

WATSON: A Workflow-based Data Storage Optimizer for Analytics

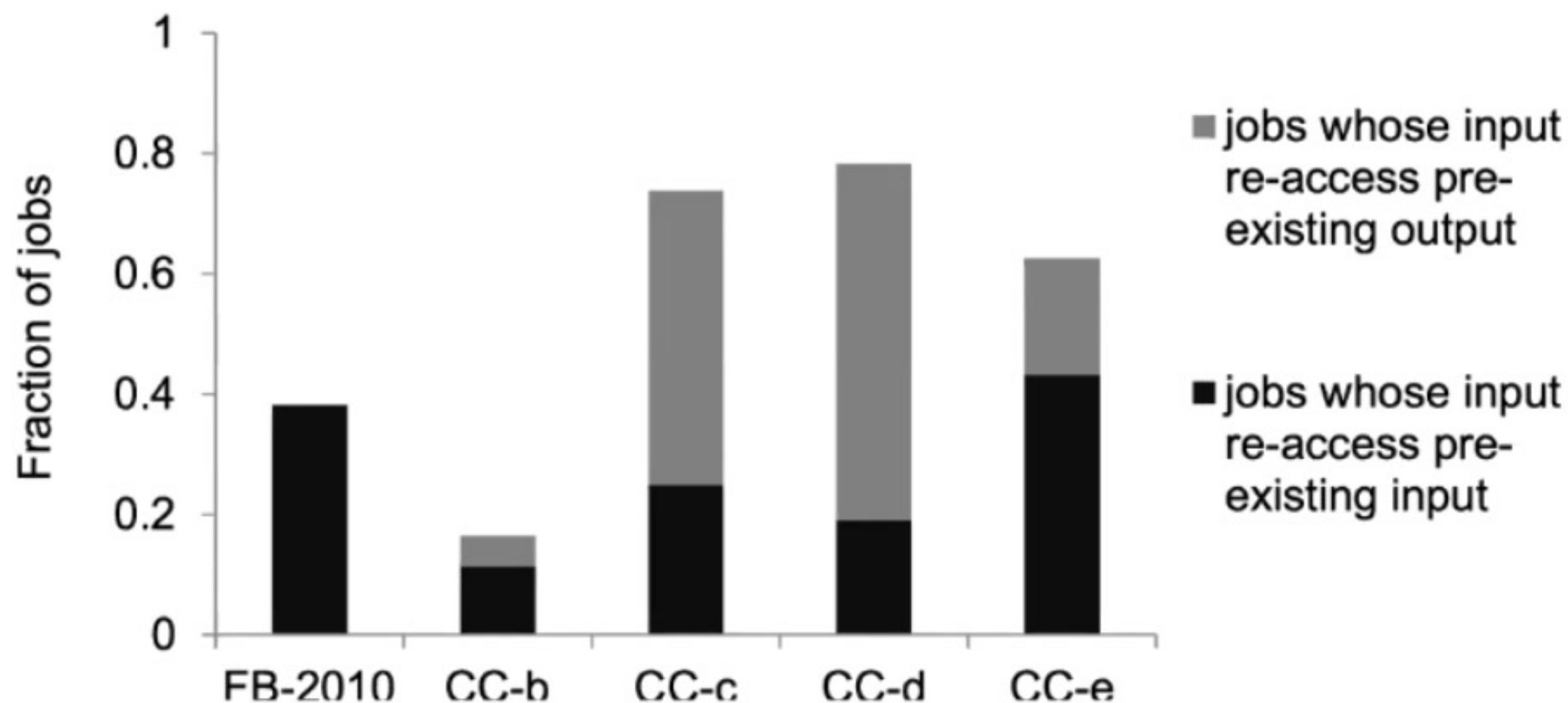
Jia Zou¹, Ming Zhao¹, Juwei Shi², Chen Wang³

¹Arizona State University

²Microsoft Search Technology Center Asia

³Tsinghua University, National Engineering Lab for Big Data Engineering

Motivation: Write Once and Read Many (WORM)



Chen, Yanpei, Sara Alspaugh, and Randy Katz. "Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads." VLDB 2012

Motivation: Write Once and Read Many (WORM)

- Example 1: Data Warehousing
 - Hive
 - SparkSQL
 - Presto

Once data is persisted to DFS, the data may be repeatedly processed by a large number of different jobs for various purposes, e.g. queries and pre-processings.

Motivation: Write Once and Read Many (WORM)

- Example 2: Iterative Analytics

- Applications

- K-Means
 - PageRank
 - Latent Dirichlet Allocation
 - Deep learning

- Algorithms

- EM
 - Gibbs Sampling
 - Alternative Least Square
 - Stochastic Gradient Descent

A lot of machine learning algorithms are iterative-converging in nature. In such iterative analytics workloads, the same input data (i.e. training samples) needs to be repeatedly processed.

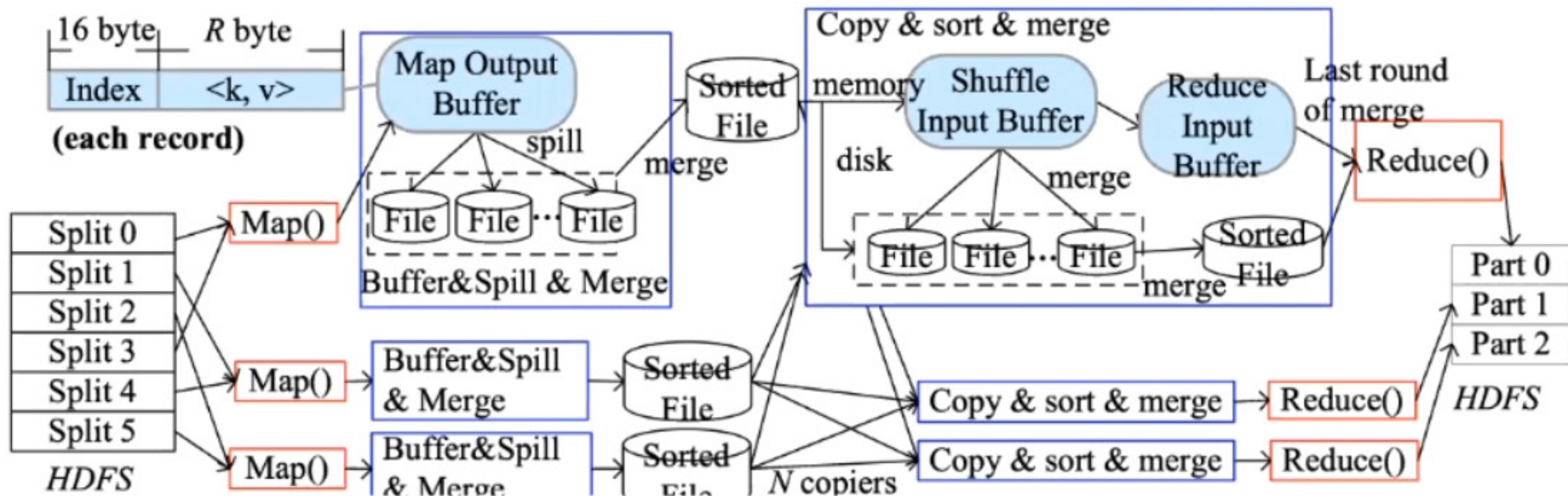
The Problem: Automatic Storage Configuration for Inter-job Optimization

- Storage configuration is still mostly performed through a manual and case by case process
 - The gaps between storage and computation
 - How will the storage parameters affect computations?
 - Which computations will process this dataset in the future?
 - What are the optimal storage configurations required for each computation pattern?
 - Current practices:
 - Reconfiguration: expensive!!
 - MRTuner:
 - single-job auto-tuning, only considers the writing aspect
 - Physical database design, database tuning:
 - rely on relational schemas and functional dependencies

The Ideas:

- Provide an abstraction for storage parameters
- Understand the trade-offs among these parameters and computations
- Automatic storage configuration when we write the data
 - History-based workflow analysis

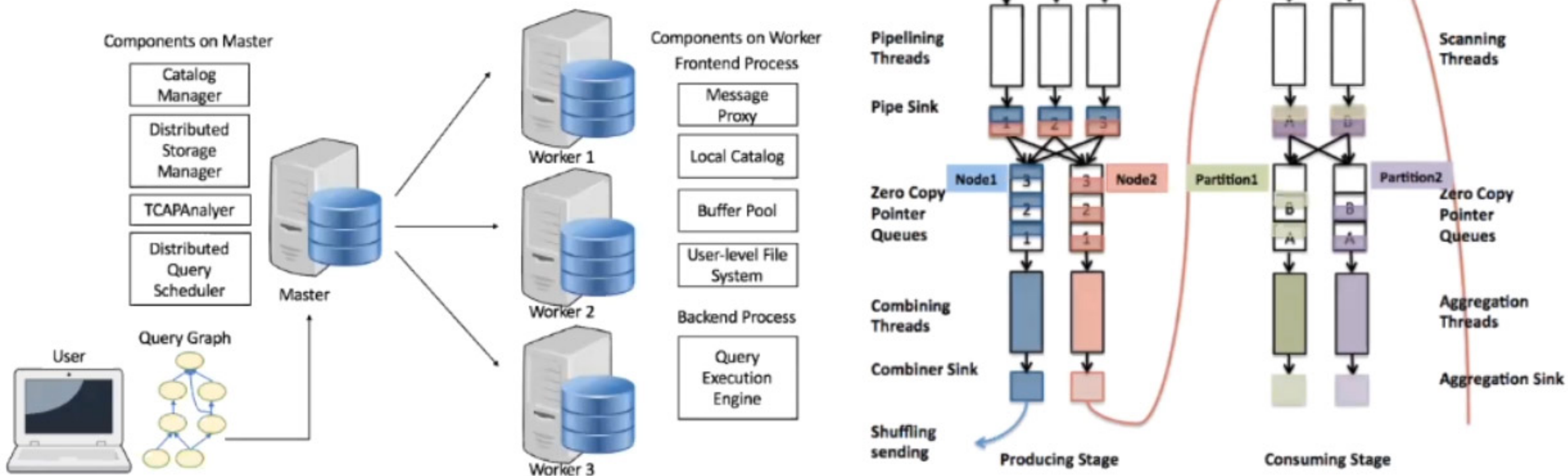
Targeting Applications: MapReduce



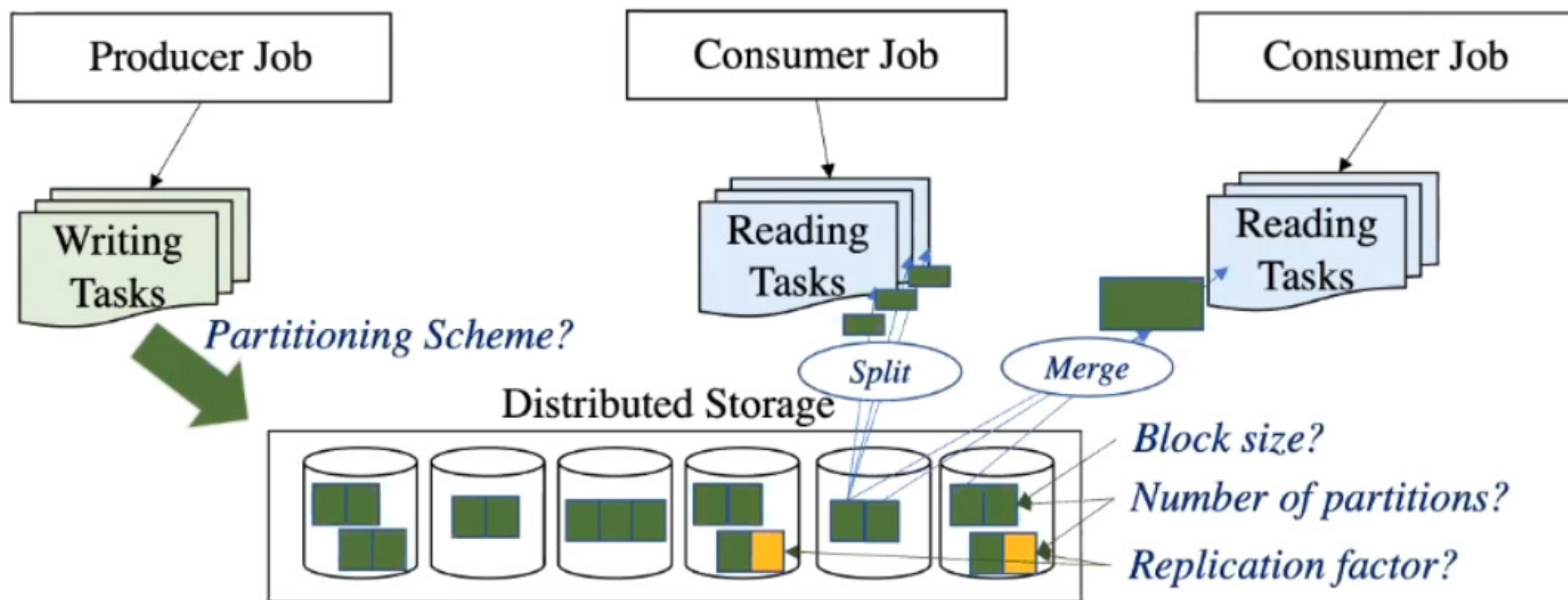
Shi, Juwei, et al. "MRTuner: a toolkit to enable holistic optimization for mapreduce jobs." *Proceedings of the VLDB Endowment* 7.13 (2014): 1319-1330.

Targeting Applications: Distributed Dataflow Platforms

- PlinyCompute



Targeting Storage: Distributed File Systems



Abstraction of Storage Parameters

- Blocking
- Number of partitions
- Partitioning
- Replication

Block Size (B)

- Chunk data into blocks and use block as the unit of data allocation and transfer to save disk seek time.
 - The *block size* parameter can be configured for each analytics job output.
- Influences to computations
 - Importance of task size
 - Intermediate data size > input size
 - ngram
 - Intermediate data size <= input size
 - terasort
 - Inconsistency between block size and task size



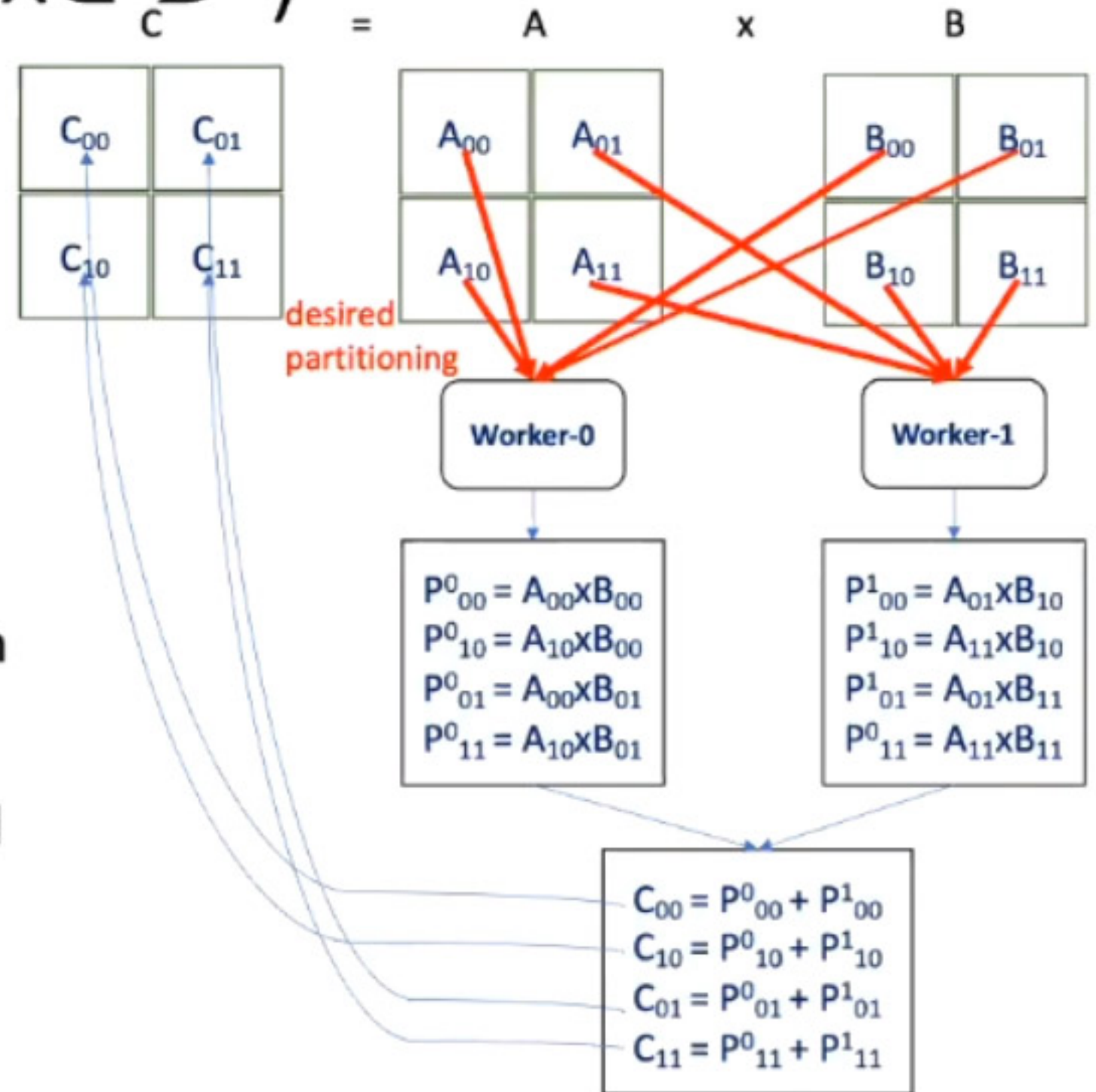
Number of Partitions (p)

- The number of writing tasks determines the number of partitions for the output data.
- It also determines the parallelism and pipelines used to write data to DFS, which influences the writing speed.



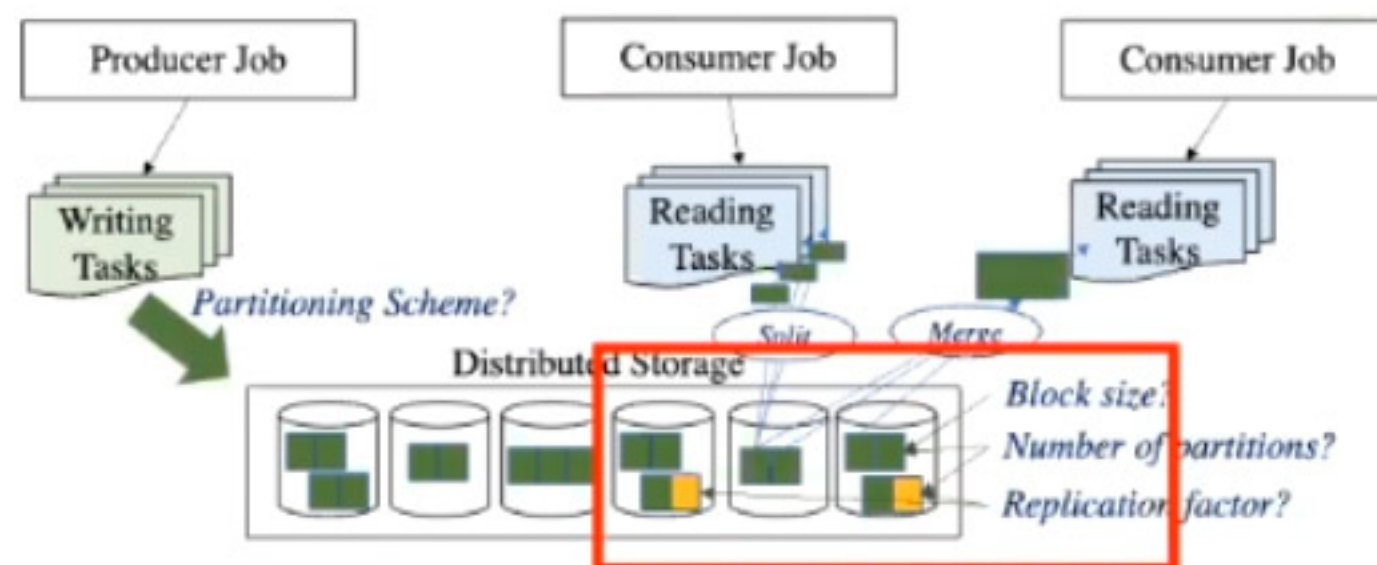
Partitioning Scheme $(f(x), x \in D_c)$

- Partitioning schemes
 - Hash partitioning
 - Range partitioning
 - Random partitioning
 - Round-robin partitioning
- Influences to computations
 - Co-partitioning base on hash or range can save shuffling overhead for join or aggregation operations
 - Round-robin or random is helpful for load balance
 - Random partitioning is helpful for iid distribution



Replication Factor (r)

- Replication factor
 - For each block in a partition, DFS will replicate it to a few nodes across the cluster for resilience.
- Influences to computations
 - More replicas can improve data locality
 - More replicas can save shuffling overhead
 - More replicas can provide better availability
 - More replicas will slow down the writing and increase the storage overhead



The Producer-Consumer Model (1/3)

$$O_{total} = O_{producer} + \sum_{i=1}^n f_i \times O_{consumer_i}$$

The Producer-Consumer Model (2/3)

Producer notations

Symbol	Explanation
Dataset to store	$D = \{x\}$
Dataset size	S
Number of shuffling threads	k
Maximum parallelism	ns

Consumer notations

Symbol	Explanation
s_i	The i -th consumer's task split size
g_i	The i -th consumer's desired partitioning function
Maximum parallelism	k

Storage Parameter notations

Symbol	Explanation
Block size	B
Number of partitions	p
Partitioning scheme	$f(x), x \in D$
Replication factor	r

Overhead notations

Symbol	Explanation
o_p	Unit per-object partitioning communication overhead
o_f	Unit per-object partitioning computation overhead for $f(x)$
o_i	Unit per-block writing overhead
o_w	Unit per-byte writing overhead
o_t	Unit per-byte transmission overhead

The Producer-Consumer Model (3/3)

$$O_{total} = O_{producer} + \sum_{i=1}^n f_i \times O_{consumer_i}$$


$$O_{producer} = O_{partition} + O_{writing}$$

The Producer-Consumer Model (3/3)

$$O_{total} = O_{producer} + \sum_{i=1}^n f_i \times O_{consumer_i}$$

$$O_{producer} = O_{partition} + O_{writing}$$

$$O_{partition}(x) = \begin{cases} o_p & \text{host}(x) \neq f(x) \\ 0 & \text{host}(x) = f(x) \end{cases}$$

$$O_{partition} = \underbrace{\sum_{x \in D} O_{partition}(x)}_{\text{IO overhead}} + \underbrace{o_f \times \frac{ns \times S}{k} \times \lceil \frac{k}{ns} \rceil}_{\text{computation overhead}}$$

The Producer-Consumer Model (3/3)

$$O_{total} = O_{producer} + \sum_{i=1}^n f_i \times O_{consumer_i}$$


$$O_{producer} = O_{partition} + O_{writing}$$


$$O_{writing} = \frac{o_i \times \frac{S}{B} \times r + o_w \times S \times r}{p} \times \lceil \frac{p}{ns} \rceil$$

The Producer-Consumer Model (3/3)

$$O_{total} = O_{producer} + \sum_{i=1}^n f_i \times O_{consumer_i}$$

$\downarrow$

$$O_{consumer_i} = O_{repartition_i} + O_{input_i}$$

The Producer-Consumer Model (3/3)

$$O_{total} = O_{producer} + \sum_{i=1}^n f_i \times O_{consumer_i}$$

$$O_{consumer_i} = O_{repartition_i} + O_{input_i}$$

$$O_{input_i} = o_{input_i} \times nw = \frac{\sum_{j=1}^{\frac{S}{s_i}} o_t \times (p_j \times s_i)}{\frac{S}{s_i}} \times \left\lceil \frac{\frac{S}{s_i}}{ns} \right\rceil$$

The Derived Rules (1/4)

- Consistent Partitioning Scheme Rule.
 - O_{consumer_i} can be minimized only if the partitioning scheme of the input data is consistent with the desired partitioning scheme of the i -th consumer job. (i.e., $f = g_i$).
 - If the data is already partitioned as desired, each object resides on the right host, and $O_{\text{repartition}_i} = 0$.

The Derived Rules (2/4)

- Aligned Split and Block Size Rule
 - O_{consumer_i} can be minimized only if block size is equal to split size (i.e. $s_i = B$).

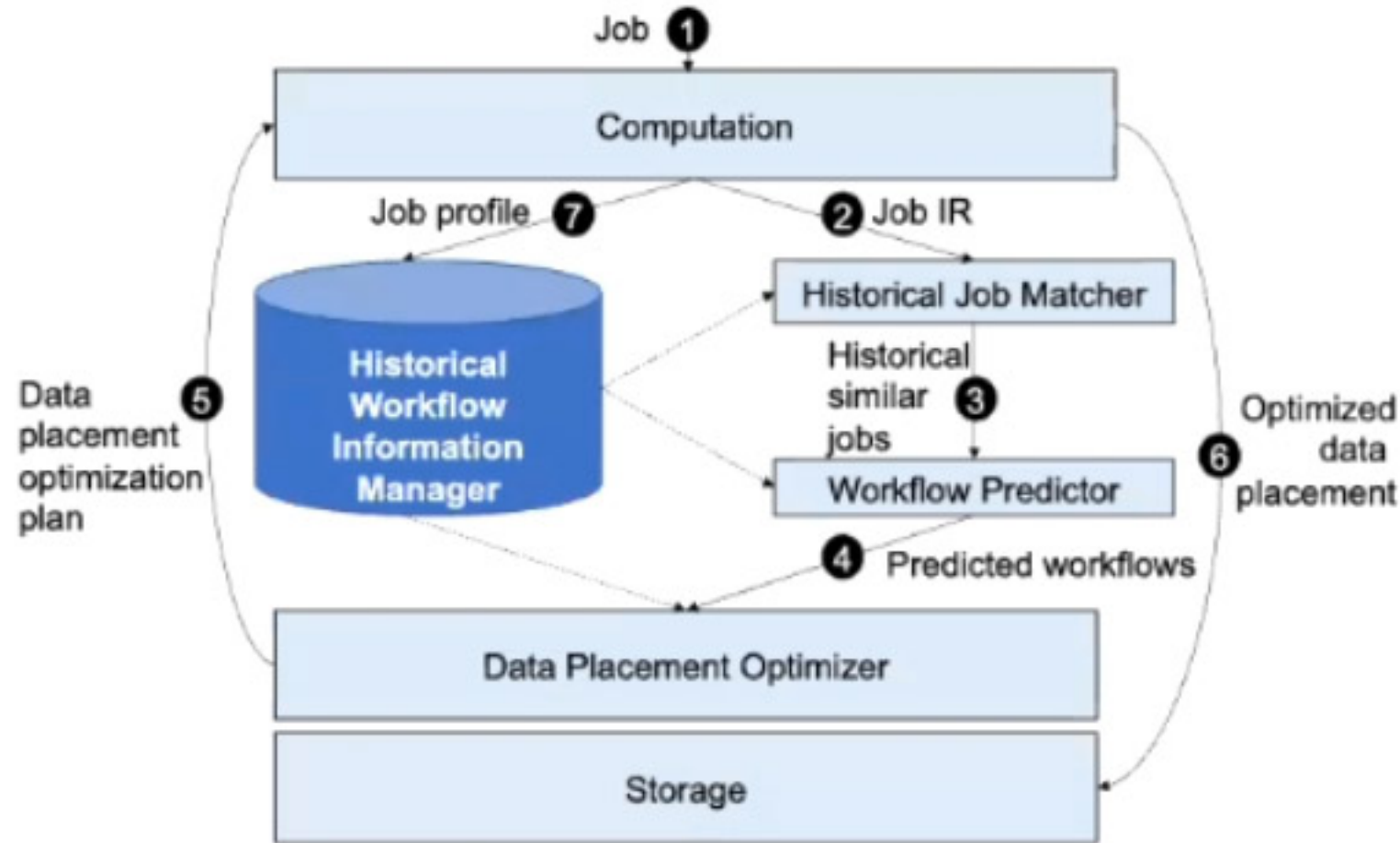
The Derived Rules (3/4)

- Proportional Data Distribution Rule.
 - O_{consumer_i} can be minimized only if the size of data stored in each node is proportional to the number of slots in each node
 - i.e. $\forall \text{node } j, ns_j / ns = nb_j / nb$, where nb represents total number of blocks and nb represents number of blocks on the j -th node

The Derived Rules (4/4)

- Invariant Replication Rule.
 - If for a consumer job, the storage block size is aligned with task split size and input data is distributed proportionally to slot number for each node in the cluster there will be $O_{\text{input}_i} = 0$, regardless of the replication factor.
 - tuning replication factor can not further improve the performance of this particular consumer, if other rules are satisfied.

System Architecture



Algorithm 1 Greedy Strategy with Multi-way Block-level Replication.

Output: Block Size: B ; Partition Number: np ; Partitioning Scheme: f ; Replication factor: r

```

1:  $dct \leftarrow searchIndexOfMostFrequentCustomerType()$ 
2: if  $s_{dct} \times ns > S$  then
3:    $B \leftarrow S/ns$ 
4: else
5:    $B \leftarrow s_{dct}$ 
6: end if
7:  $f \leftarrow g_{dct}$ 
8:  $nw \leftarrow \frac{S}{mb \times ns}$ 
9:  $np \leftarrow nw \times ns$ 
10:  $O_{min} \leftarrow \infty$ 
11:  $r_{optimal} \leftarrow r_{default}$ 
12: if  $n > 1$  then
13:   for each  $r_{default} \leq r \leq r_{max}$  do
14:      $O \leftarrow O_{producer} + \sum_{i=1}^n f_i \cdot O_{consumer_i}$ 
15:     if  $O < O_{min}$  then
16:        $O_{min} \leftarrow O$ 
17:        $r_{optimal} \leftarrow r$ 
18:     end if
19:   end for
20:    $r \leftarrow r_{optimal}$ 
21: end if
22: return

```

Experiment Environments: MapReduce

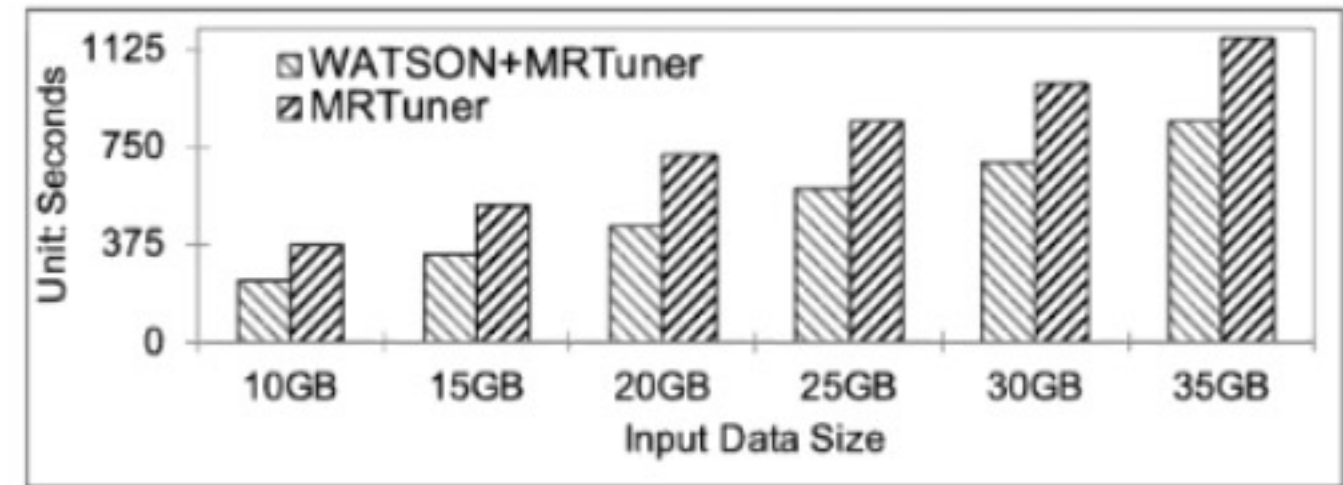
- Hadoop 1.1.1
- A cluster that consists of ten HS21 blade servers connected by 10Gb Ethernet. Each of the nodes has four single core 3.6GHz processor, 4GB memory, and 130GB disk drive.
- The slot number is set to four, and all other parameters are configured by MRTuner.

Experiment Environments: Distributed Dataflow

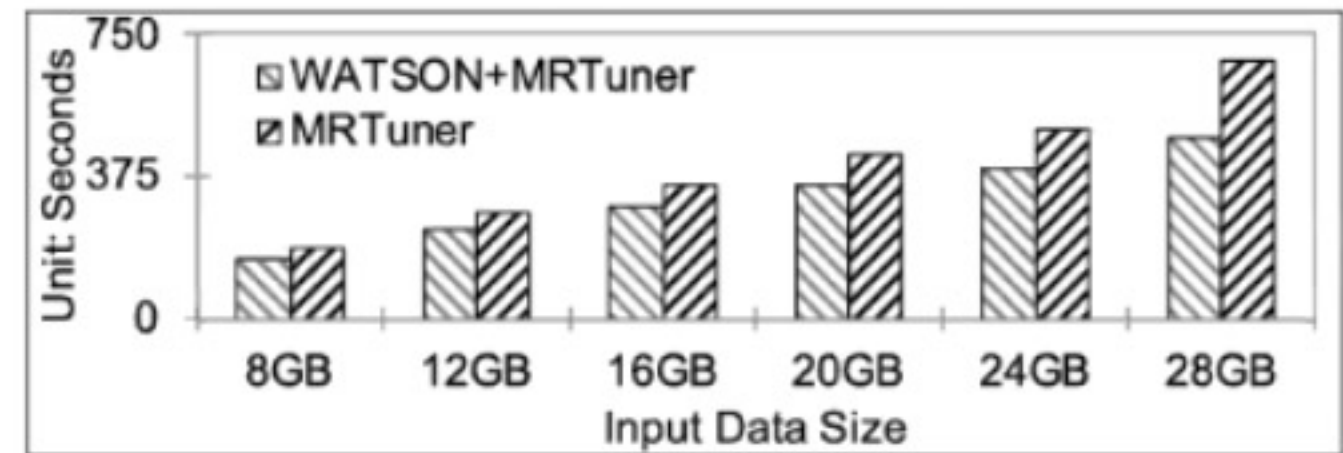
- PlinyCompute, Spark
- A cluster consists of ten AWS r4.2xlarge instances on AWS. Each instance consists of eight cores, 61GB memory, and 100GB CPU cores.
- We tune the number of slots to eight, and the buffer pool size to fifty gigabytes.

Hadoop Results (1/3)

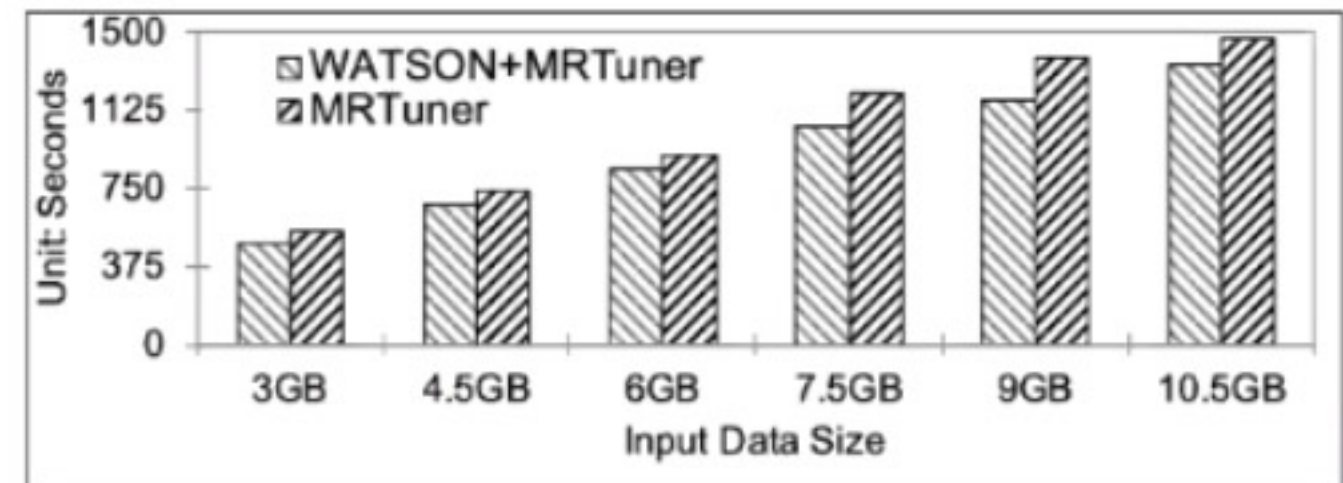
- Significant performance gain by simply tuning the block size and number of partitions



(a) Terasort



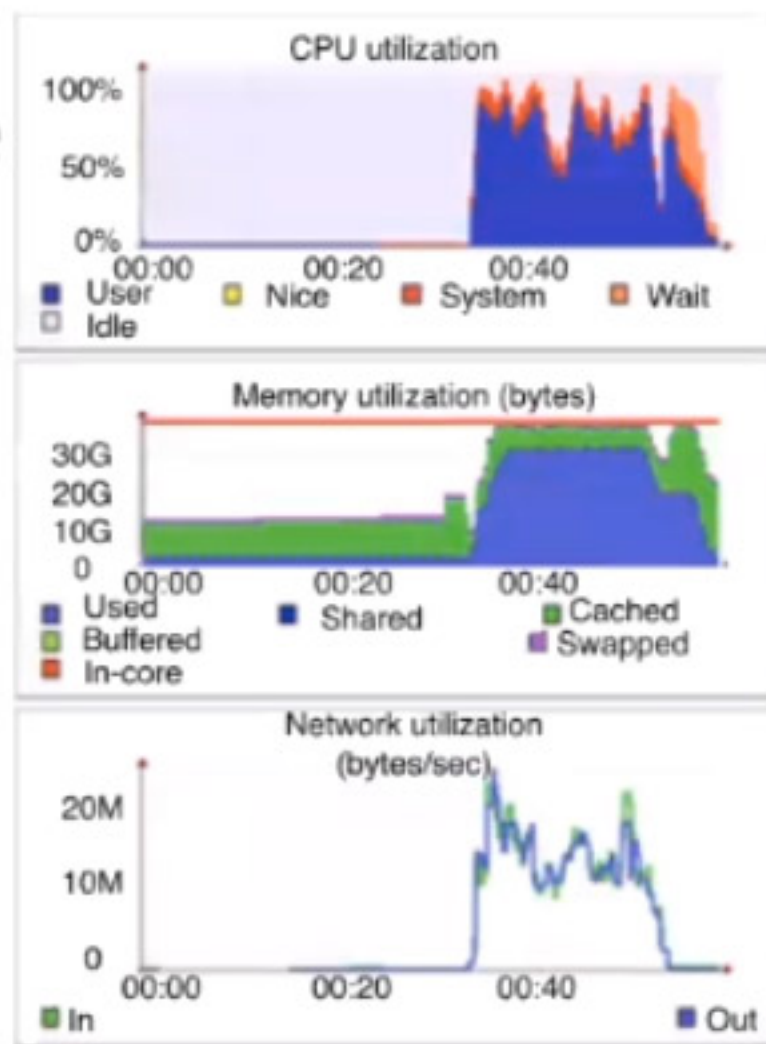
(b) KMeans



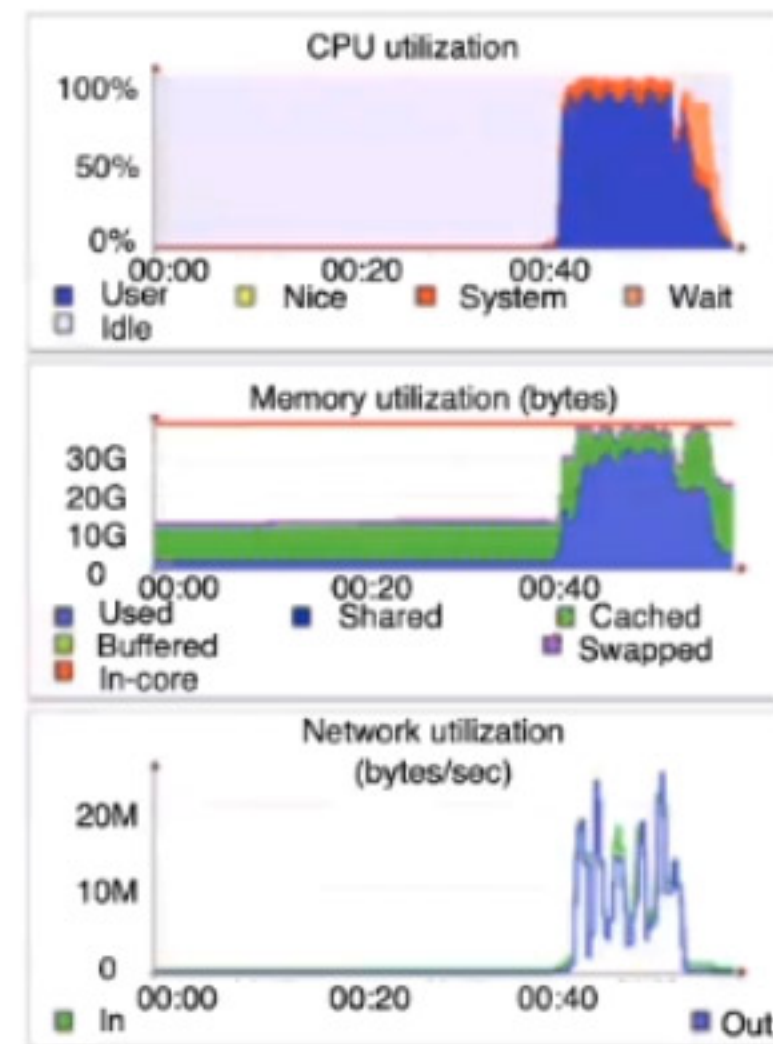
(c) NutchIndex

Hadoop Results (2/3)

- Profiling of CPU, memory, and network utilization of the cluster, before and after storage parameter tuning of the Terasort workload.



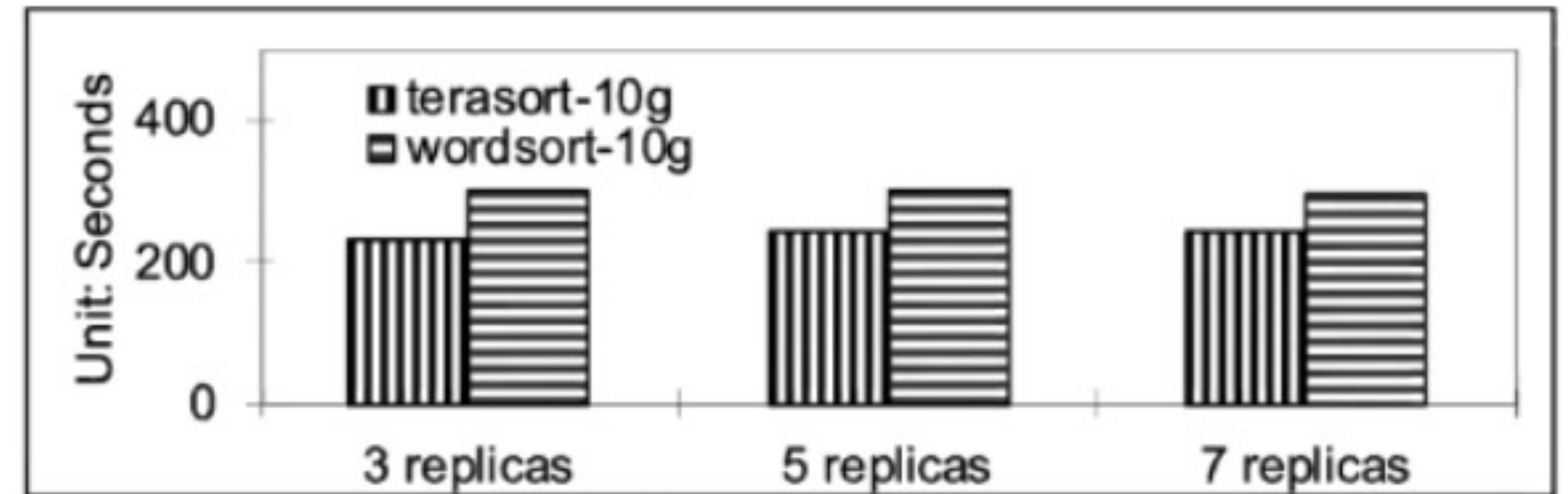
(a) Before Tuning



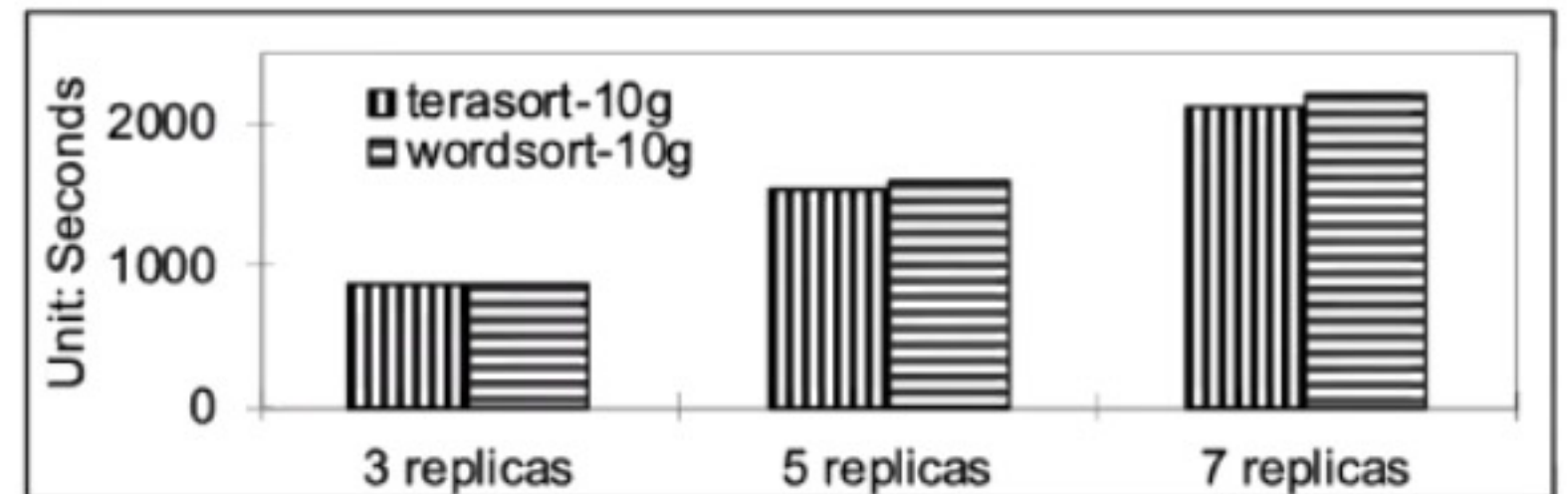
(b) After Tuning

Hadoop Results (3/3)

- Replication has no effect, if the block size and number of partitions are fully tuned.



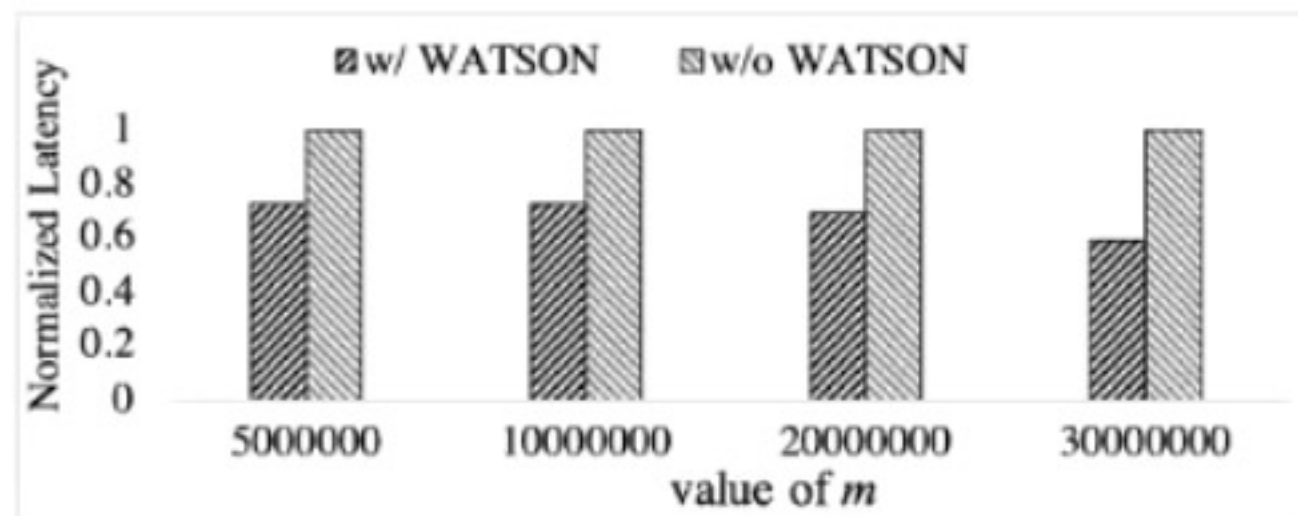
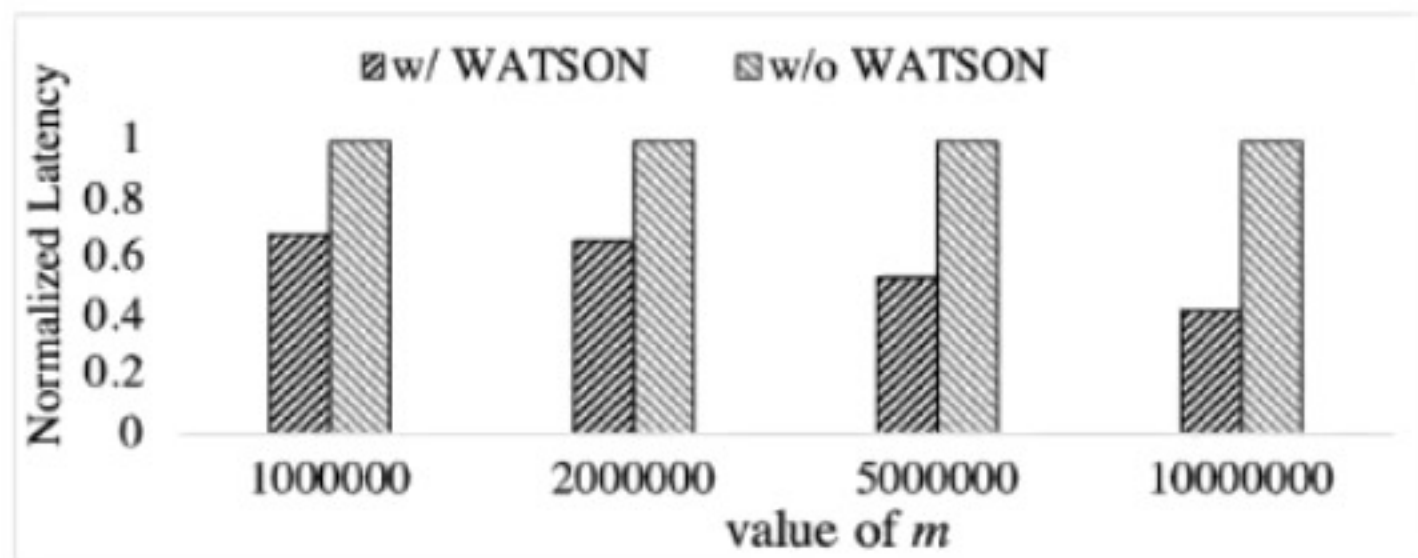
(a) Consumer performance



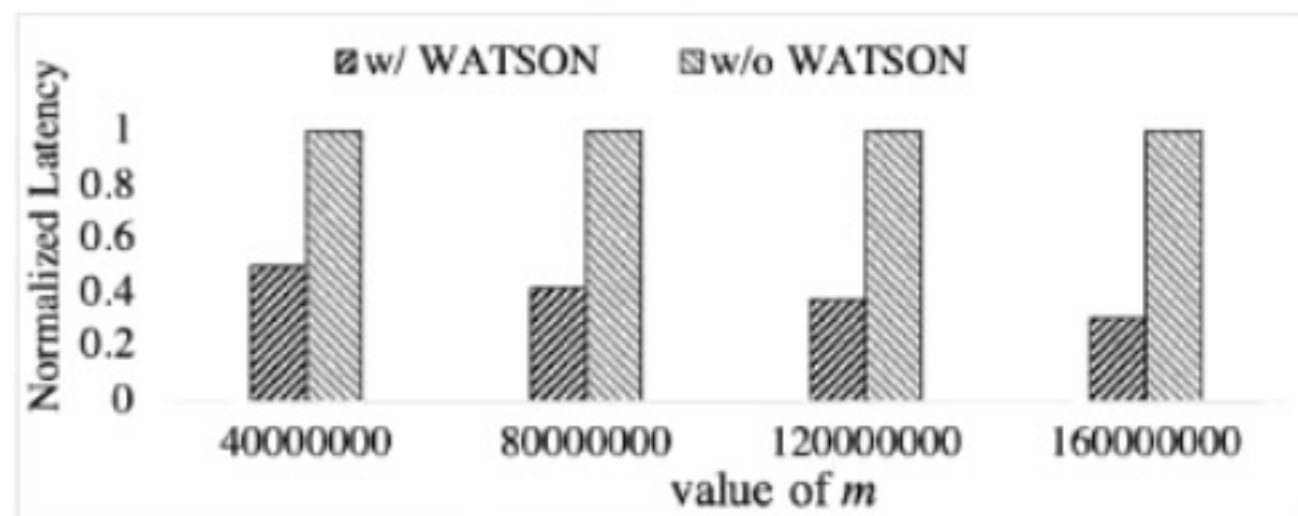
(b) Producer performance

PlinyCompute Results (1/3)

- Matrix Multiplication: Up to more than two times speedup.



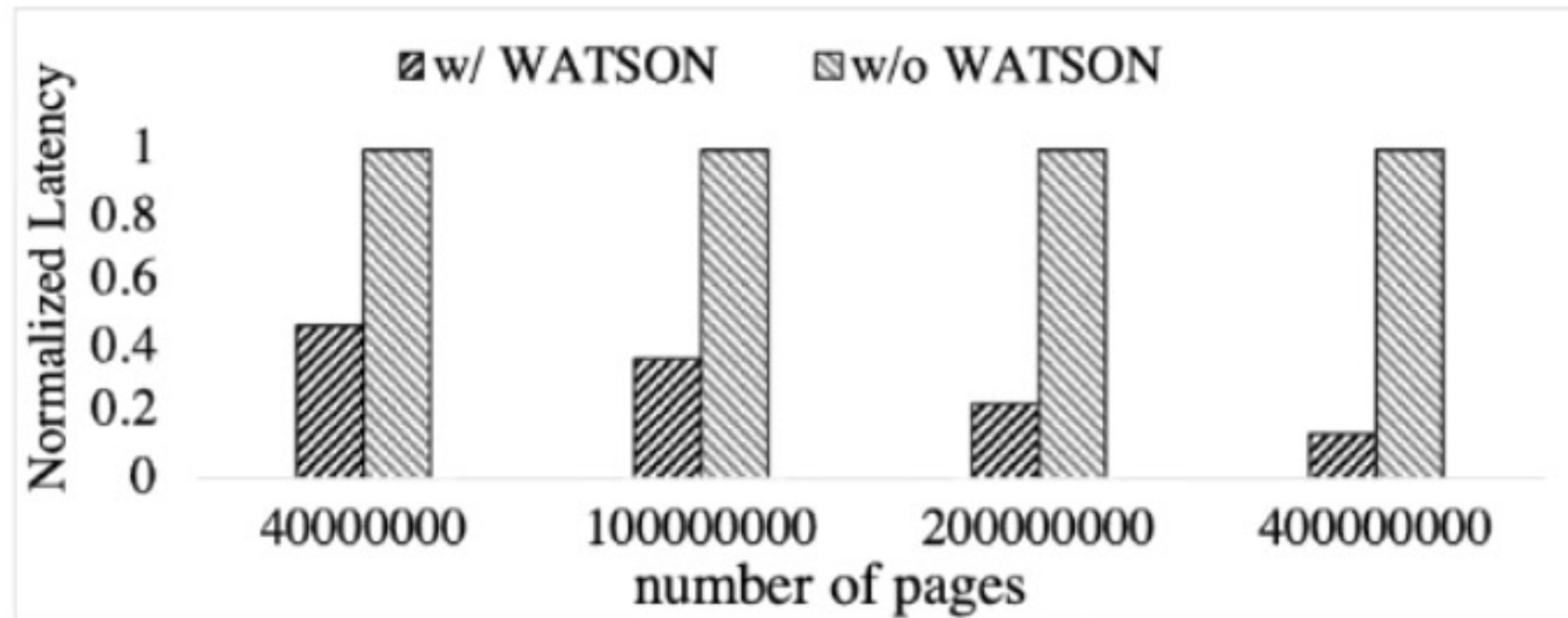
(a) sparsity=0.001



(b) sparsity=0.000001

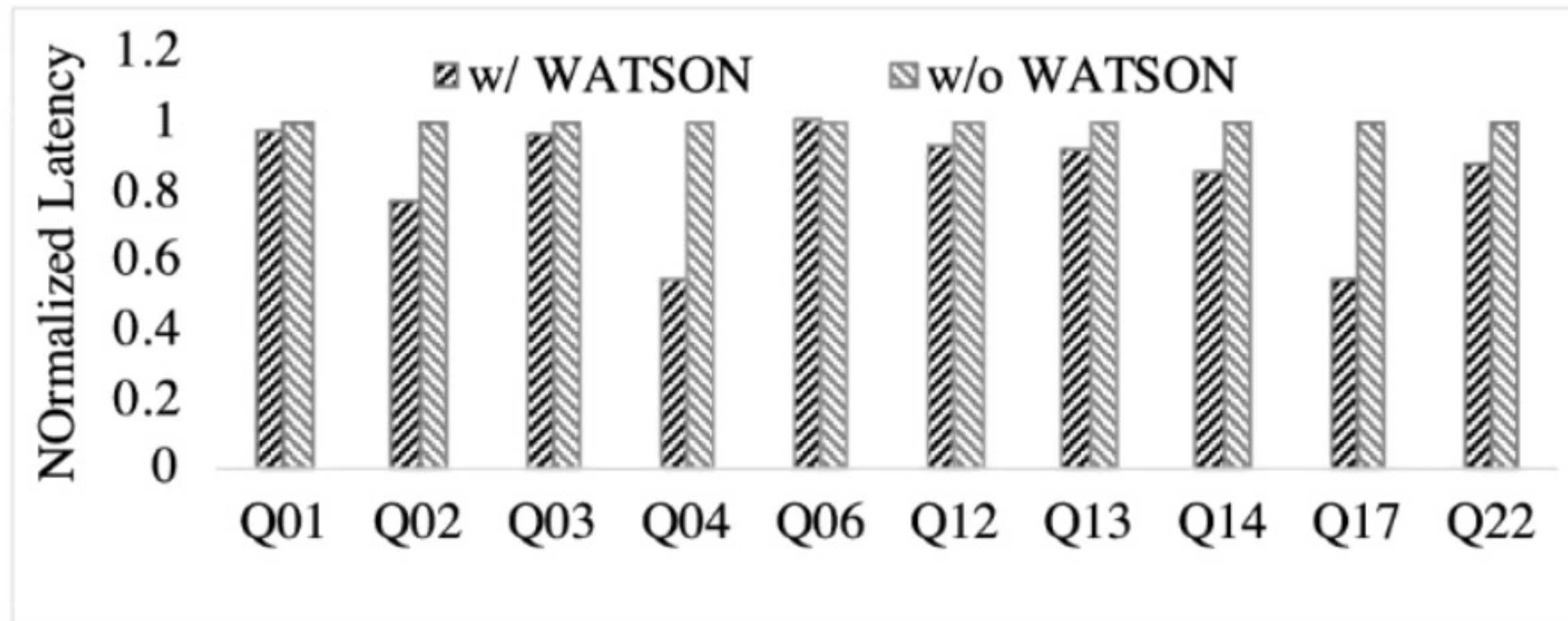
PlinyCompute Results (2/3)

- PageRank: More than six times speedup.



PlinyCompute Results (3/3)

- Data warehouse (TPC-H): Up to two times speedup.



Spark Results

- We didn't implement WATSON on Spark, but we observed 3x speedup in a top-K aggregation query on TPC-H by simply
 - Configuring the optimal split size by setting "spark.hadoop.mapreduce.input.fileinputformat.split.minsize= 2147483648" and "spark.hadoop.mapreduce.input.fileinputformat.split.maxsize=2147483648" in the Spark job configuration.

Conclusions

- We provide an abstraction of storage parameters on Distributed File Systems for MapReduce and Dataflow workloads
- We model the problem using a producer-consumer model, from which we derive four useful rules
- Based on these rules and history-based workflow analysis, we design an automatic parameter tuning system and implement it on Hadoop and PlinyCompute
- The evaluation results validate the derived rules and demonstrate the effectiveness of our proposed approach

Thanks!