Geo-Distributed Big Data Processing

Patrick Eugster

Outline

- Big data background
- Geo-distribution motivation
- Geo-distributed tasks
- Geo-distributed workflows
- Conclusions and outlook

Outline

- **Big data background**
- Geo-distribution motivation
- Geo-distributed tasks
- Geo-distributed workflows
- Conclusions and outlook

Big Data

- Large datasets ranging from hundreds of GBs to hundreds of TBs (for most users) and even PBs for large corporations [Wikipedia]
  - Often GB range [Schwarzkopf et al.;HotCloud'12]
- Too large for traditional relational database tools and single nodes to handle
- Processed using data-parallel software running tens, hundreds, even thousands of computers

Big Data - Why ?

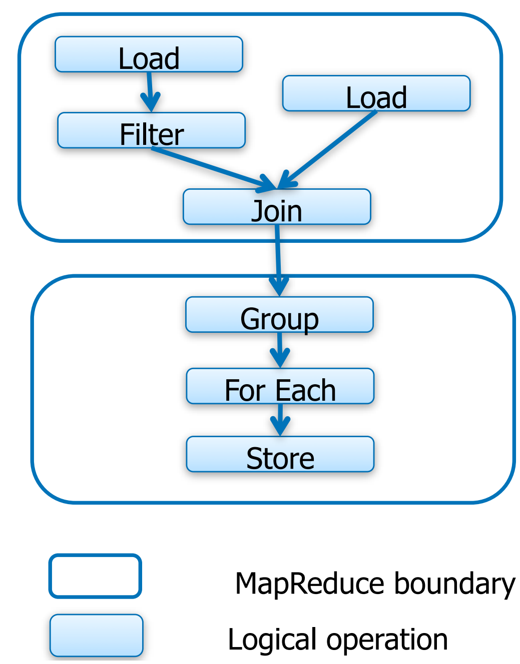
- We need it
  - More users connected to the Internet: “*Everyone on earth will be connected to the Internet by 2020*” [E. Schmidt’13]
- We want it
  - Applications use large datasets, e.g., for operation, monitoring, auditing, knowledge extraction
- Because we can
  - Large amounts of cheap “cloud” storage available: “*Amazon S3 contains over 449 billion objects and during peak time, processes more than 290K requests per second*” [AWS blog’11]

Processing Big Data

- MapReduce (MR) popularized by [Dean and Ghemawat;OSDI'04]
  - Inspired by functional programming
  - Consists of two phases
    - `map` - takes input records and outputs sets of <`key`, `value`> pairs
    - `reduce` - handles set of `value`s for given `key`s and emits sets of `value`s
  - Open source Apache Hadoop
- HDFS distributed file system inspired by Google's GFS [Ghemawat et al.;SOSP'03]

Workflow Programming

- Many “high-level languages” proposed, e.g.,
  - Pig Latin [Olston et al.;SIGMOD’08]
    - (Mostly) declarative untyped scripting language
    - Open source Apache Pig
  - Flume Java [Chambers et al.;PLDI‘10]
    - Java library
    - Open source Apache Crunch
- Many compile to MR



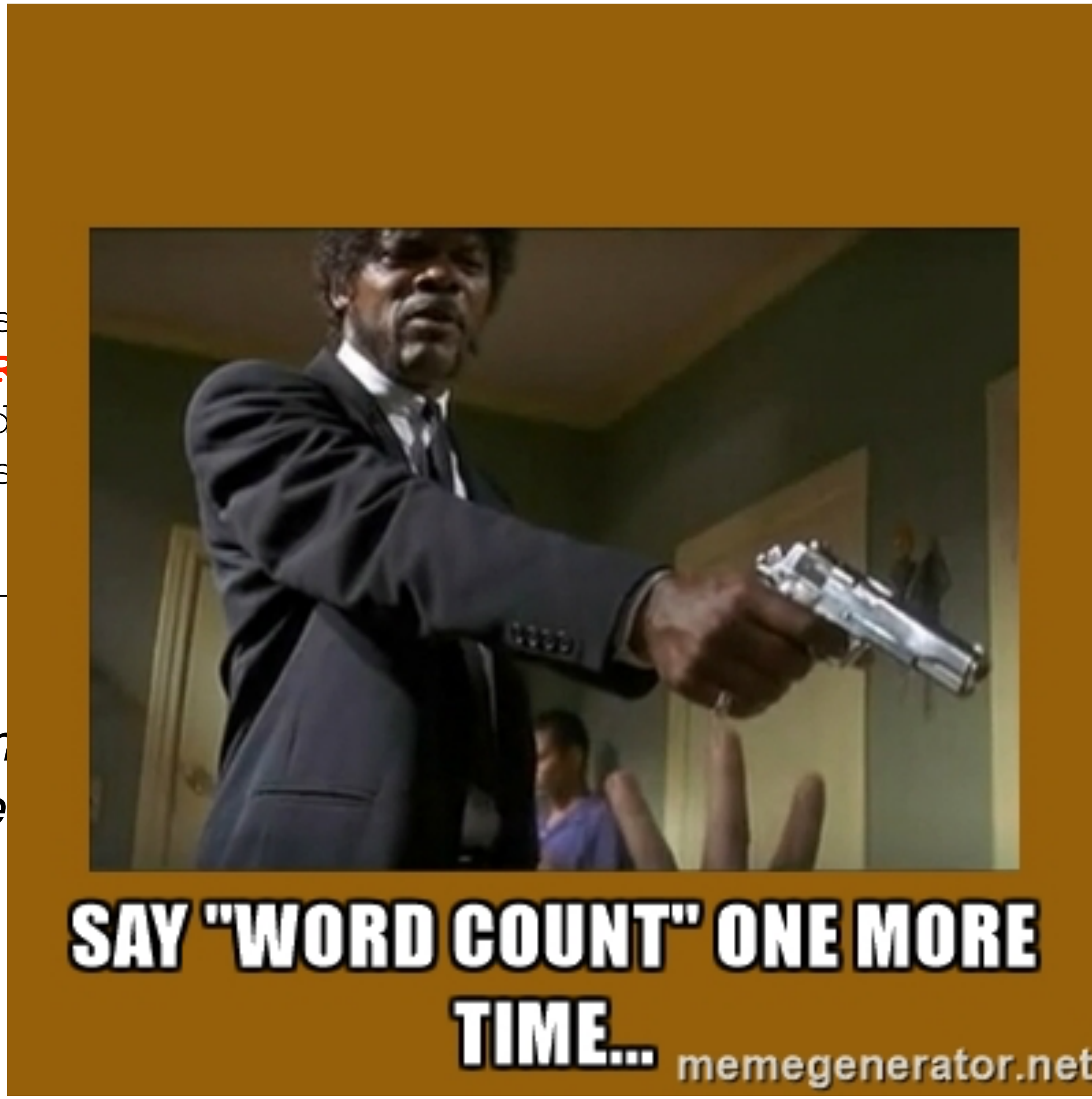
Pig Latin Example

"Word count"

```
input_lines = LOAD 'input_file' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line))
    AS word;
word_groups = GROUP words BY word;
word_count = FOREACH word_groups GENERATE group, COUNT(words);
STORE word_count INTO 'output_file';
```

"*Yahoo estimates that between 40% and 60% of its Hadoop workloads are generated from Pig* [...] *scripts. With 100,000 CPUs at Yahoo and roughly 50% running Hadoop, that's a lot* [...]" [IBM DeveloperWorks'12]

Pig Latin Example




```
input_lines                                        ;
words = FOR                                      ZE(line))
    AS word
word_groups
word_count                                       UNT(words);
STORE word_
```

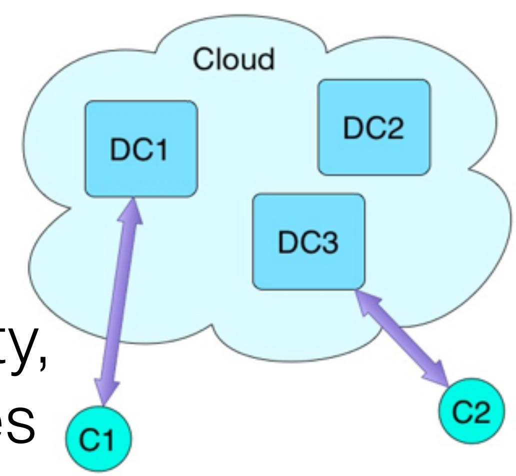
"Yahoo estim … *workloads*
are generate … *ahoo and*
roughly 50% … erWorks'12]

Outline

- Big data background
- **Geo-distribution motivation**
- Geo-distributed tasks
- Geo-distributed workflows
- Conclusions and outlook

Geo-Distributed Big Data

- Many large datasets geo-distributed, i.e., split across sites
  - Stored near resp. sources, frequently accessing entities
  - Gathered and stored by different (sub-)organizations yet shared towards a common goal
    - E.g., US census, Google "buckets"
  - Replicated across datacenters for availability, incompletely to limit the overhead of updates



Geo-Distributed Big Data

- Many analysis tasks involve several datasets, which may be distributed
  - Legal constraints may confine certain datasets to specific locations
- The “cloud” is not a single datacenter
- Inter-DC latency ≠ intra-DC latency

Concrete Scenario

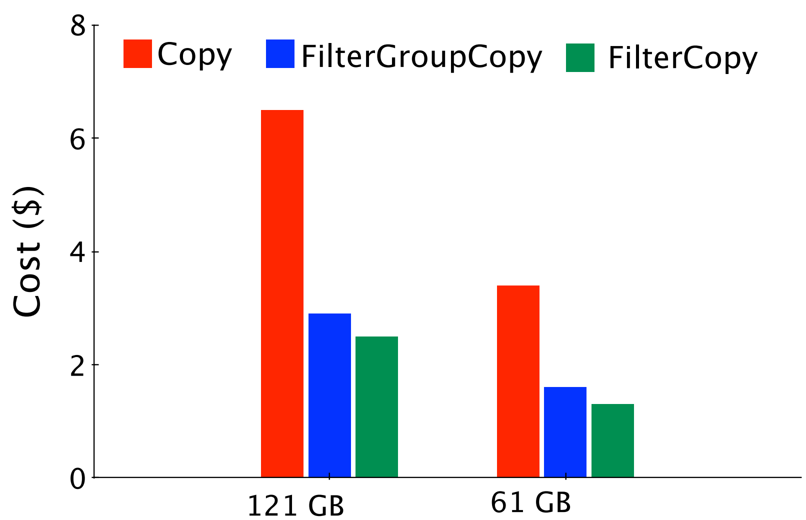
- Global web-based service provider
  - Serve customers from close-by datacenters
    - “Regional” customer bases
  - Run analyses across all regions
    - E.g., average age of customers buying product *x*

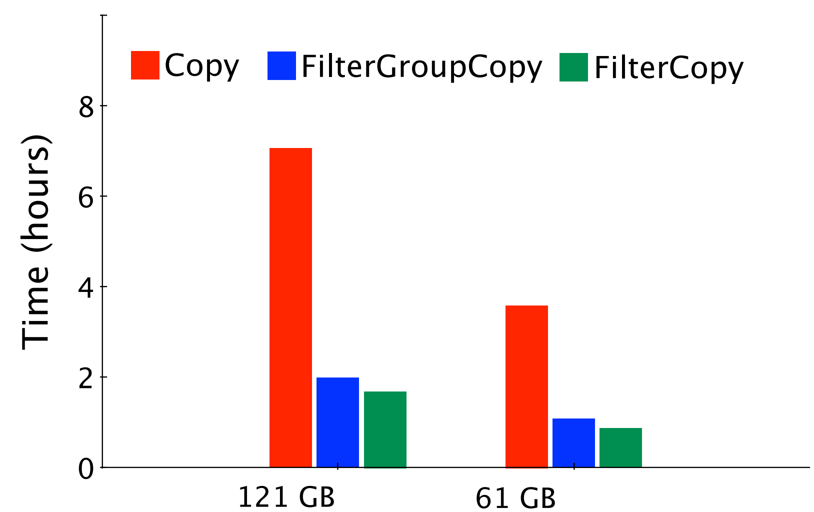
GD in current Toolchain

- Hadoop
  - Assumes uniform latencies
  - Reducer placement based on resource availability
  - Data must be in one HDFS instance or S3 bucket
- HDFS
  - Single point of management (namenode)
  - Performs poorly with high and/or inconsistent latencies
- Pig Latin, Flume Java et al.
  - Inherit weaknesses of underlying systems
  - No support for expressing distribution

Potential for Improvement

- Conjecture: poor execution choices result in high costs/delays
- E.g., US Census 2000 data (121 GB), 2 Amazon EC2 datacenters, MapReduce cluster of 10 nodes each
- Two tasks (MR jobs) (1) filter records (2) group records
    - Associative: can execute on subsets of data and then aggregate





State of the Art

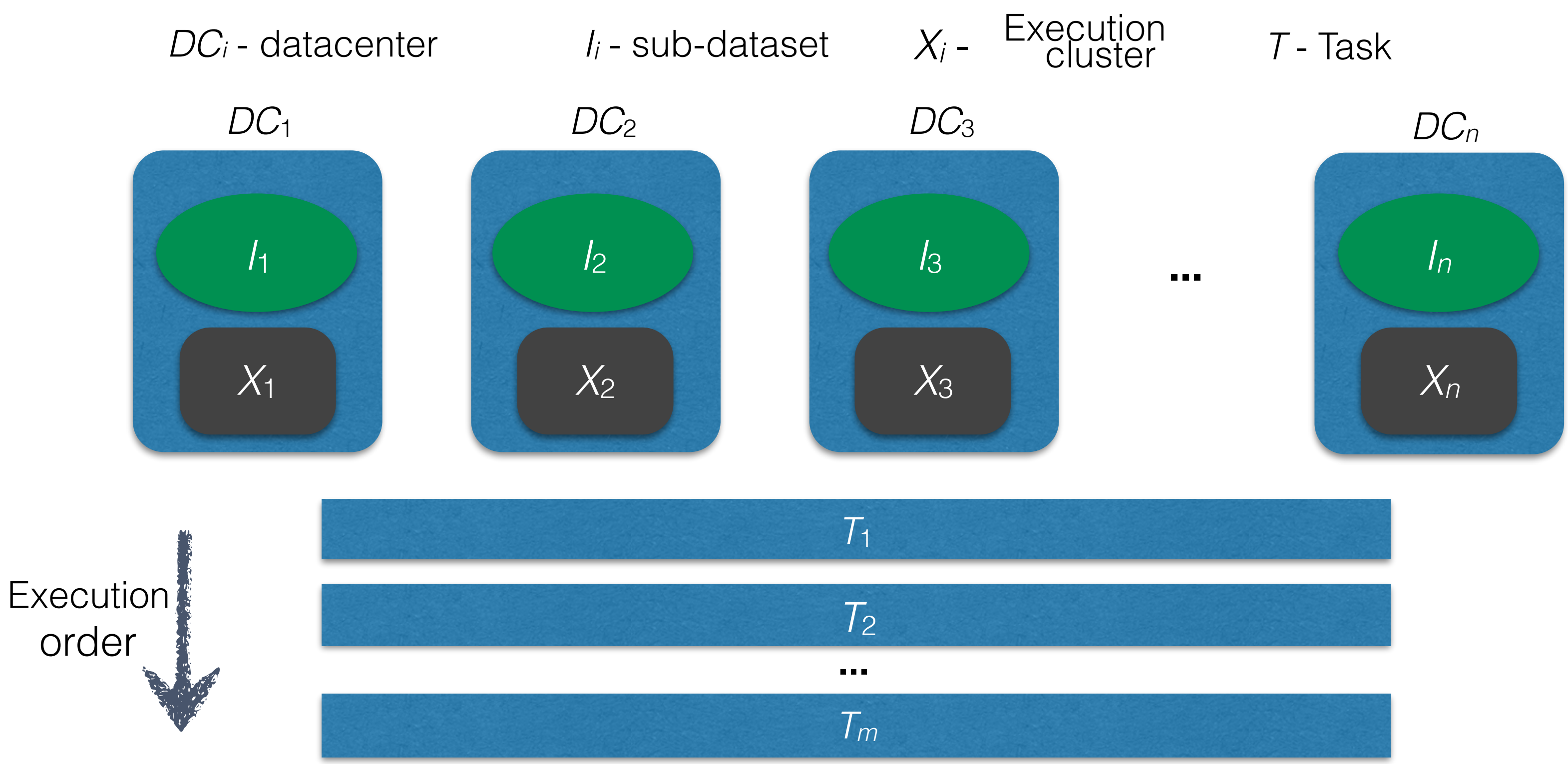
- **GD storage**: Many systems, e.g., [Lloyd et al.;SOSP'11], [Sovran et al.;SOSP'11], [Cho&Aguilera;ATC'12],[Sciasica&Pedone;DSN'13], [Zhang et al.;SOSP'13], consider GD data reads&writes.
- **GD data location**: Volley [Agraval et al.;NSDI'10] or [Tran et al.;ATC'11] migrate GD big data based on application needs.
- **GD computation**: HOG [Weitzel et al.;MTAGS'12] modifies Hadoop for Open Science Grid. Focus on site failures, not performance. G-Hadoop [Wanga et al.;Future Gen. Comp. Systems'13] similar.
- **(G)D programming**: Flink [Ewen et al.;PVLDB'12], Presto [Venkataraman et al.;HotCloud'12], Spark [Zaharia et al.;NSDI'12] support distributed datastructures but still in single datacenter.

Outline

- Big data background
- Geo-distribution motivation
- **Geo-distributed tasks**
- Geo-distributed workflows
- Conclusions and outlook

GD Tasks
[Jayalath&Eugster;IEEE TC'14]

- Dataset *I* distributed across *n* datacenters (DC_1 to DC_n), each has execution cluster
- Sequence of *m* tasks T_1 to T_m (cf. transducers)



Problem Statement

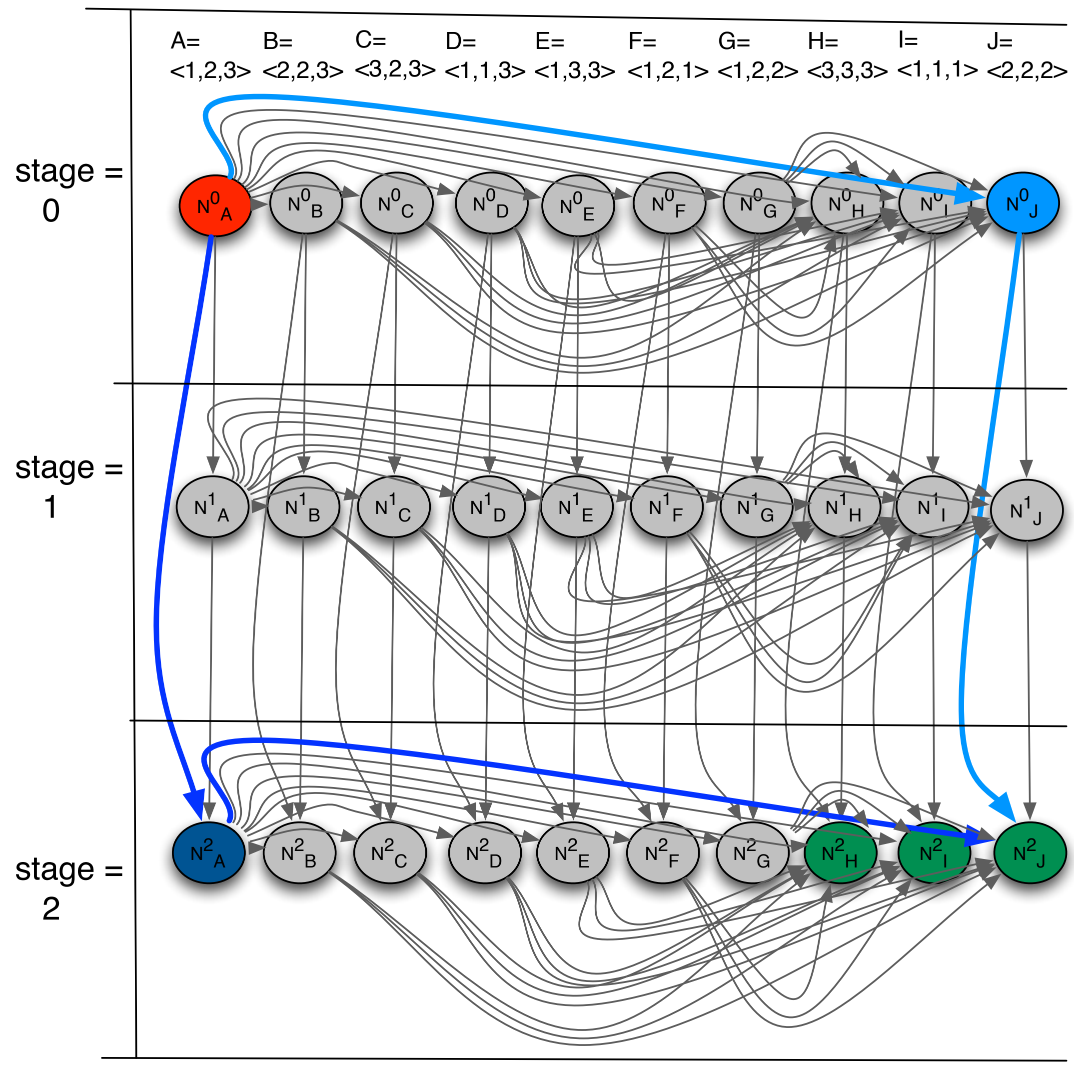
- How to efficiently perform a task sequence on a GD dataset?
- Several solutions varying by consolidation point, e.g., MR:
  - Copy all data to 1 datacenter, perform job
  - Perform mapping in respective datacenters, allocate all reducers in 1 datacenter
  - Perform mapping and reducing in respective datacenters, aggregate subsequently (assuming “associativity”)
- Combinations, e.g., consolidate input from 2 of 3 datacenters, perform mapping individually, then reducing in 1 datacenter

Data Transformation Graphs (DTGs)

- A node - distribution of data and the task execution progress
- Weight of an edge - cost (monetary) or time for performing a task or a copy operation
- Each path from a starting node to an end node is a possible **execution path**
- A shortest path calculation algorithm is used to determine the optimal path
- Optimal with respect to a given partition distribution and other parameter values

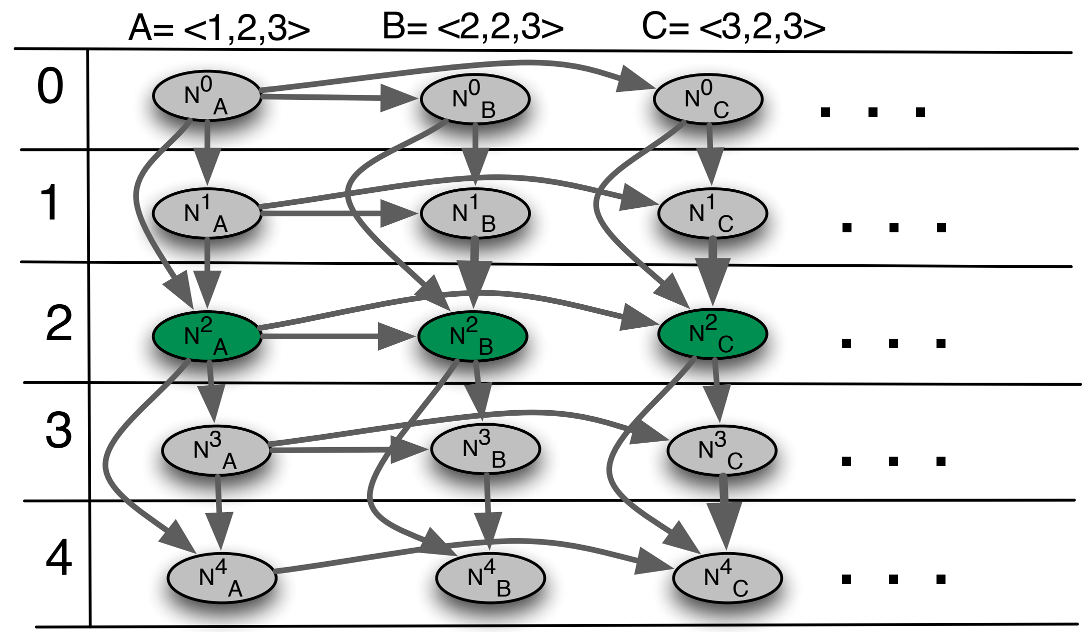
DTGs by Example

- 3 datacenters, 1 input partition in each, 1 MR job (2 tasks - `map` and `reduce`)
- 3 stages - stage *i* contains all nodes with exactly *i* tasks executed
- "Direct" vs "indirect" MR
  - Intermediate data stored locally



Sequences

- DTG for each job
- Each node in stage 2 of DTG of MR job *i* merged with corresponding node in stage 0 of MR job *i*+1 DTG

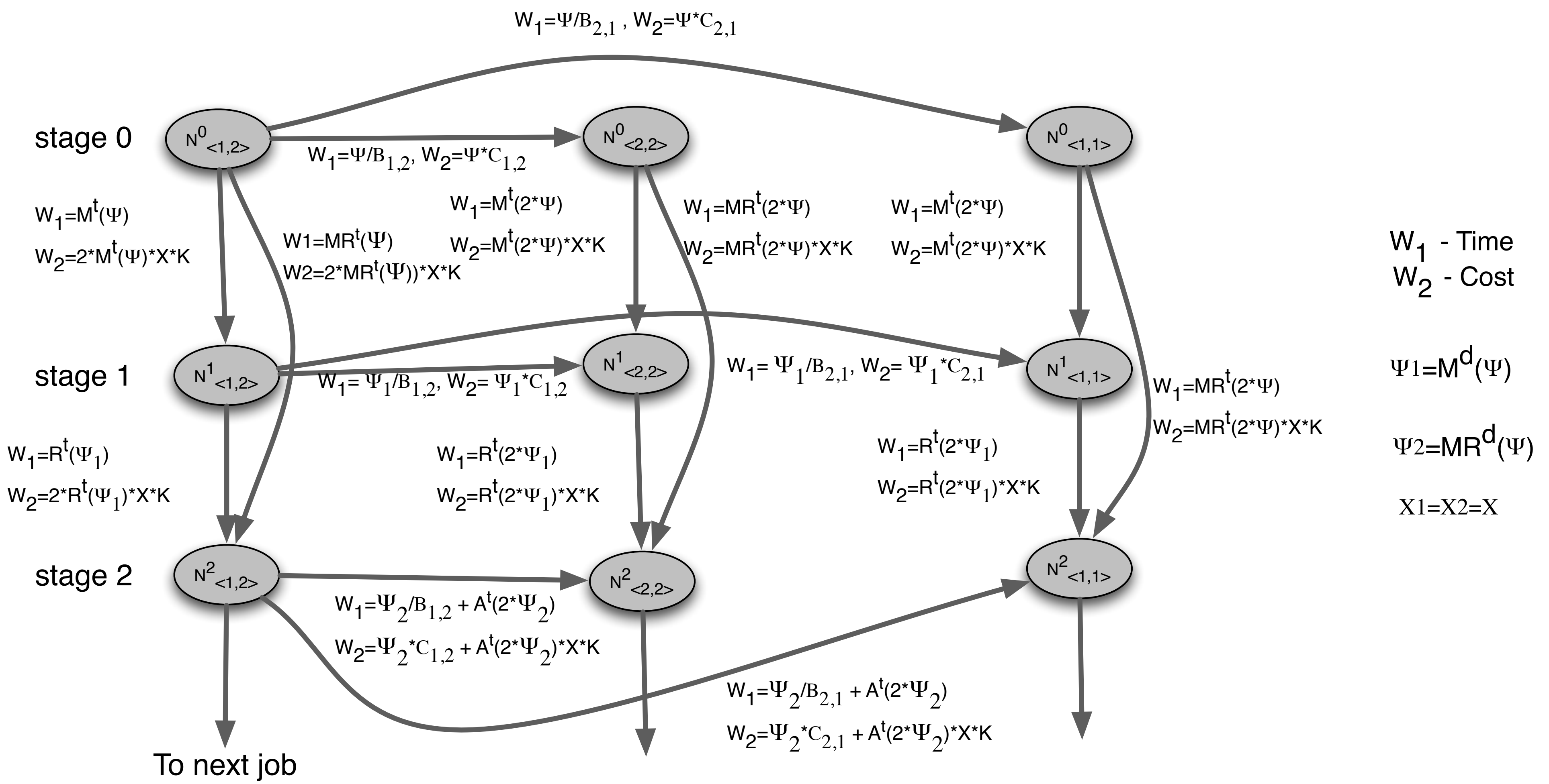


Sampling and Extrapolation

- Determining edge weights
  - Execute each task on data samples in all execution clusters (in parallel), develop functions to determine execution time and output size
    - (**Not** sampling all paths)
  - Extrapolation used to predict execution time and output size for large amounts of data
  - Users can manually specify functions

Determining Edge Weights

Example DTG and functions



G-MR

- Java framework implementing DTGs and corresponding algorithms
    - Extends Apache Hadoop
    - Java annotations for associativity, functions
- Tested in Amazon EC2 with up to 1 TB of data distributed across 4 datacenters

Evaluation Setup

- Up to 4 EC2 datacenters located in US East Coast, US West Coast, Europe and, Asia
- 10 large EC2 nodes (7.5 GB of memory, 4 EC2 compute units) in each datacenter
- Nodes leased at $0.34 per hour, data transfer $0.1 per GB

Datasets

Dataset	GBs	Description
CENSUSData	121	Year 2000 US Census
EDUData	5	University Website crawl
WEATHERData	20	Weather measurements
PLANTData	10	Properties of Iris plant
HADOOPData	100	Logs of Yahoo! Hadoop cluster
NGRAMData	300	Google Books Ngrams

Task sequences

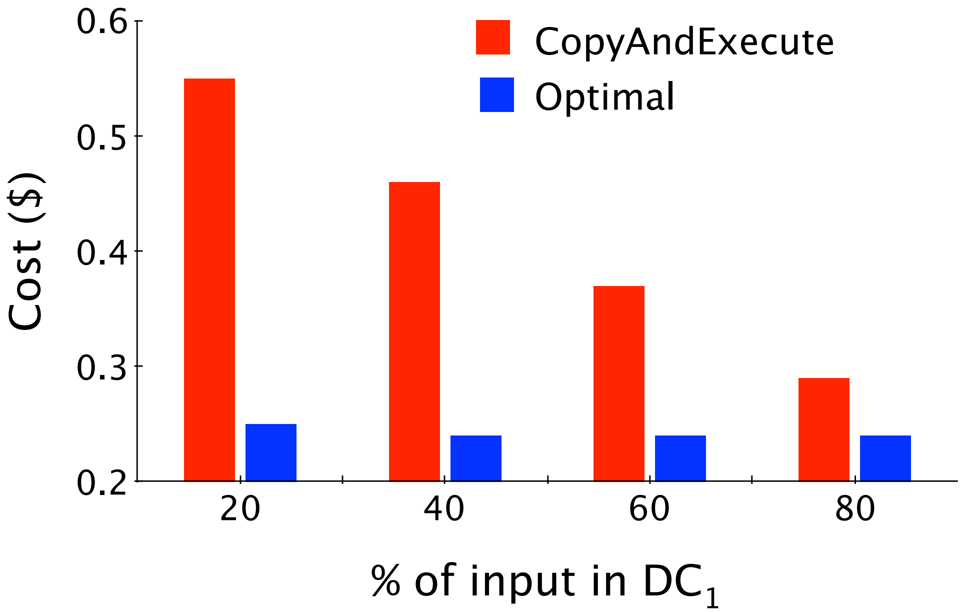
Job	Description
CENSUSPROCESSOR	Filters and groups CENSUSData.
WORDCOUNT	Counts the number of occurences of words in EDUData
MEDIANWEATHER	Computes the median of a record in WEATHERData
KNN	Type of each plant record in PLANTData
ETL	Extracts and performs a cross product on HADOOPData
NGRAM	All combinations of last two words of 4 grams

Evaluation

- Two datacenters (*DC_1* and *DC_2*)
- Different execution paths
  - CopyAndExecute - copy all data to a single datacenter prior to execution
  - ExecuteAndCopy - execute all tasks prior to copying
  - PartialCopy - balance the partitions in the middle

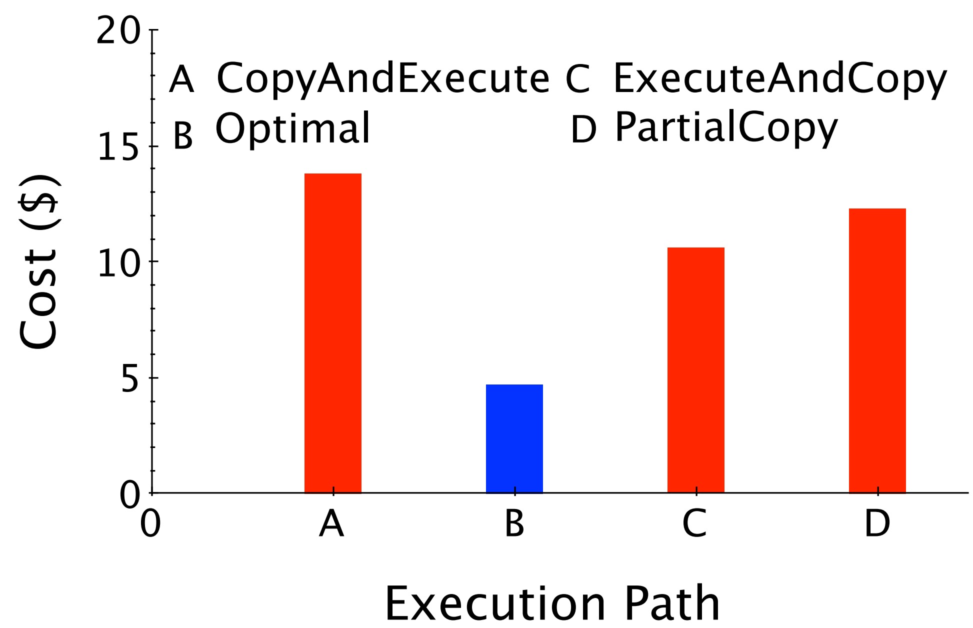
Monetary Cost

WORDCOUNT



Optimal - ExecuteAndCopy

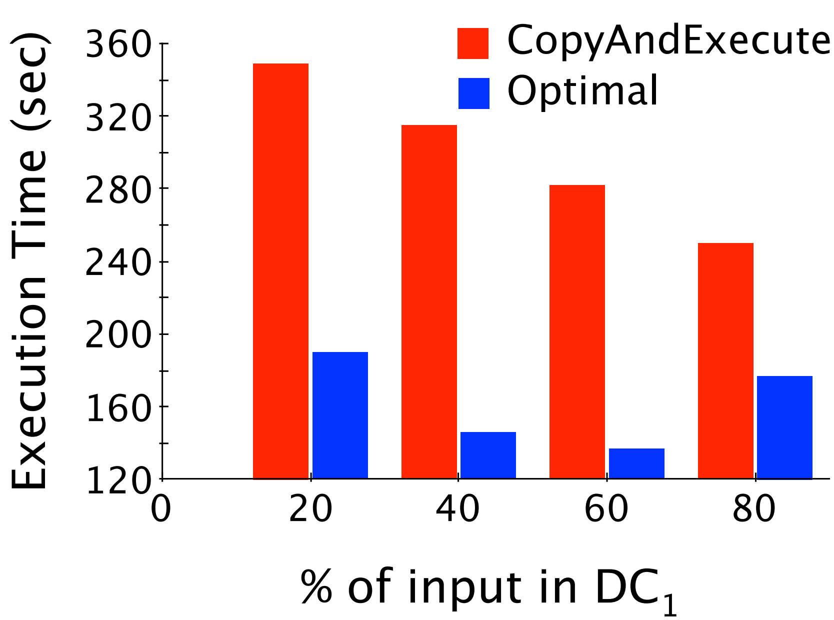
NGRAM



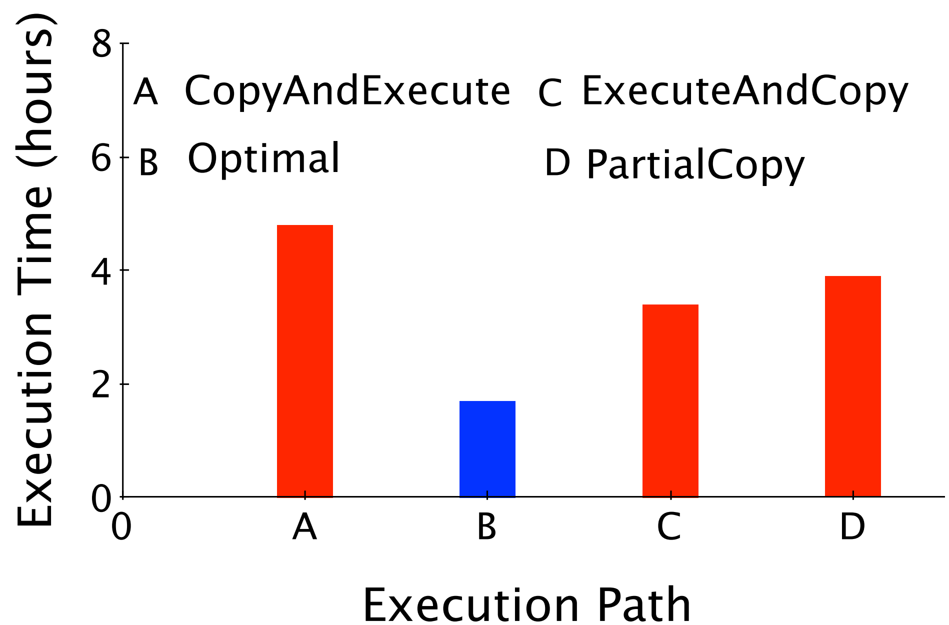
Optimal - copy data after first MR job

Execution Time

WORDCOUNT



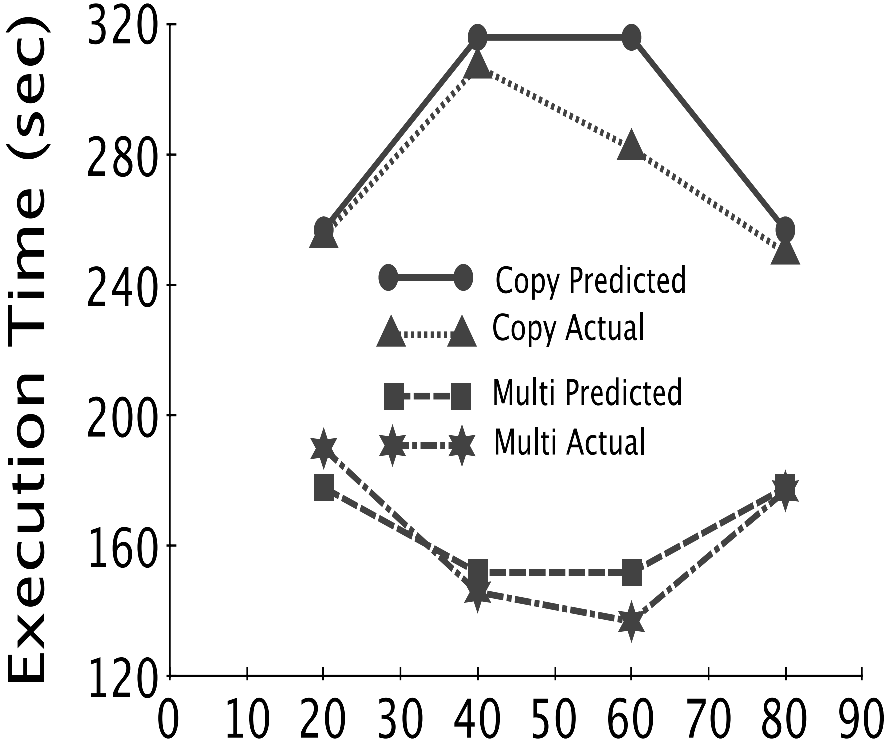
NGRAM

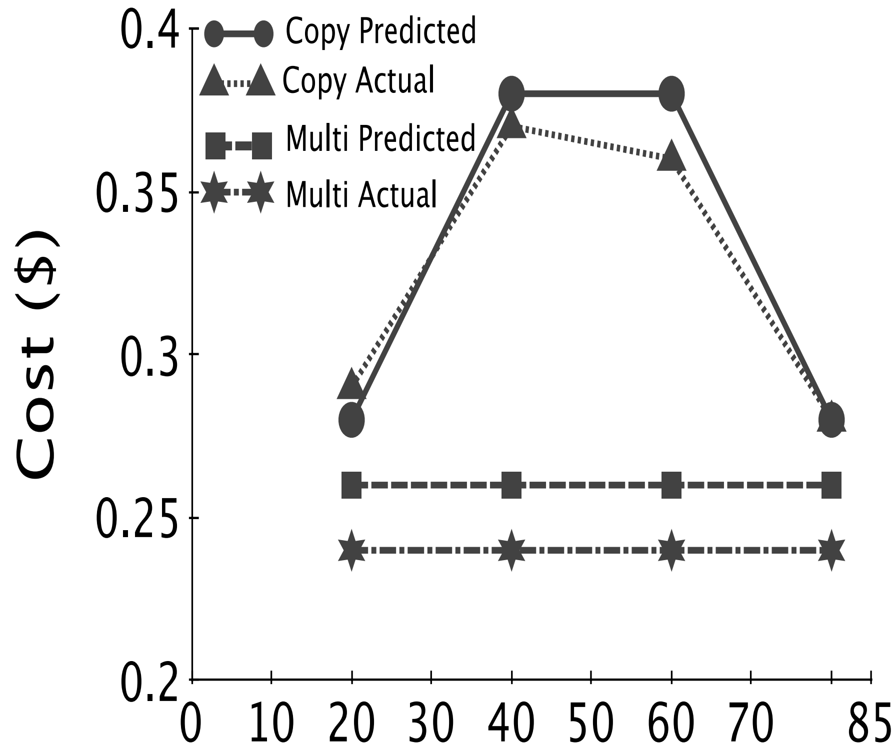


Optimal - ExecuteAndCopy

Optimal - copy data after first MR job

Prediction Accuracy





Outline

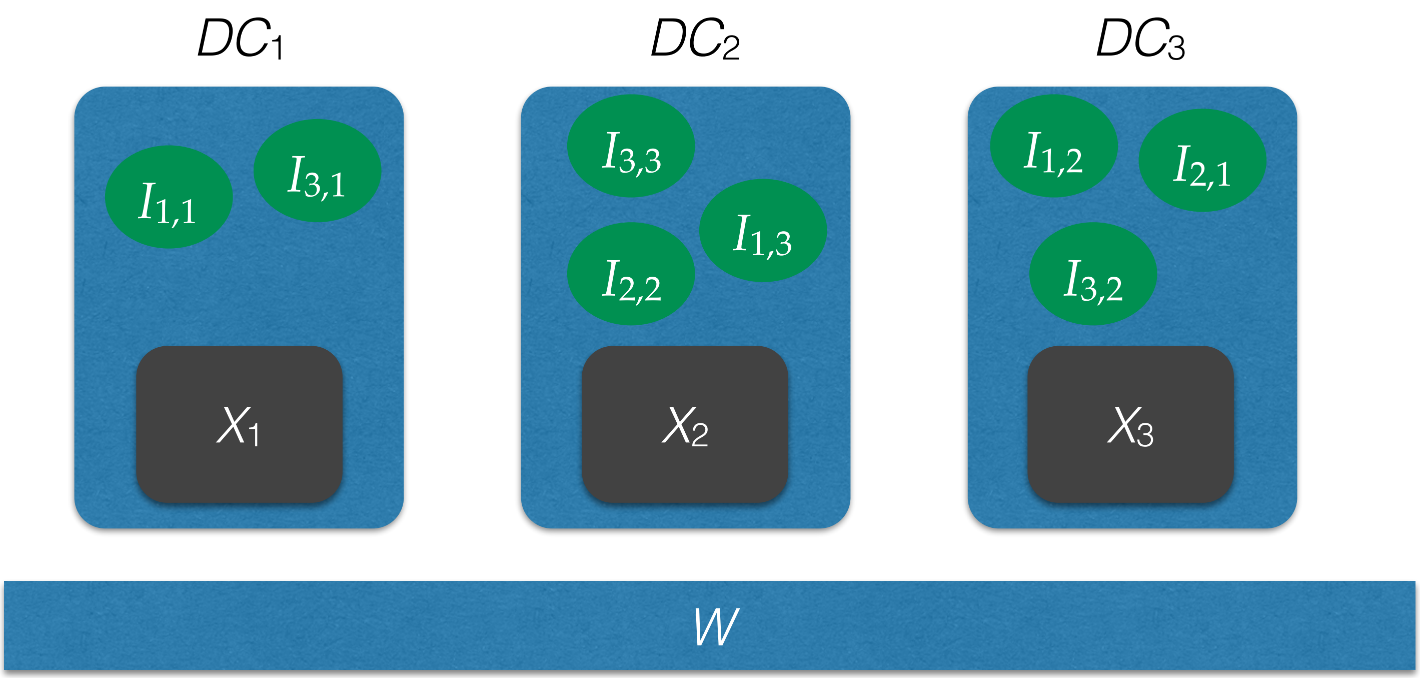
- Big data background
- Geo-distribution motivation
- Geo-distributed tasks
- **Geo-distributed workflows**
- Conclusions and outlook

GD Workflows

[Jayalath&Eugster;ICDCS'13]

- n datacenters DC_1 to DC_n and d input datasets DS_1 to DS_d - dataset DS_i consists of s_i sub-datasets ($1 \leq s_i \leq n$)
- GD workflow W, each task taking in one or more (possibly GD) datasets as input and generating one or more datasets as output

Example deployment



X_i - Execution cluster

Problem Statement

- How to efficiently perform workflows with GD datasets?
- Two aspects
  1. Executing efficiently: runtime extensions
  2. Expressing computation and constraints: language
     - Model: geo-distributed datastructures and operations
       - Why not transparent?
     - Pig Latin/Pig and Flume Java/Crunch

“Levels of Associativity”

A function $\boldsymbol{f}$ can be

1. Associative (mathematical sense), $f(X_1 \cdot X_2) = f(f(X_1) \cdot f(X_2))$, e.g., **`max, min, top-k`**
2. There exists a (known) function $\boldsymbol{g}$ s.t. $f(X_1 \cdot X_2) = g(f(X_1) \cdot f(X_2))$, e.g., **`avg.`**
   A. $\boldsymbol{g}$ is well-known (mostly for built-in simple functions, e.g., **`avg, word count`**)
   B. Can be synthesized (cf. 3rd Homomorphism Thm. [Morihata et al.;POPL’09])
   C. Can not be synthesized
3. Doesn’t exist / is unknown, e.g., **`join, top-k word count`**

Why not let programmer explicitly code?

Manual Distribution Example

```
input_lines = LOAD 'input_file'
   AS (line:chararray);
words = FOREACH input_lines GENERATE
   FLATTEN(TOKENIZE(line)) AS word;
word_groups = GROUP words BY word;
word_count = FOREACH word_groups
   GENERATE group, COUNT(words);
STORE word_count INTO 'output_file';
```

- More lines of code
- One hard-wired path - may not be optimal
- Has to be associative (e.g., **AVG**) or "aggregatable" if not strictly associative (e.g., **COUNT+SUM**)

```
// Part 1 : Executed in both datacenters
input_lines = LOAD 'input_file'
   AS (line:chararray);
words = FOREACH input_lines GENERATE
   FLATTEN(TOKENIZE(line)) AS word;
word_groups = GROUP words BY word;
word_count = FOREACH word_groups
   GENERATE group, COUNT(words);
STORE word_count INTO 'file1';

// Part 2 : Executed in datacenter DC2 only
// Copied data is stored in file2.
// -> Copy file1 of DC2 to file2 in DC1.

// Part 3: Executed in datacenter DC1 only
records1 = LOAD 'file1' AS
   (word:chararray, count:int);
records2 = LOAD 'file2' AS
   (word:chararray, count:int);
all_records = UNION records1, records2;
grouped = GROUP all_records BY word;
word_count = FOREACH grouped GENERATE
   group, SUM(all_records.count);
STORE word_count INTO 'output_file';
```

Pig Latin Background: Types

- Simple: `int`, `long`, `float`, `double`, `chararray`, `bytearray`, `boolean`
- Complex
  - `tuple` - an ordered set of fields
  - `bag` - a collection of `tuple`s
  - `map` - a set of key-value pairs
  - relation - an outer bag with no complex types

Operations and Functions

- Operations
  - **`UNION`** ceates a union of two or more relations
  - **`CROSS`** creates a cross product of two or more relations
  - **`JOIN`** joins two or more relations
  - **`GROUP`** groups elements of a relation based on a given criteria
- Functions
  - *Eval* functions, e.g., **`AVG`**, **`COUNT`**, **`CONCAT`**, **`SUM`**, **`TOKENIZE`**
  - *Math* functions, e.g., **`ABS`**, **`COS`**
  - *String* functions, e.g., **`SUBSTRING`**, **`TRIM`**
  - *User defined functions* (UDFs)

Rout

- Define two new complex data structures
  - **`gdbag`** - collection of tuples but may represent tuples from multiple datacenters
  - **`gdmap`** - collection of key-value pairs which may be from multiple datacenters
  - **`bag`** and **`map`** are retained but *pinned* to single datacenters
- Operations and functions
  - String and math functions are always applied tuple-wise
  - Applied individually to sub-datasets in respective datacenters

Eval Functions

- Eval functions are usually applied to groups of tuples
- Users can provide optional *merge* function "**`g`**"(original eval function "**`f`**" is called *work* function)
  - Merge function: eval function is applied to individual sub-datasets followed by aggregation via merge
  - Otherwise: all data represented by the corresponding datastructure copied to a single datacenter
    - Destination is decided by Rout runtime (Rrun)

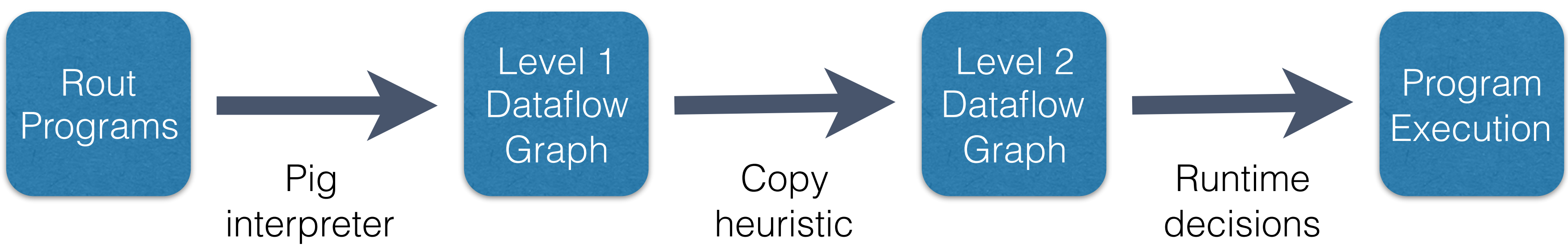
Operators and Example

- Other operations
  - Load and store operations - **GDLOAD**, **GDSTORE**
  - Operations for converting between **bag**s/**map**s and **gdbag**s/**gdmap**s - **COLLAPSE**, **GDCONVERT**
  - GD relational operations - **GDJOIN**, **GDGROUP**

```
// Input represents data in both datacenters
gd_input_lines = GDLOAD 'input' AS (line:chararray);
gd_words = FOREACH input_lines GENERATE
  FLATTEN(TOKENIZE(line)) AS word;
gd_word_groups = GDGROUP gd_words BY word;
gd_word_count = FOREACH gd_word_groups GENERATE group,
  COUNT(gd_words);
word_count = COLLAPSE gd_word_count;
STORE word_count INTO 'output_file';
```

Rout Runtime Infrastructure (Rrun)

Execution steps of Rrun

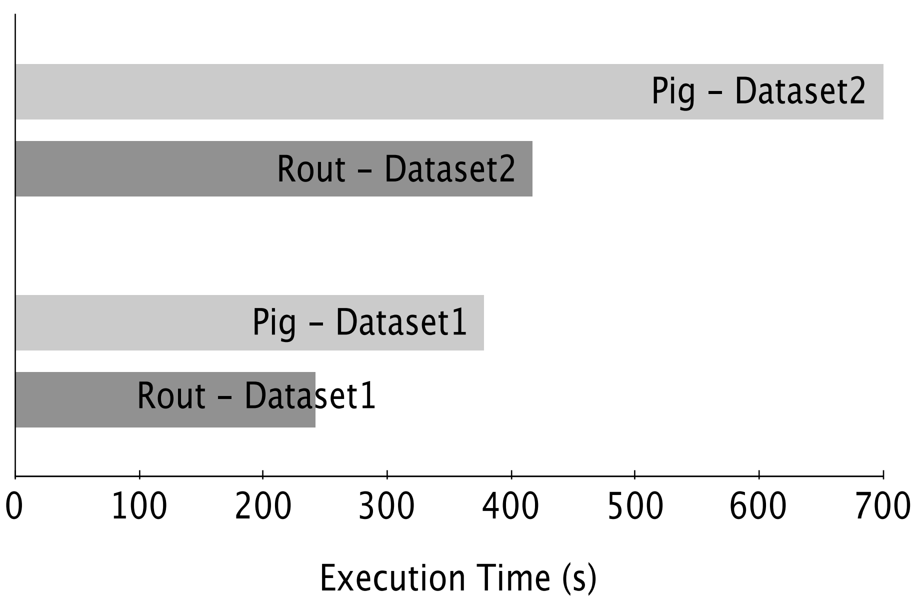


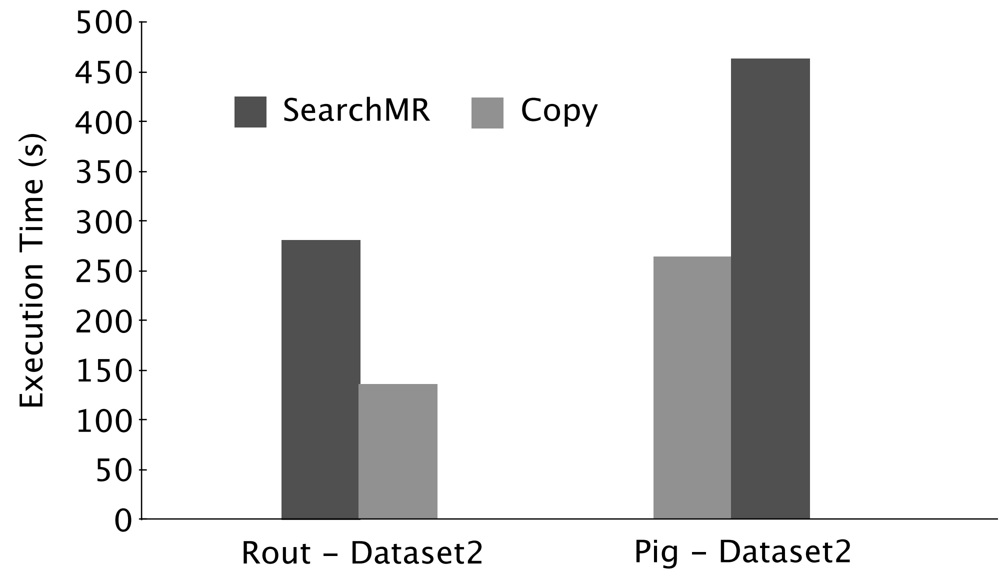
- Lazy heuristic copying data across datacenters when needed
  - E.g., operation is non-associative and no merge function is provided
  - Only decides at which points data should be copied, not where to

Evaluation

- 2 EC2 datacenters
- 10 nodes from each datacenter with 1.7 GB of memory and 1 EC2 virtual core running Ubuntu Linux
- Uses HADOOPData dataset, search for exceptions

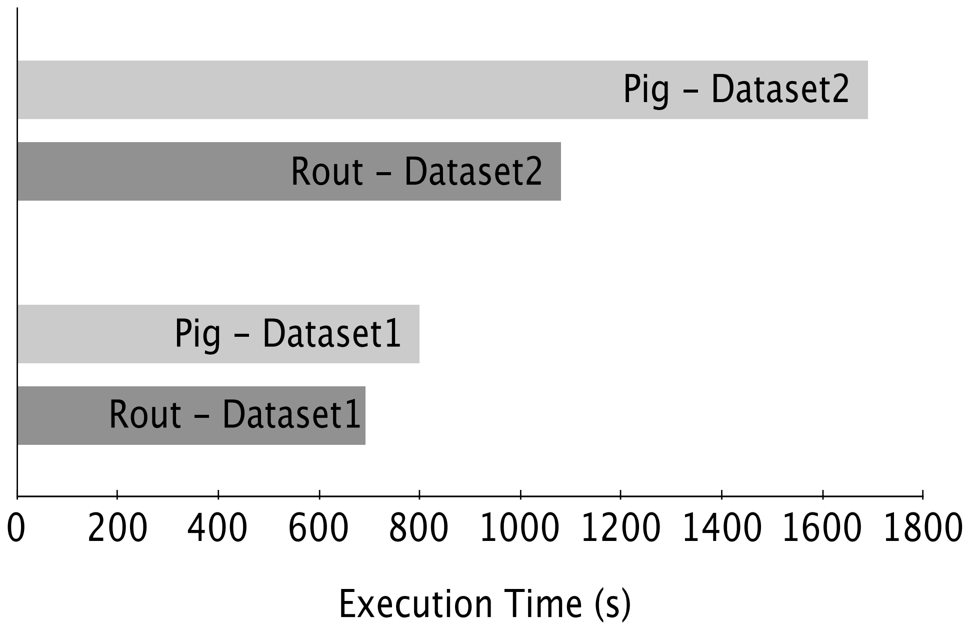
Log Search

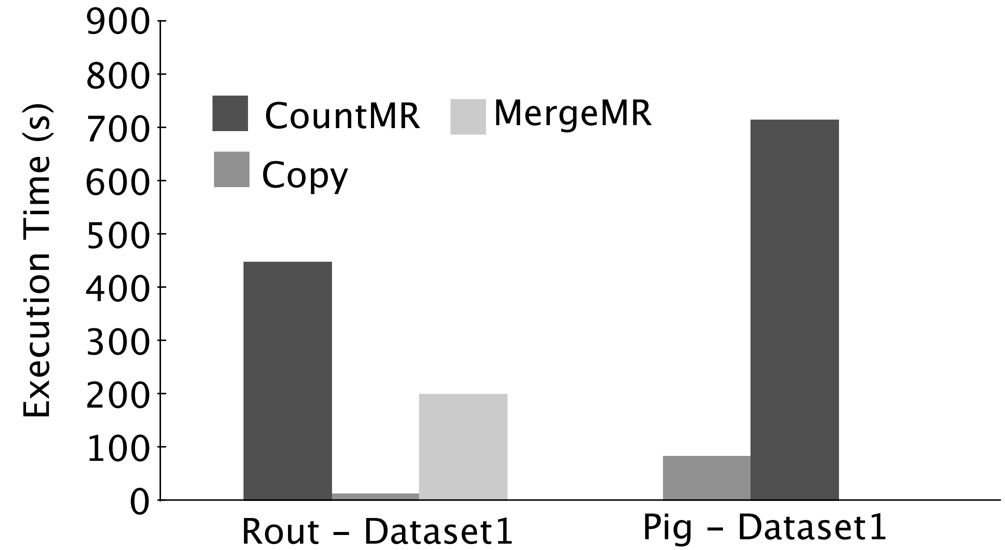




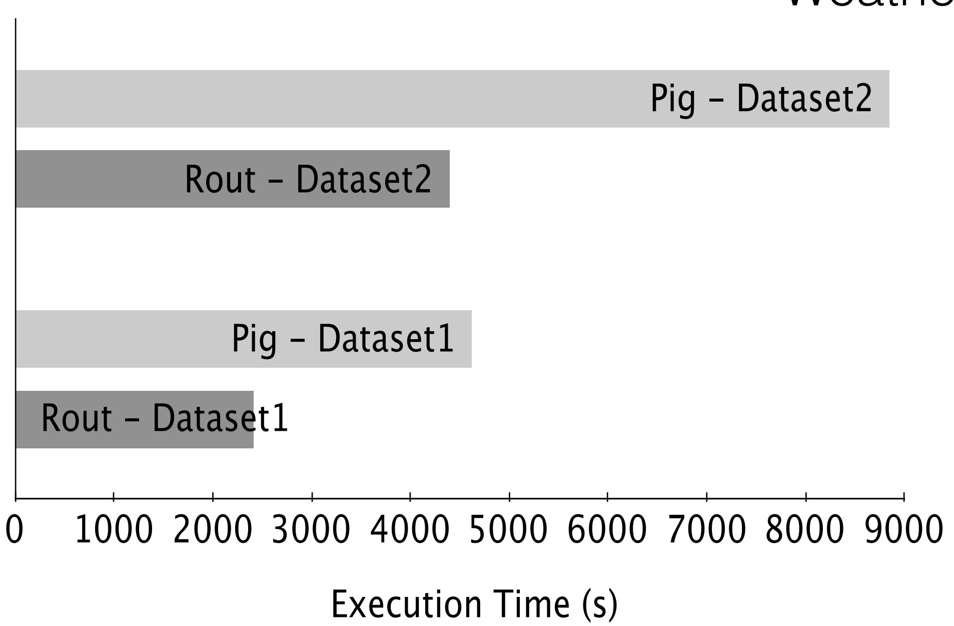
Rout - copy data after individual searches

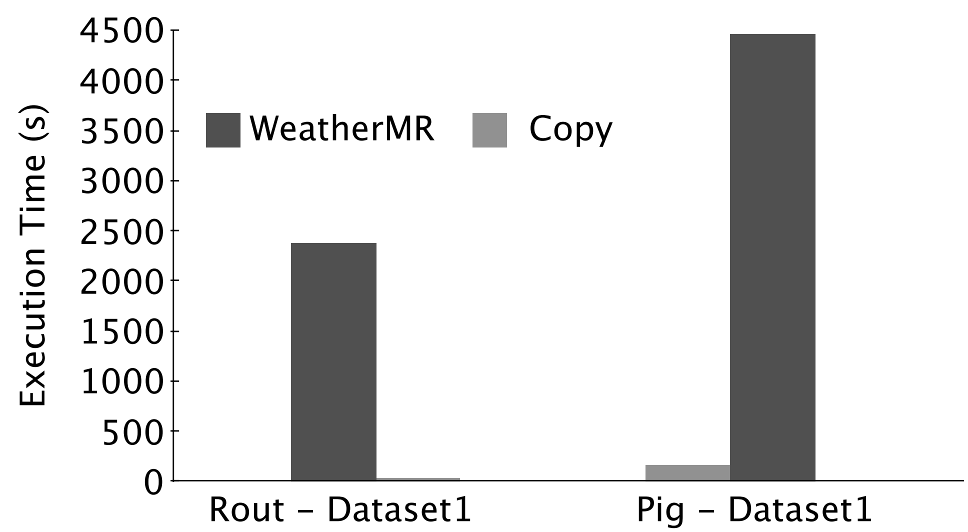
Log Debugger





Weather Explorer





Programmer Effort

Experiment	FlumeJava						Pig					
	Naïve		Explicit		DuctWork		Naïve		Explicit		Rout	
	LoC	K	LoC	K	LoC	K	LoC	K	LoC	K	LoC	K
Log debugger	5	7	11	14	6	8	6	15	13	26	7	18
Log search	3	4	8	9	4	5	3	7	8	13	4	8
Weather explorer	7	10	12	15	8	11	7	17	12	24	7	18
Weather top k count	5	8	11	14	6	10	6	17	11	23	7	20
Weather top k average	5	7	11	13	6	9	6	17	12	25	7	20

Experiment	DuctWork/FlumeJava				Rout/Pig			
	Compared To Naïve		Compared To Explicit		Compared To Naïve		Compared To Explicit	
	LoC	K	LoC	K	LoC	K	LoC	K
Log debugger	$+20\%$	$+28\%$	-45%	-43%	$+17\%$	$+20\%$	-46%	-31%
Log search	$+33\%$	$+25\%$	-50%	-44%	$+33\%$	$+14\%$	-50%	-38%
Weather explorer	$+14\%$	$+10\%$	-33%	-27%	$+0\%$	$+6\%$	-42%	-25%
Weather top k count	$+20\%$	$+25\%$	-45%	-28%	$+17\%$	$+18\%$	-36%	-15%
Weather top k average	$+20\%$	$+28\%$	-45%	-31%	$+17\%$	$+18\%$	-42%	-20%

Outline

- Big data background
- Geo-distribution motivation
- Geo-distributed tasks
- Geo-distributed workflows
- **Conclusions and outlook**

Conclusions

- Unlikely all data in the world will ever be in 1 datacenter
- Communication latency related to distance
- Geographical constraints matter
- Operating closer to data pays off in most cases
- Beyond the atmosphere - fog computing

Future Work

- Optimization
  - DTGs/G-MR
    - Heuristics to further reduce complexity
    - Higher-degree polynomials for approximation
  - Rout reconciliation
    - Fine granularity of DTG offline optimization
    - Simple Rout heuristic considering online resource usage

Yet More Future Work

- Model and expressiveness
  - Flume Java/Ductwork
  - Iterative and incremental jobs, main-memory datastructs, cf. Flink, Spark
    - Optimal aggregation [Culhane et al.;HotCloud'14], [Culhane et al.;INFOCOM'15]
  - UDFs
- Security
  - Integrity, availability, and isolation [Stephen&Eugster;Middleware'13]
  - Confidentiality [Stephen et al.;HotCloud'14], [Stephen et al.;ASE'14]

Next Week

- Resource management
- Security