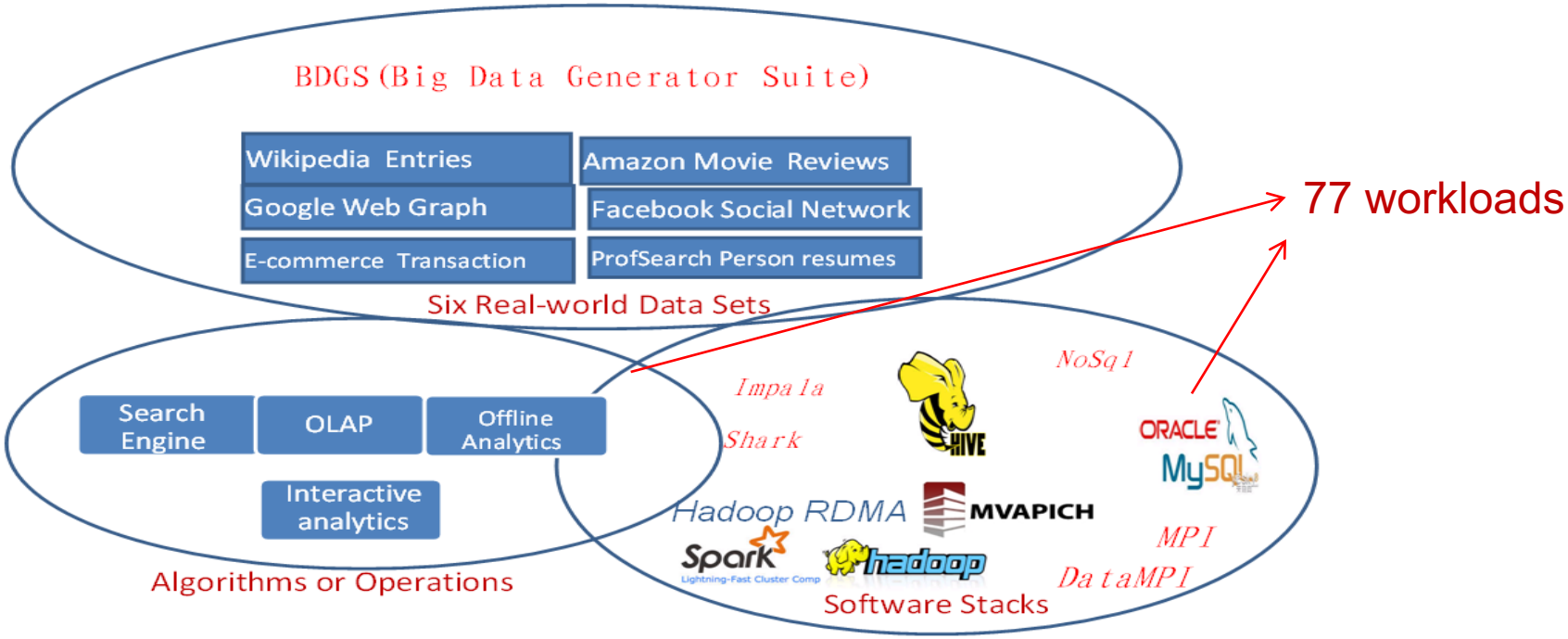
Selecting Representative Workloads

- Select workloads near the cluster center

- Select workloads near the cluster boundary

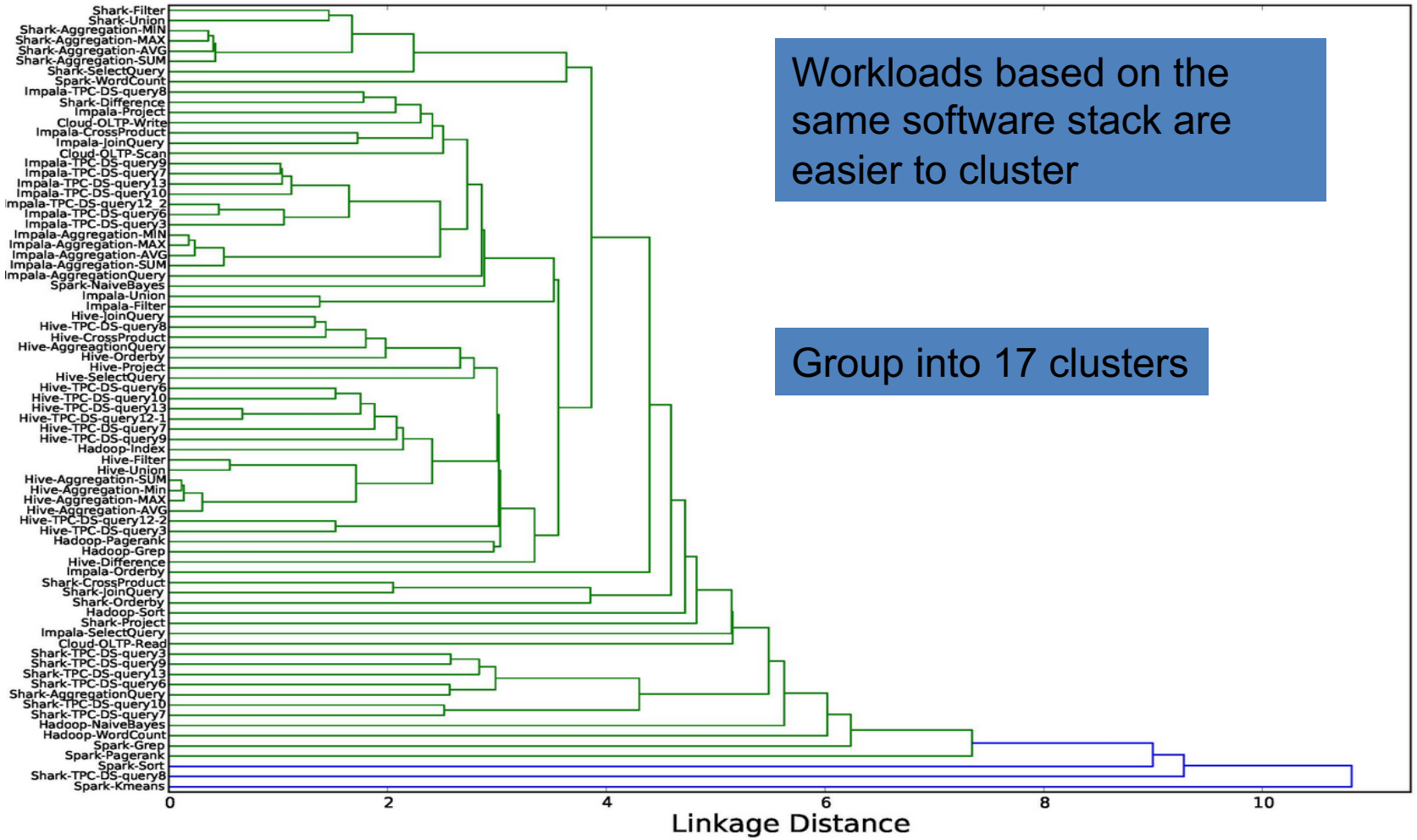


Revisit BigDataBench



- Reasonable to include multiple software stacks
- Too time-consuming to run and analyze them all

Subsetting 77 Workloads



Workloads based on the same software stack are easier to cluster

Group into 17 clusters

Architecture Subset Workloads

No. of cluster	Workload name	Number of workloads in cluster
1	Cloud-OLTP-Read	10
2	Hive-Difference	9
3	Impala-SelectQuery	9
4	Hive-TPC-DS-query3	9
5	Spark-WordCount	8
6	Impala-Orderby	7
7	Hadoop-Grep	7
8	Shark-TPC-DS-query10	4
9	Shark-Project	4
10	Shark-Orderby	3
11	Spark-Kmeans	1
12	Shark-TPC-DS-query8	1
13	Spark-Pagerank	1
14	Spark-Grep	1
15	Hadoop-WordCount	1
16	Hadoop-NaiveBayes	1
17	Spark-Sort	1

What do those 17 workloads do?

- Offline analytics:
  - Sort, Grep, Word Count, Page Rank, K-means, Bayes
- No-SQL operation: Read
- TPC-DS queries:
  - Query 3, 8, 10
- Basic relational algebra operations:
  - Difference
  - Select query to filter data
  - Sorting
  - Project

Overview

- Subsetting Big Data Workloads from BigDataBench

- Characterizing Big Data Analytics Workloads on modern processors

Goals

- Understanding the behaviors of big data analytics workloads on modern processors

- Identifying potential optimization methods for big data analytics workloads

Experimental Methodology

- Start from a small scale
  - Five nodes: 1 master + 3 slaves
  - Each node: Xeon E5-2680 v3, 2 Sockets, 12 Cores@2.5GHz
- Performance results
  - Use PMC for micro-architecture level performance data
  - Top-Down method
  - Measure the entire run

Benchmarks and Their Data Sets

No.	Workload	Input Data Size	#Retired Instructions (Billions)
1	Sort	100 GB binary files	8500
2	WordCount	100 GB documents	12600
3	Grep	100 GB documents	4230
4	Naive Bayes	100 GB text	7170
5	SVM	100 GB text	10200
6	K-means	100 GB vector	30500
7	Fuzzy K-means	100 GB vector	118000
8	IBCF	101 GB ratings data	77200
9	HMM	100 GB text file	4450
10	PageRank	2^{29} vertexes	34100
11	Hive-Bench	75.5 million records	1100

- Store in both memory and disk

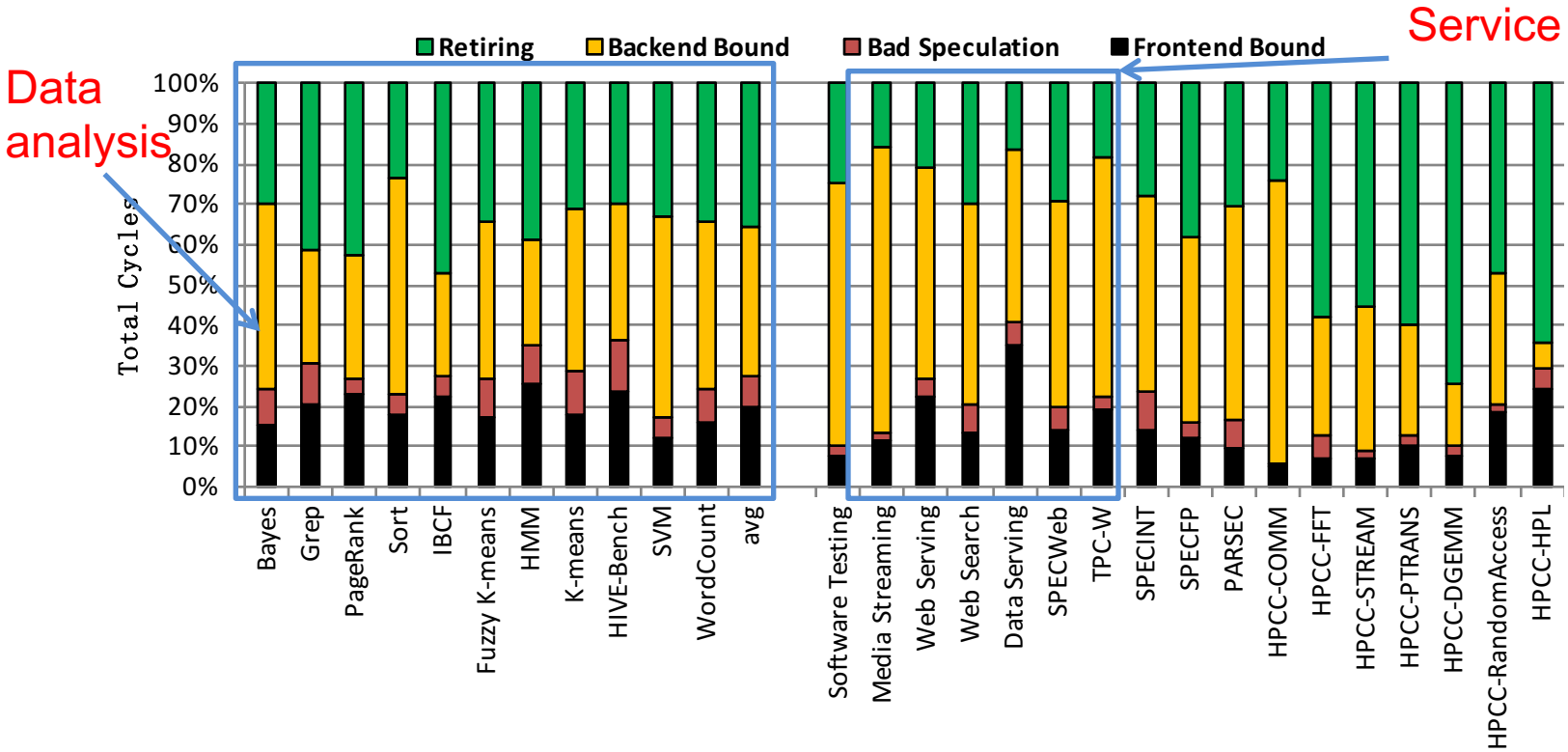
Compared Benchmarks

Filed：	Scale out workloads	HPC	CPU	Web
Workloads：	CloudSuite v1	HPCC	SPEC CPU 2006	SPEC Web 2005
	Web search	HPL	SPEC INT	TPC-W
	Data serving	Streaming	SPEC FP	
	Web serving	Ptrans	PARSEC	
	Media streaming	RandomAccess		
	Software testing	DGEMM		
		FFT		
		Comm		

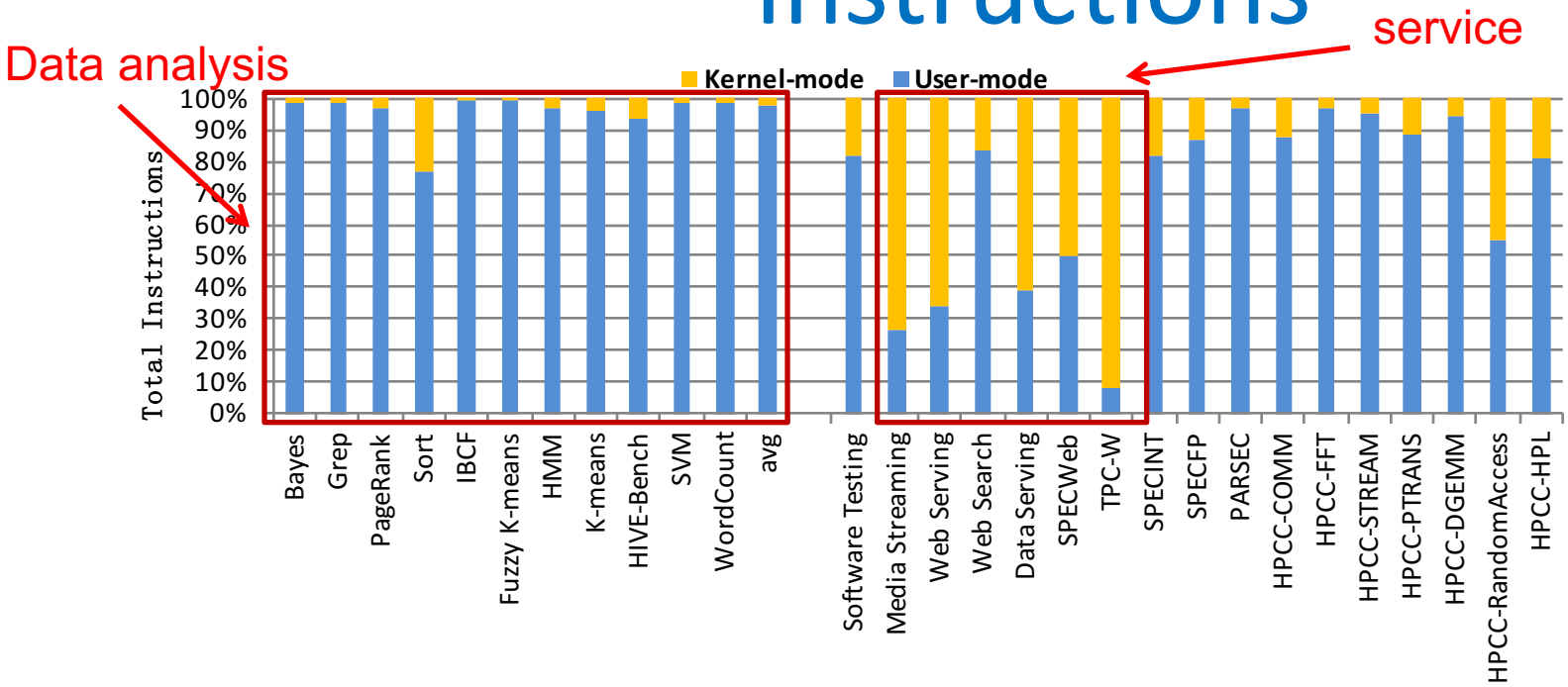
- Scale-out service workloads share many similarity characteristics with that of traditional service workloads.
- So we just use the service workloads to describe them

The Top Level Top-Down Analysis Breakdown

- The service workloads are more backend bound
- Front end stalls are more than most of traditional workloads



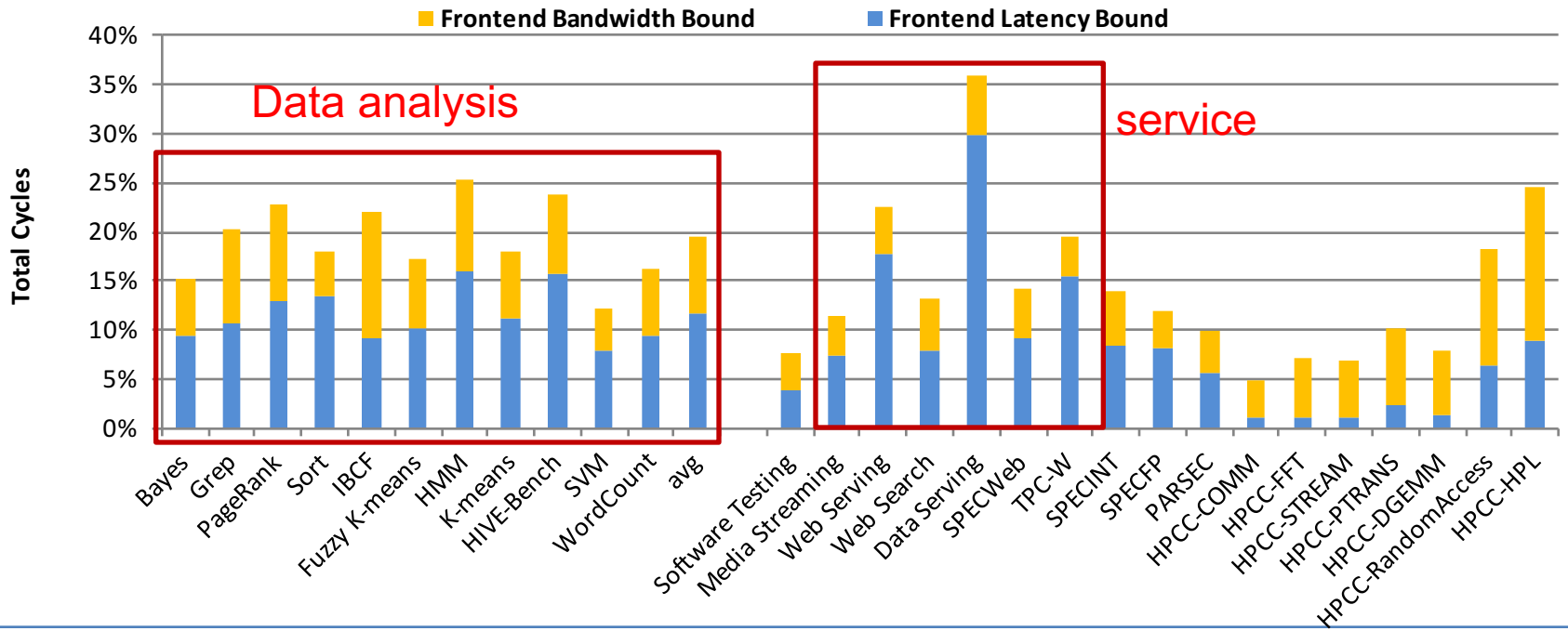
Breakdown of Executed Instructions



- Analysis workloads have more application level instructions
- The service workloads have higher percentages of kernel level instructions

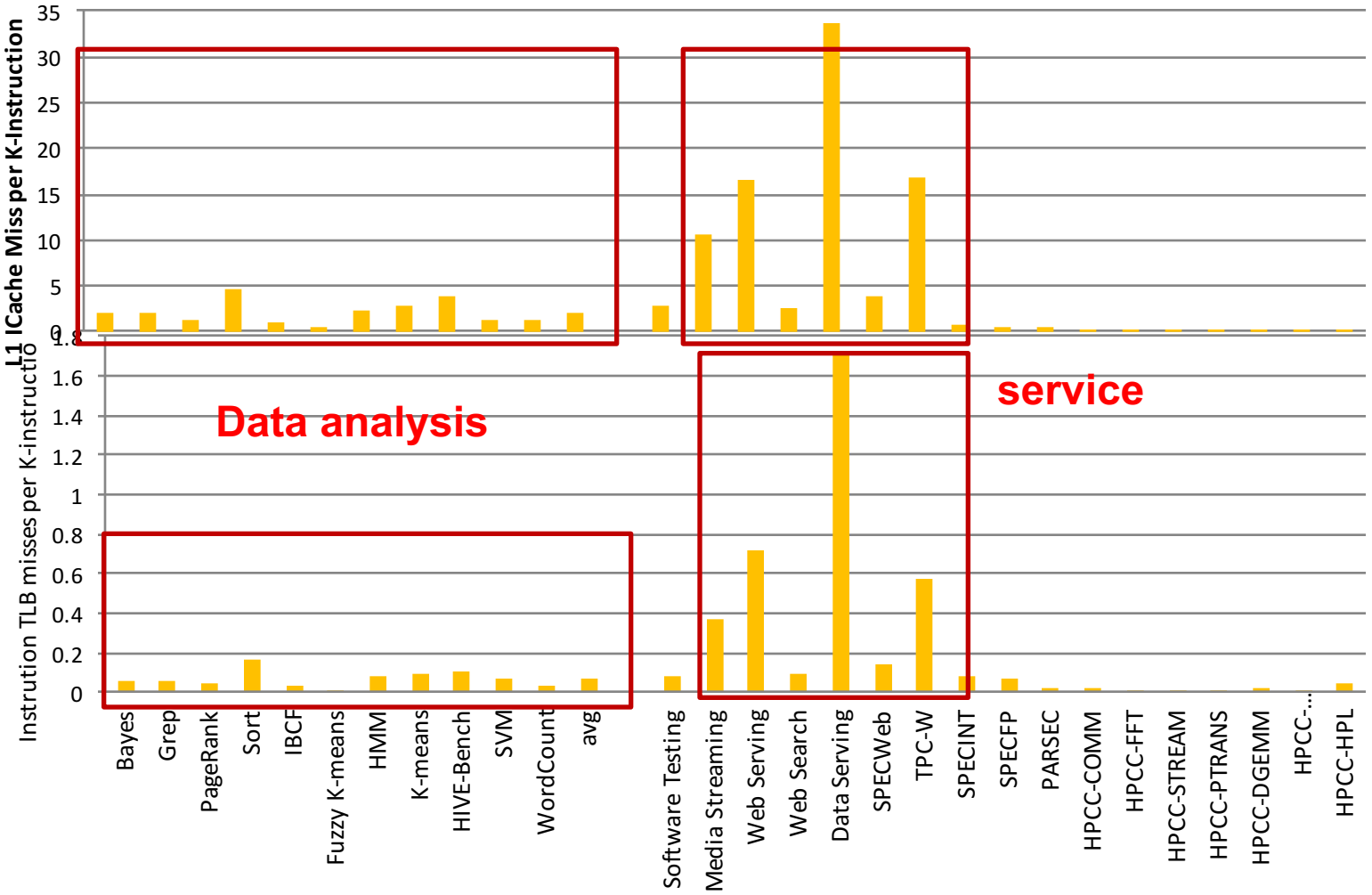
Frontend Breakdown

- Analytics and service workloads are more latency bound; service workloads are more severe.
- Traditional workloads are more bandwidth bound



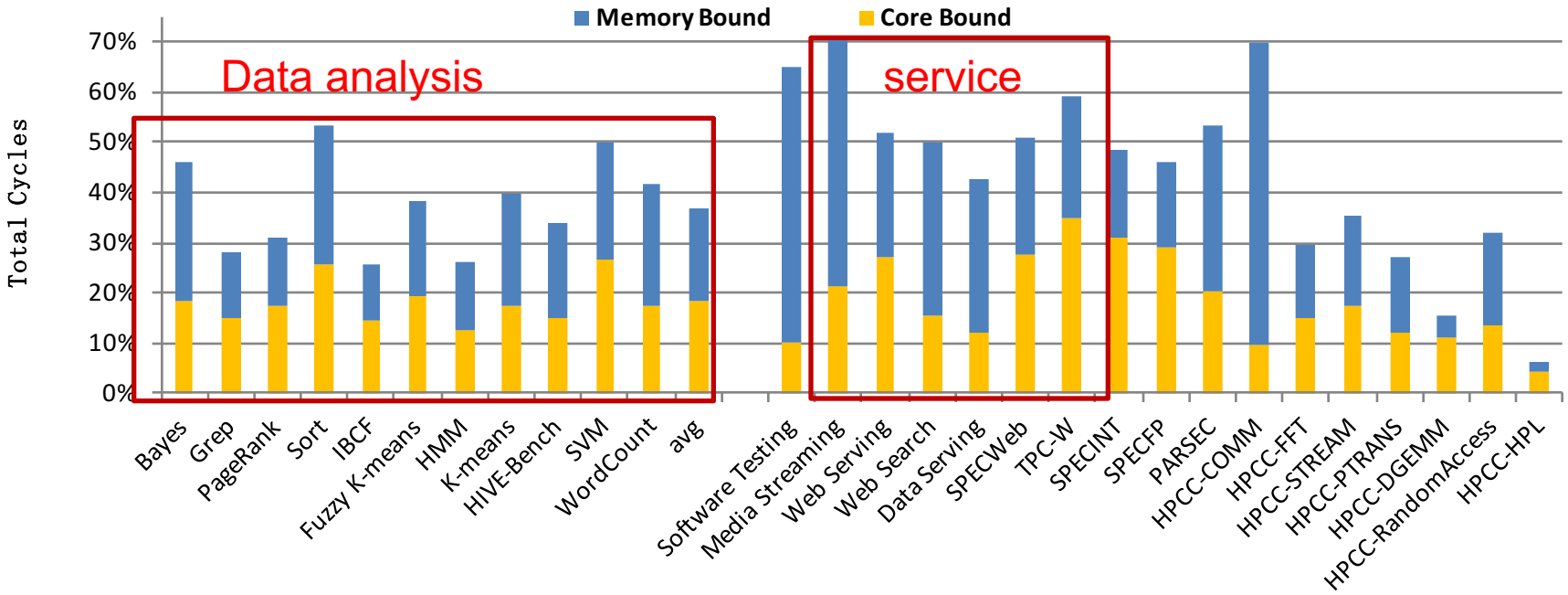
Reasons of Front End Stalls

- High Icache misses and ITLB misses cause front end stall



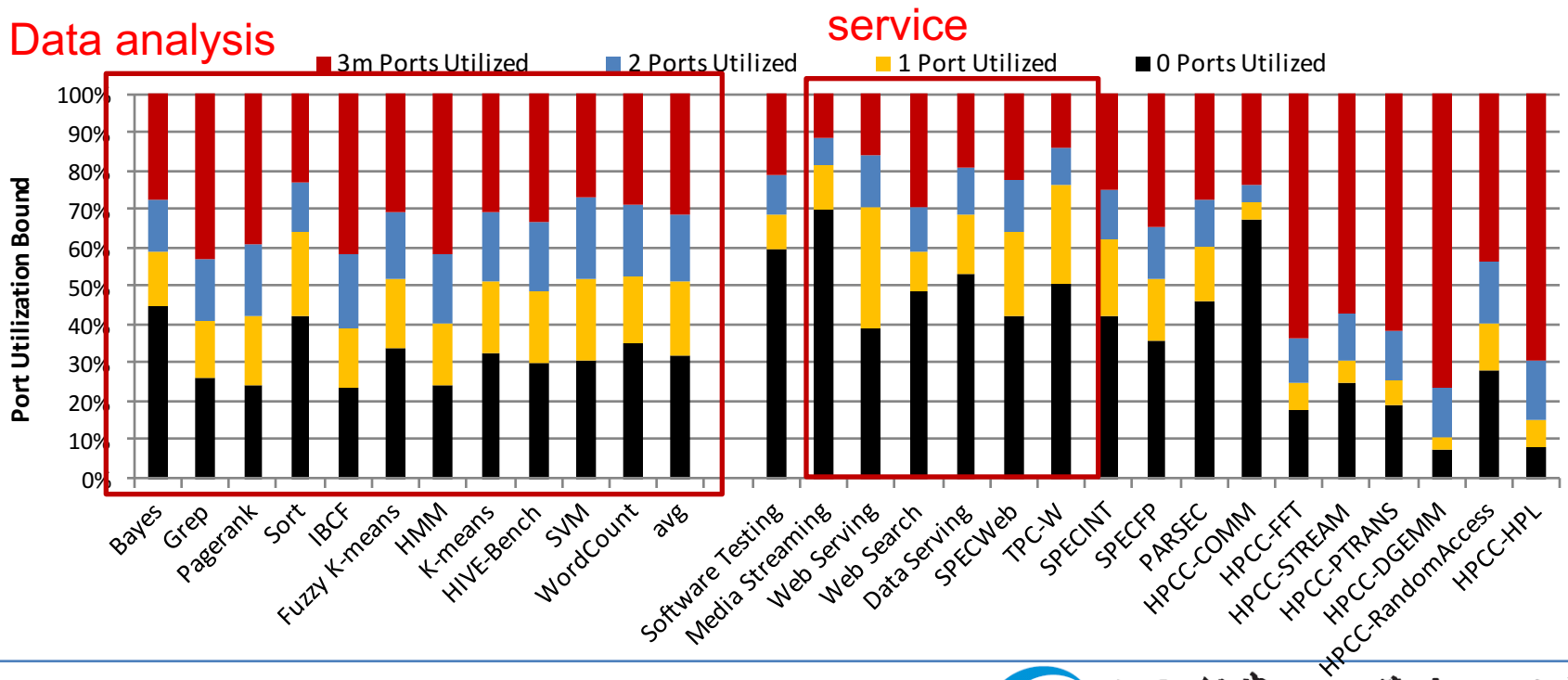
Backend Breakdown

- Service workloads are more memory bound
- Data analysis workloads: Memory and core bound are similar.



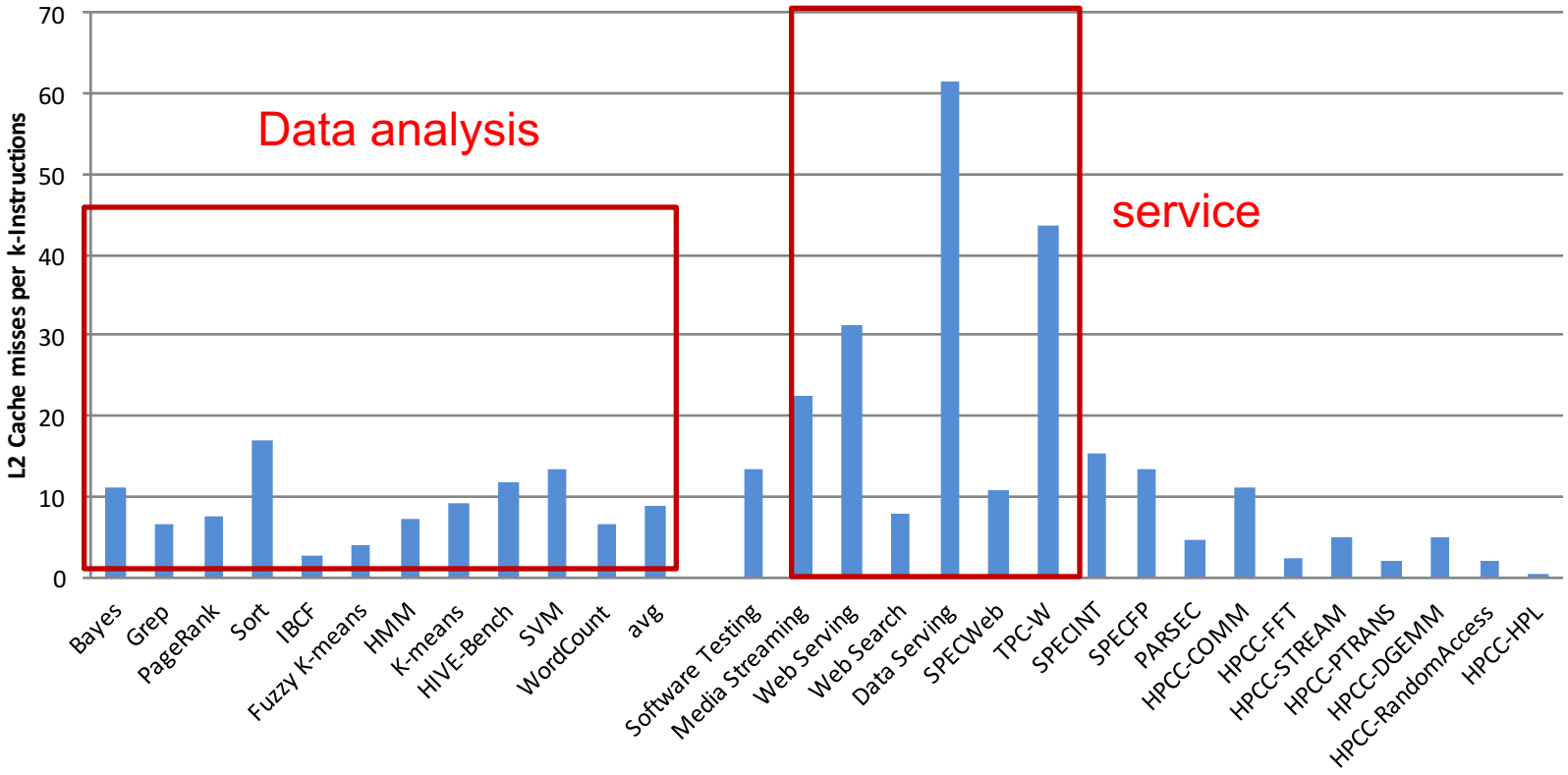
Execution Port Utilization

- Low port utilization for both data analysis and service. Service is more severe.
- Increase the port utilization: vectorization or better scheduling



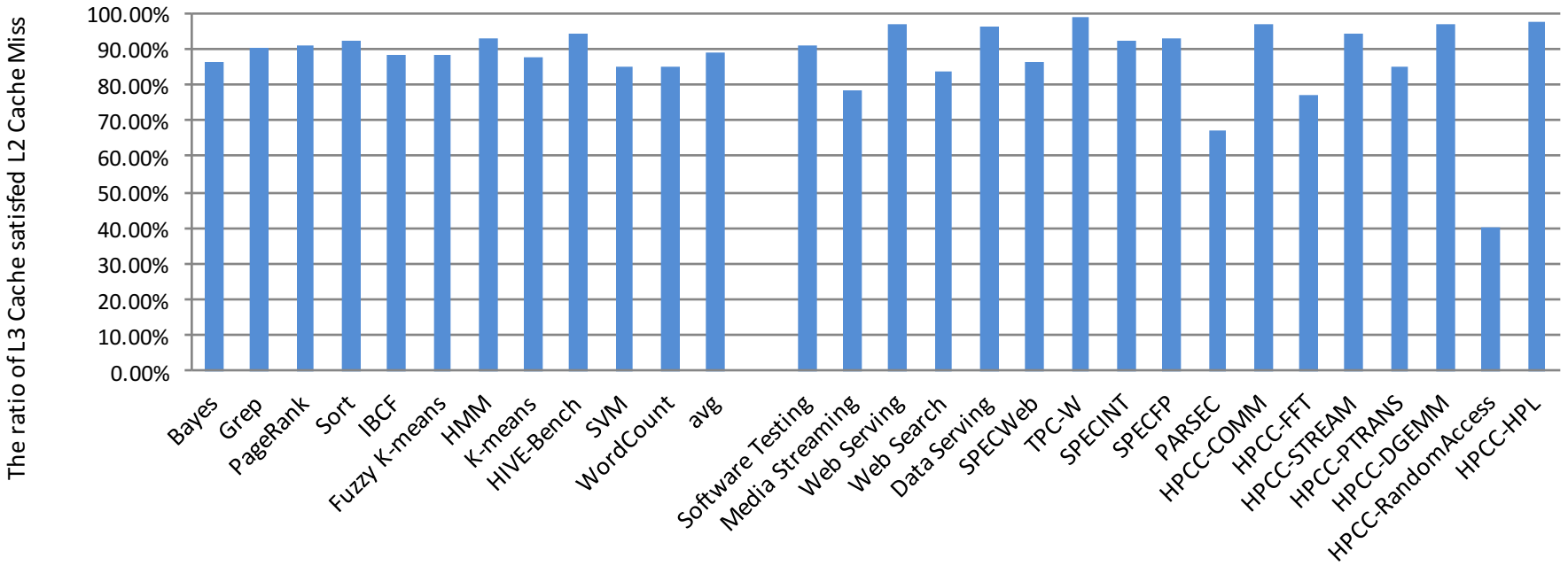
L2 Cache Behaviors

- Data analysis workloads have good L2 cache behaviors



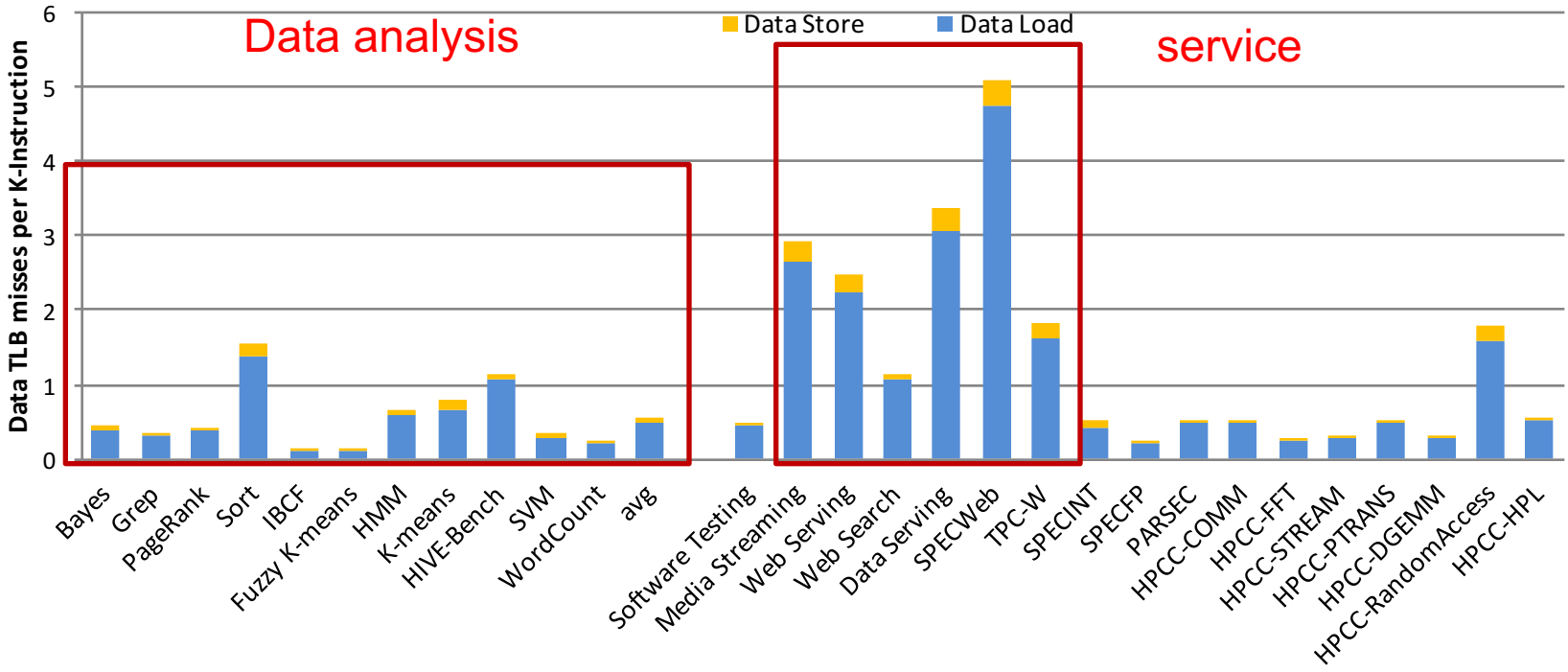
LLC behaviors

- Data Center workloads
  - Have good LLC behaviors
  - Better than most of the HPC workloads



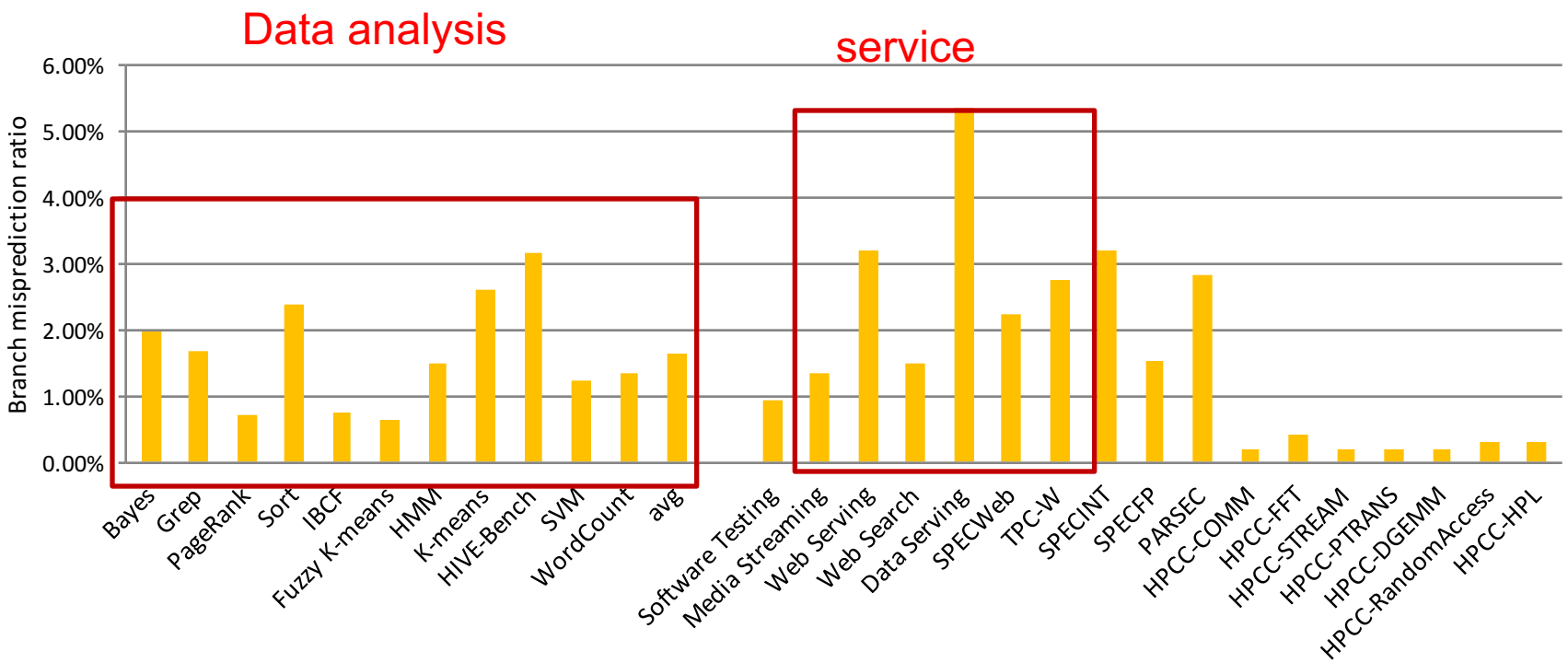
DTLB Behaviors

- Data analysis workloads :
  - More DTLB misses than most of HPC workloads
  - Fewer DTLB misses than service workloads.



Branch Prediction

- Data analysis workloads have pretty good branch behaviors
- Branches of Services workloads are hard to predict



Some Observations

- Analysis workloads are different from scale-out service workloads and traditional workloads
- More app level instructions are executed
- High Icache and ITLB misses
  - Impact: High percentage of front end stall
  - Cause: Massive scale of software infrastructure, high level languages, third party lib
  - Rethink the design of Icache or ITLB or simplify SW stack
- Low level caches are good
  - L2 cache is reasonably effective
  - L3 cache is extremely effective
  - Pay more attention to area and energy of caches
- The branch predictor is quite effective

Correlation Analysis

- Compute the correlation coefficients of CPI with other micro-architecture level metrics.
  - Pearson's correlation coefficient:

$$\rho(X,Y) = corr(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

  - Absolute value (from 0 to 1) shows the dependency:
    - The bigger the absolute value, the stronger the correlation.

Top five coefficients

Workload	Correlation Coefficients		Workload	Correlation Coefficients	
Naive Bayes	ITLB miss	0.965	SVM	ITLB miss	0.995
	L2 cache miss	0.957		L2 cache miss	0.993
	Data Store TLB miss	0.950		Data Store TLB miss	0.992
	Kernel-mode instruction	0.948		Kernel-mode instruction	0.984
	Data Load TLB miss	0.881		Data Load TLB miss	0.981
Grep	ITLB miss	0.994	WordCount	Kernel-mode instruction	0.979
	L2 cache miss	0.993		ITLB miss	0.962
	Data Store TLB miss	0.989		L2 cache miss	0.92
	Kernel-mode instruction	0.981		Data Store TLB miss	0.571
	Data Load TLB miss	0.974		Store Bound	0.501
K-means	Data Store TLB miss	0.781	Fuzzy K-means	Data Store TLB miss	0.936
	L2 cache miss	0.691		L2 cache miss	0.889
	Kernel-mode instruction	0.683		Branch misprediction	0.881
	ITLB miss	0.682		ITLB miss	0.848
	Data Load TLB miss	0.569		L3 cache miss	0.0.835
PageRank	ITLB miss	0.979	Sort	L2 cache miss	0.959
	Data Store TLB miss	0.977		ITLB miss	0.957
	Data Load TLB miss	0.975		Data Store TLB miss	0.919
	L2 cache miss	0.967		Data Store TLB miss	0.865
	Kernel-mode instruction	0.927		Kernel-mode instruction	0.854
Hive Bench	ITLB miss	0.988	IBCF	L2 cache miss	0.981
	L2 cache miss	0.987		ITLB miss	0.960
	Data Store TLB miss	0.978		Data Store TLB miss	0.944
	Kernel-mode instruction	0.909		Kernel-mode instruction	0.893
	Store Bound	0.890		Store Bound	0.864
HMM	L2 cache miss	0.990	TPC-W	Data Load TLB miss	0.733
	ITLB miss	0.970		ITLB	0.726
	Data Store TLB miss	0.947		L2 cache miss	0.726
	Data Load TLB miss	0.922		Data Store TLB miss	0.715
	Kernel-mode instruction	0.904		Branch misprediction	0.655

Insights

- Frontend related stall does not have high correlation coefficient value for most of big data analytics workloads
  - Frontend stall is not the factor that affects the CPI performance most.
- L2 cache misses and TLB misses have high correlation coefficient values.
  - The long latency memory accesses (access L3 cache or memory) affect the CPI performance most and should be the optimization point with highest priority.

Software Stack Impacts

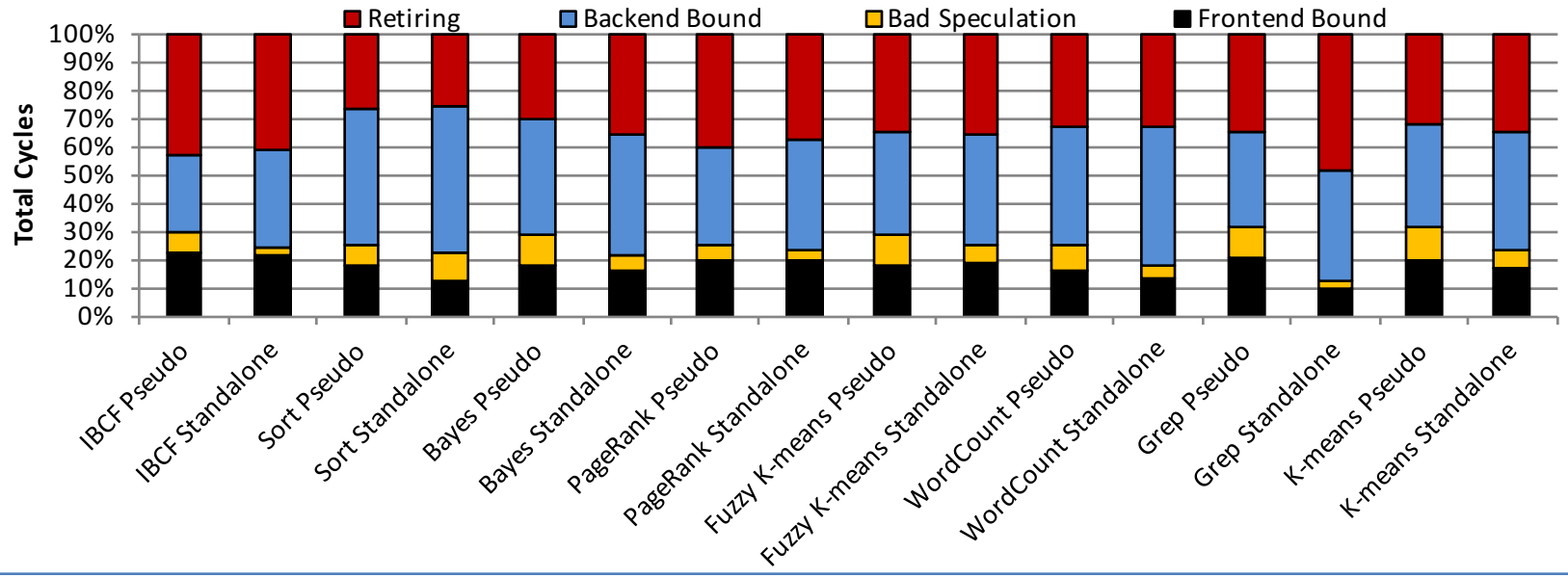
- Hadoop has different modes, with which we can run the same app but invoke different call hierarchy.

	Standalone	Pseudo-distributed
Hadoop daemons	N	Y
HDFS	N	Y
MapReduce API	Y	Y
JVM	Y	Y

- Standalone and Pseudo-distributed mode are used to compare the impact of software stack.

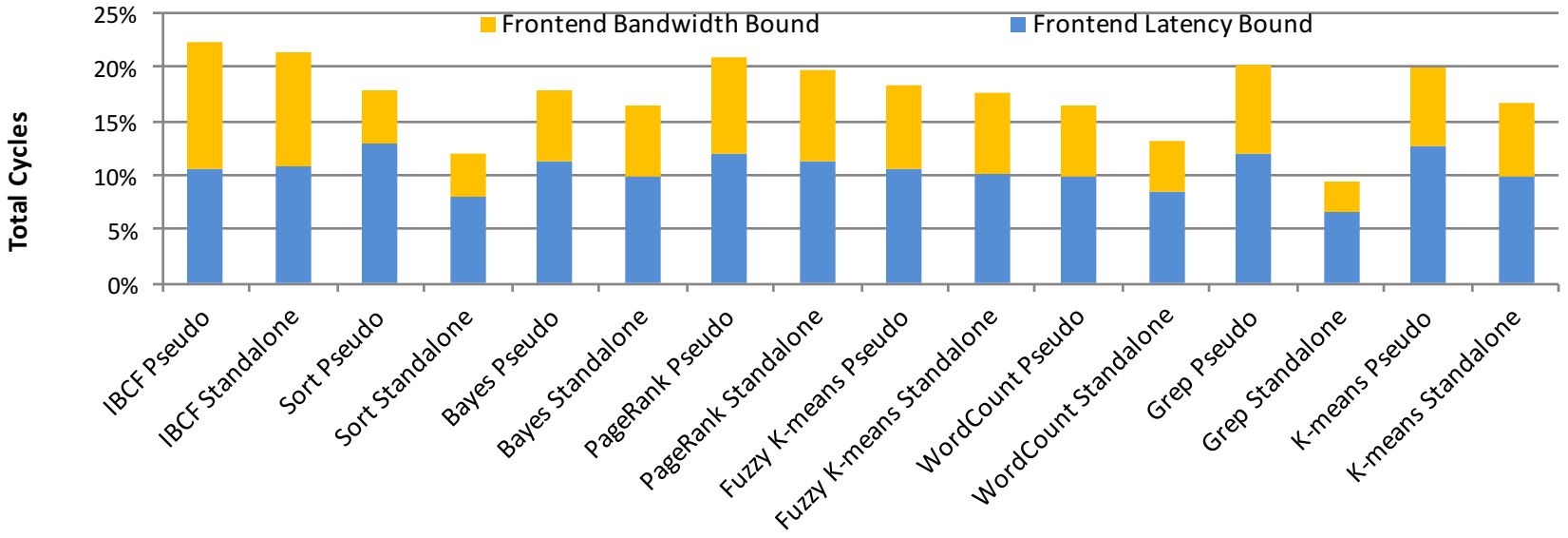
Top Level differences

- Full software stack (Pseudo-distributed)
  - more Frontend stall
  - More bad speculation
  - Less Backend stall



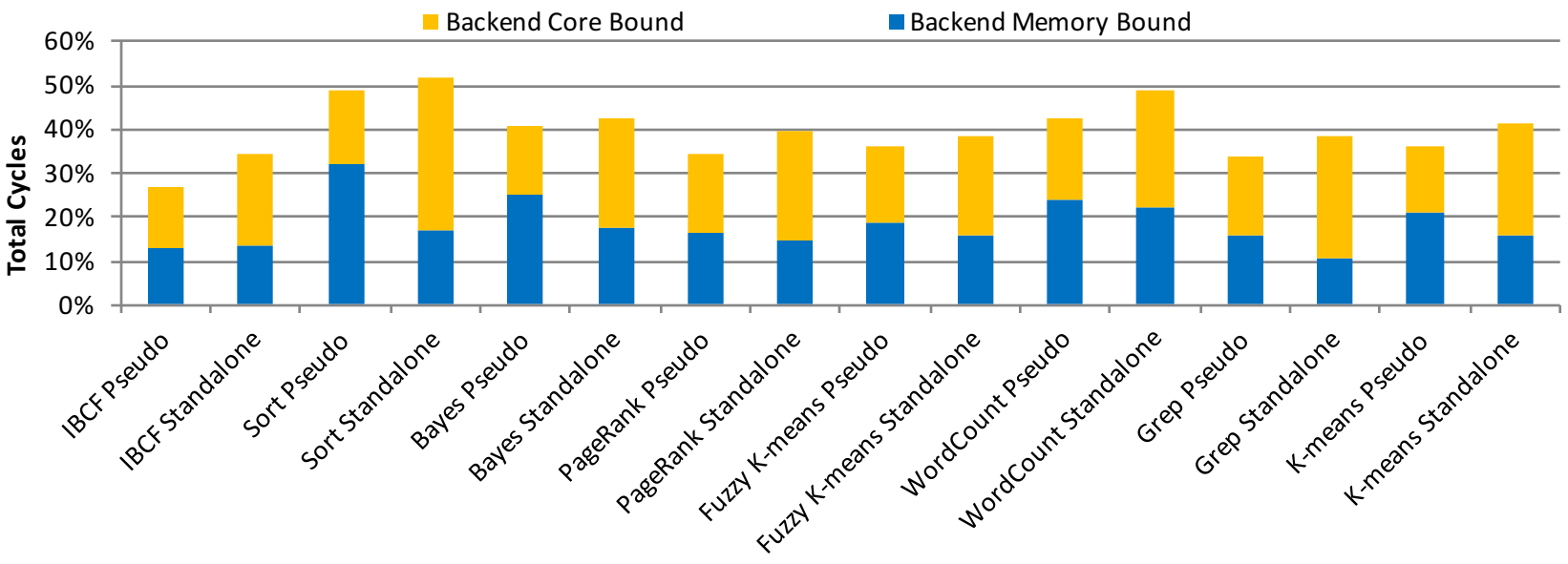
Frontend

- Pseudo-distributed mode workloads have more bandwidth bound cycle.
  - The software stack increases the Icache and ITLB pressures; prolongs the instruction refill latency.



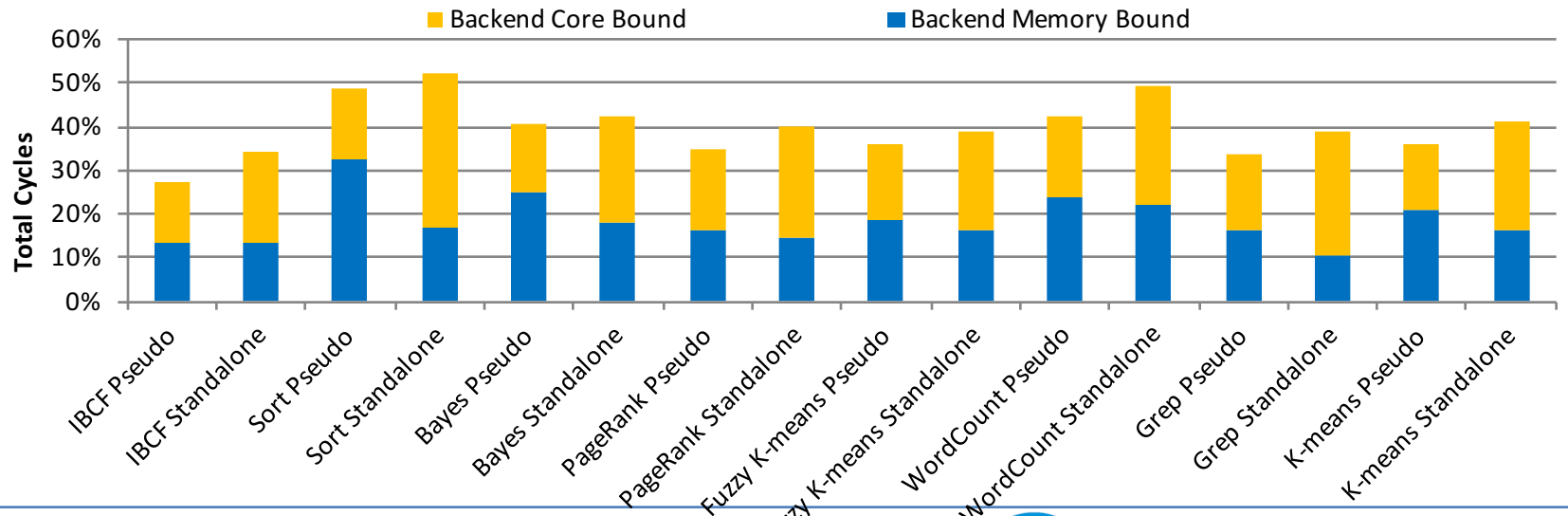
Backend

- Pseudo-distributed workloads have more memory bound cycles.
  - The software stack increases the working set



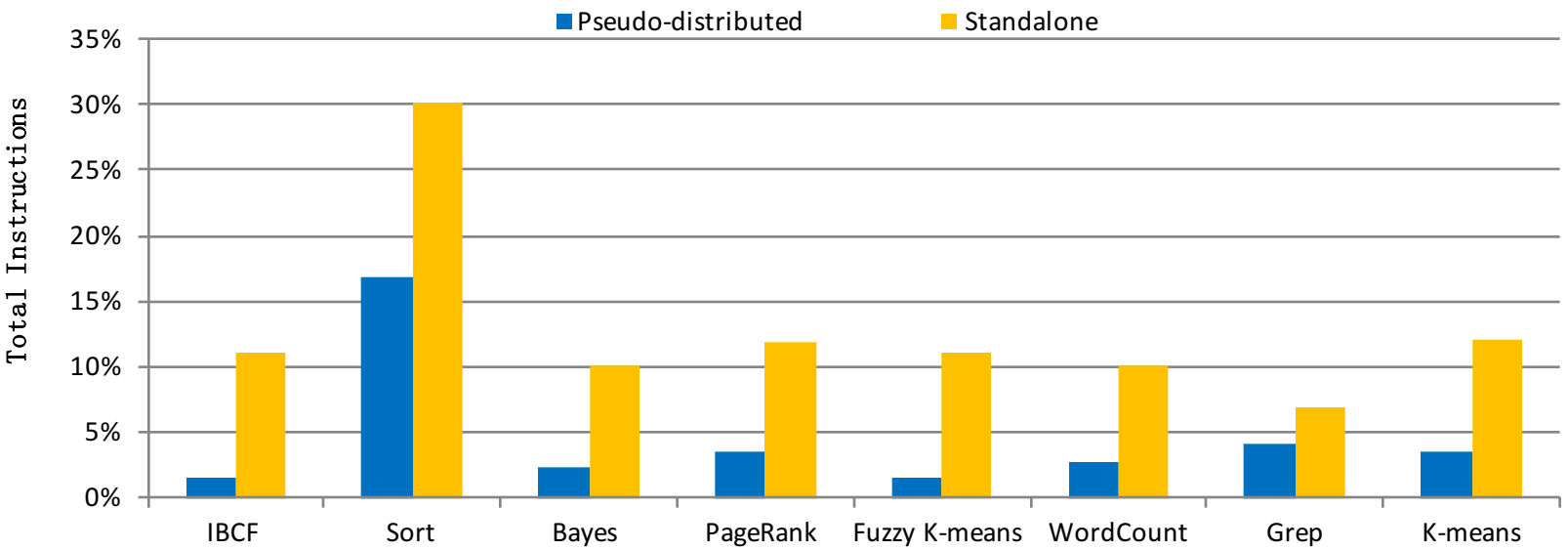
Backend

- Pseudo-distributed workloads have less core bound cycles.
  - The software stack shifts the pressures from backend to frontend (more frontend stall).
  - The software stack diversify the kinds of instructions.



Retired Kernel-mode Instructions

- Pseudo-distributed workloads have less kernel-mode instructions
  - Most software stack functions are implemented in user-mode, which dilutes the kernel-mode instruction ratio.



Further Discussion?

- Contact Zhen Jia
  - jiazhen@ict.ac.cn

Thank You!