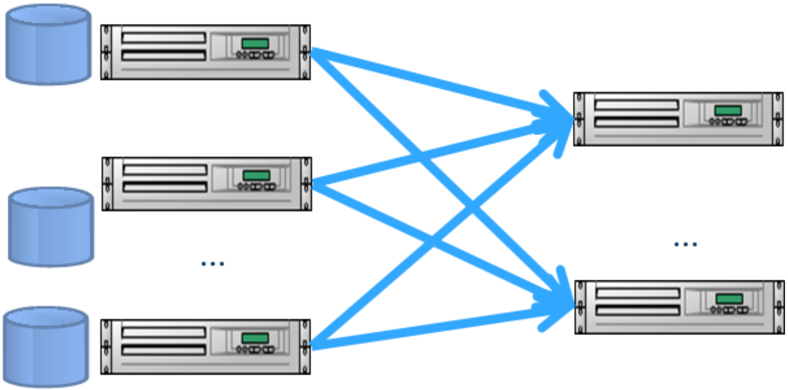
Rhea: Automatic Filtering for Unstructured Cloud Storage

Christos Gkantsidis, Dimitrios Vytiniotis, Orion Hodson, Dushyanth Narayanan, Florin Dinu, Antony Rowstron

Microsoft Research, Cambridge, UK

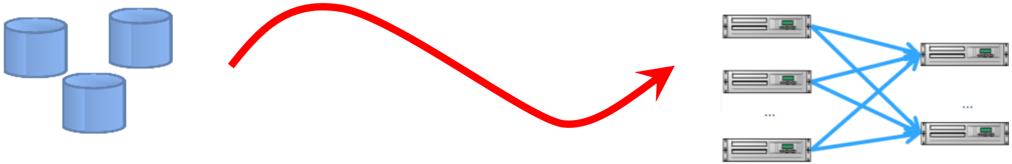
Cluster design for data analytics: [Traditional] Collocate storage & compute



- Hadoop & MapReduce, Dryad/DryadLinq, Scope, etc

Cloud Analytics: Hadoop in the Cloud

Separate storage and compute



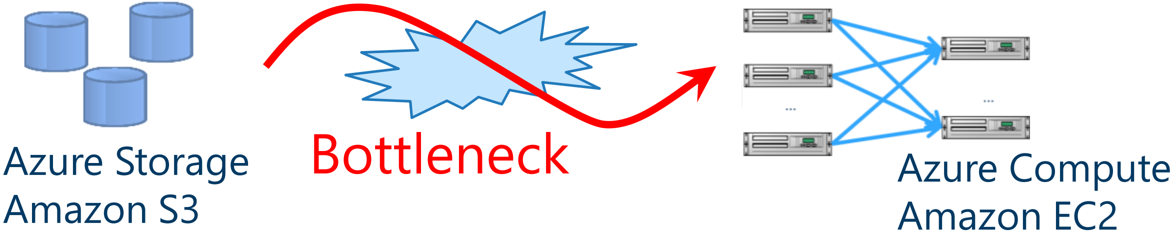
Azure Storage
Amazon S3

Azure Compute
Amazon EC2

Examples:

☐ Hadoop on Azure
☐ Amazon’s Elastic MapReduce

Cloud Analytics: Hadoop in the Cloud
Separate storage and compute



Azure Storage
Amazon S3

Bottleneck

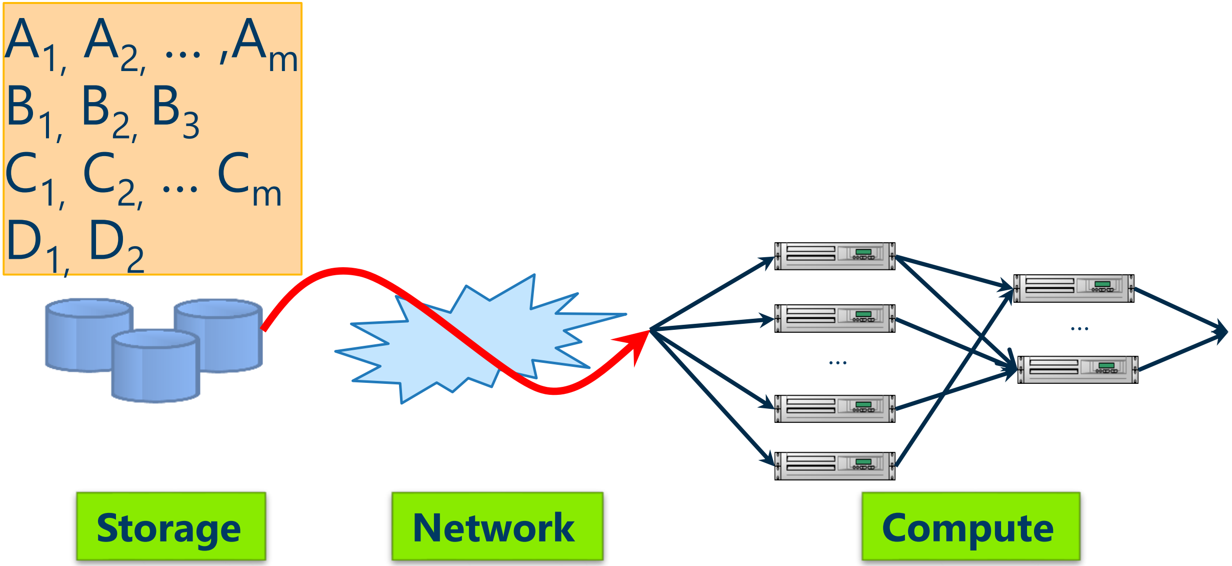
Azure Compute
Amazon EC2

Why separate storage from compute?

+ (User) Don’t pay for compute just to keep data alive
+ (User) Offload storage management to operator
+ (Operator) Evolve compute & storage independently
+ (Operator) Offer services that do not require both

- **Network between storage and compute is limited**
  (see paper for details)

Problem: Transfer lots of data …



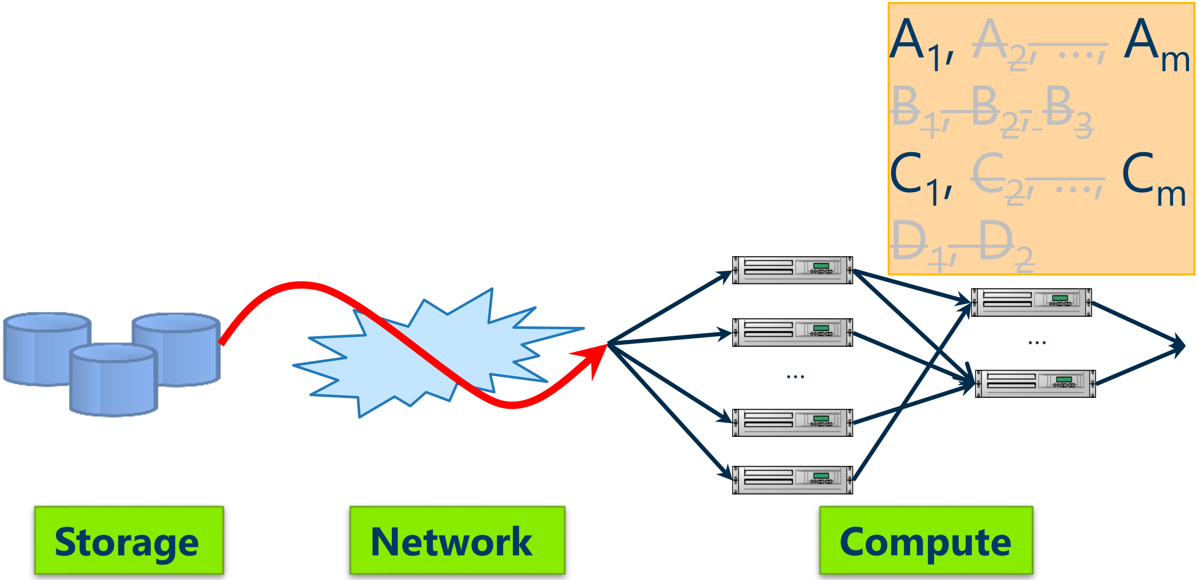
$A_{1,}\ A_{2,}\ \ldots\ ,A_m$
$B_{1,}\ B_{2,}\ B_3$
$C_{1,}\ C_{2,}\ \ldots\ C_m$
$D_{1,}\ D_2$

Storage

Network

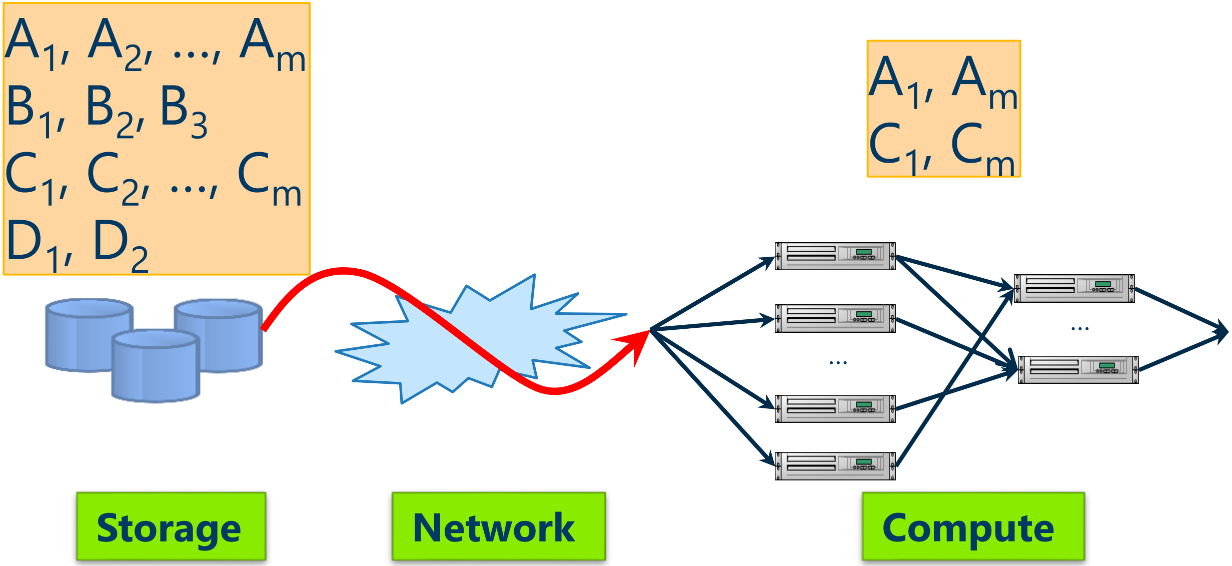
Compute

Problem: Transfer lots of data …
… even when only a subset is needed



A_1, ~~A_2, …,~~ A_m
~~B_1, B_2, B_3~~
C_1, ~~C_2, …,~~ C_m
~~D_1, D_2~~

Problem: Transfer lots of data …
… even when only a subset is needed



$A_1, A_2, \ldots, A_m$
B_1, B_2, B_3
$C_1, C_2, \ldots, C_m$
D_1, D_2

A_1, A_m
C_1, C_m

Storage

Network

Compute

Scenario

- Apache Hadoop (Map/Reduce)
- Input data in storage service
- Hadoop running in compute service
- Unstructured data:
  - **text, log files, etc**

Goal

Transparently reduce data transfers from storage to compute

How to minimize transfers?

- Strawman: Can we execute mappers on storage nodes?
  - Intuition: Mappers throw away a lot of data
  - 😣 Data reduction not guaranteed
  - 😣 Difficult to stop mappers during storage overload
  - 😣 Storage nodes have to execute complicated logic (Hadoop system & protocol)
  - 😣 Dependencies on runtime environment, libraries, etc

- Better approach: Filter unnecessary data at storage nodes
  - Filters need to be **opportunistic and transparent**
    i.e. can kill/restart at any time (e.g. during overload)
  - Filters need to be **correct**
    i.e. always preserve correctness of computation

Challenge: How to filter the data?

Recall: data are typically unstructured text

- No external source of structure/schema

Insight:

- The data analytic job knows structure
- … and what needs to be filtered

Idea: static analysis of job bytecode

Input Value

```
public void map(... value ...)
{
String[] entries = value.toString().split(“\t”);
String articleName = entries[0];
String pointType = entries[1];
String geoPoint = entries[2];

if (GEO_RSS_URI.equals(pointType)) {
     StringTokenizer st = new StringTokenizer(geoPoi
     String strLat = st.nextToken();
     String strLong = st.nextToken();
     double lat = Double.parseDouble(strLat);
     double lang = Double.parseDouble(
     String locationKey = .........
     String locationName = .........
     geoLocationKey.set(locationKey);
     geoLocationName.set(locationName);
     outputCollector.collect(geoLocationKey, geoLocationName);
} }
```

Projection operation

- 3 “columns” interesting (out of 4 for this job)

Selection operation

- roughly 1/3 of rows are of the interesting type

”selects”/”projects” implicit in Java byte code

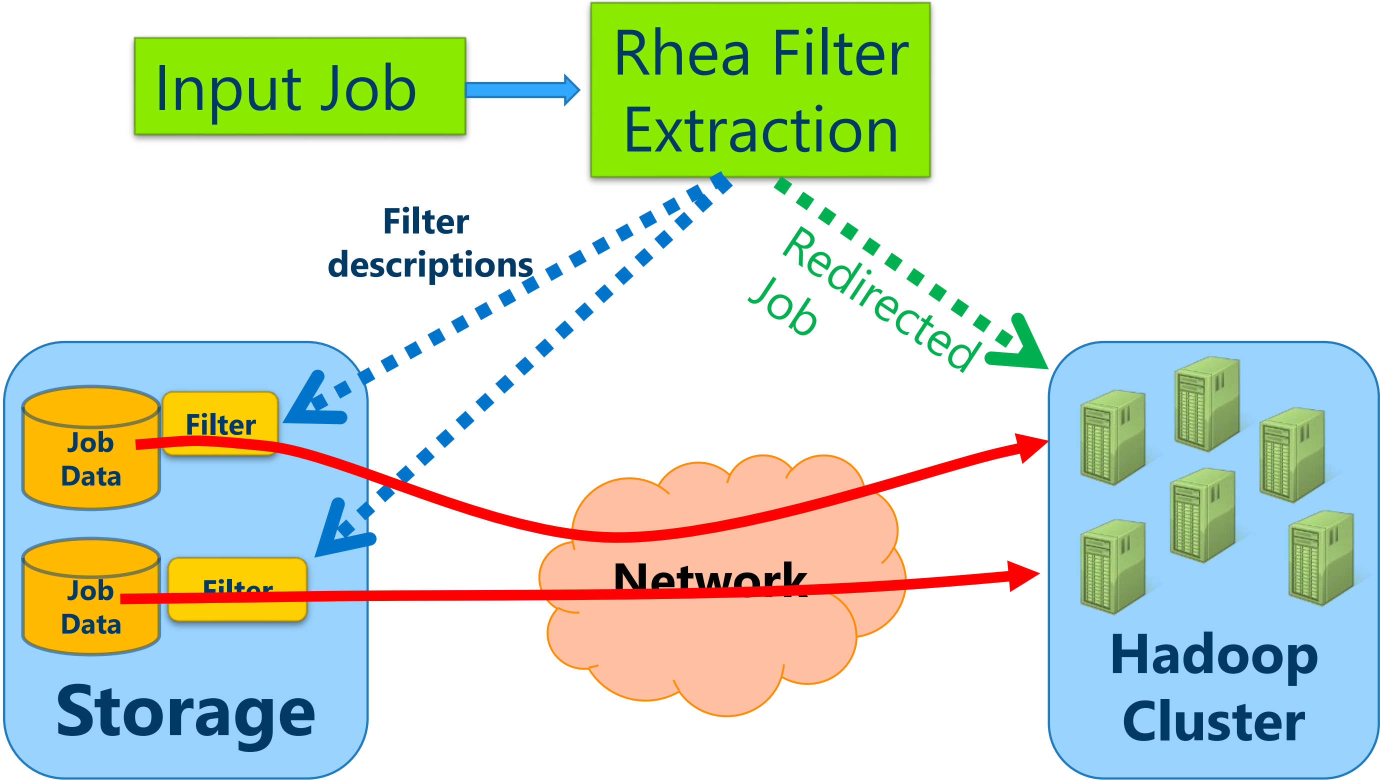
Output operation

Rhea

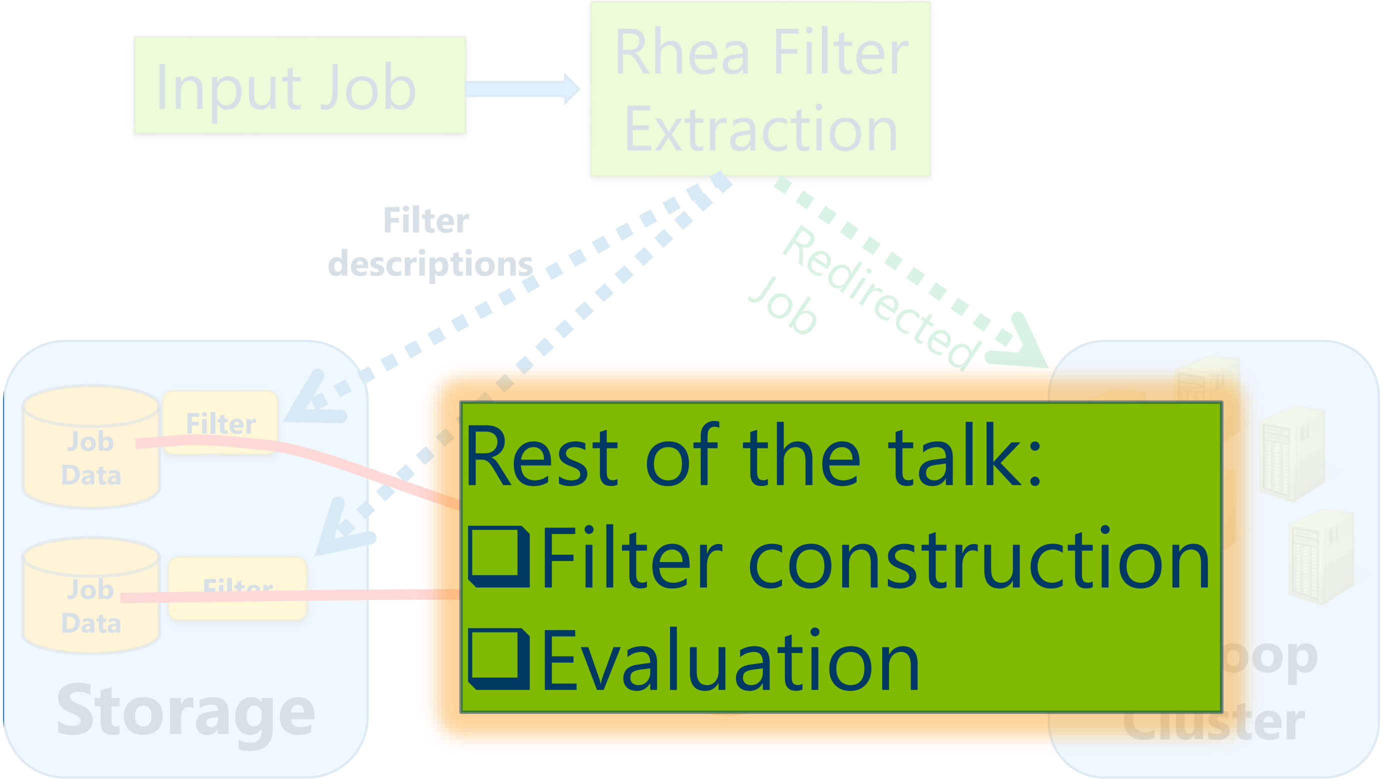
- Static analysis of Java byte code
- Extract row (select) & column (project) filters
  - as **executable** Java methods
  - column filters can also be C, regular expressions, etc.

- Filters are **conservative**:
  - May accept more data than strictly necessary
- Filters are **opportunistic**
  - kill/restart at any time (e.g. during storage overload)
- Filters are **transparent**
  - no change to Hadoop job

Rhea's Architecture



Rhea’s Architecture



Rest of the talk:

☐ Filter construction

☐ Evaluation

Filters: Identify bits of data that affect output of mapper

- Row Filters:
  - Given an input row:

    Does it lead to output?
  - Row corresponds to one invocation of map

  - Approach: Path Slicing
  - Challenge: Deal with mutable state

- Column Filters:
  - Given a row that leads to output:

    Which substrings of the row affect output?

  - Approach: Abstract interpretation
  - Challenge: Deal with loops

Row Filter Generation via Path Slicing

```
public void map(... value ...)
{
String[] entries = value.toString().split(“\t”);
String articleName = entries[0];
String pointType = entries[1];
String geoPoint = entries[2];

if (GEO_RSS_URI.equals(pointType)) {
        StringTokenizer st = new
        StringTokenizer(geoPoint, " ");
        String strLat = st.nextToken();
        String strLong = st.nextToken();
        double lat = Double.parseDouble(strLat);
        double lang = Double.parseDouble(strLong);
        String locationKey = .........
        String locationName = .........
        geoLocationKey.set(locationKey);
        geoLocationName.set(locationName);
        outputCollector.collect(geoLocationKey,
        geoLocationName);
} }
```

1. Tag “observable” instructions
2. Identify path conditions that lead to observable instructions
3. Perform dataflow analysis to identify all instructions that affect path conditions
4. Emit code

```
public boolean filter(Text bcvar2) {
        String[] bcvar5 = bcvar2.toString().split(“\t”);
        String bcvar7 = bcvar5[1];
        boolean irvar0_1 =
GEO_RSS_URI.equals(bcvar7);
        if (irvar0_1 == 1) { return true; }
        return false;
}
```

Challenge: Taming State

- Map-Reduce program are often NOT pure functions
  ➔ M/R programmers use state (i.e. objects in heap):
  - … to avoid frequent initializations
  - … to pass job parameters
  - … to optimize temporary storage (e.g. with dictionaries)

- Filters cannot rely on mutable state:
  - Recall: output of filtered data = output of original data

- Solution: Tag all access to mutable fields as “observable” (i.e. output) instructions.

Column Filter Generation (aka projects)

- Goal: Identify substrings that affect output

- Based on *abstract interpretation*
  - Captures common patterns for "reading" fields:
    e.g. string tokenizers, regular expressions, etc.
  - Guarantees termination by using numerical constraints
    - Important to deal with loops

- Output:
    - Tokenization method and sep
    - List of indices of interesting to

Rest of the talk:

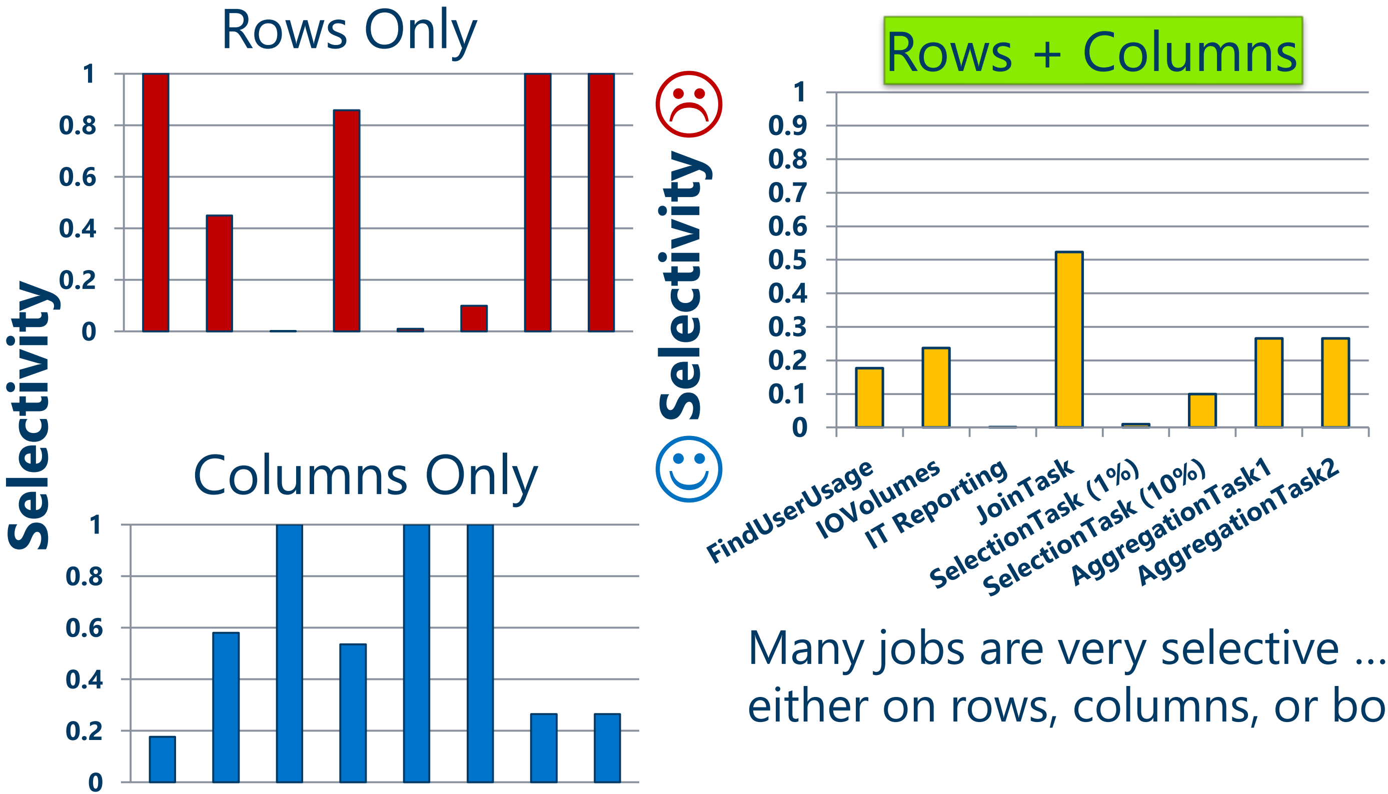
☐ Filter construction

☐ **Evaluation**

Experimental setup

- Hadoop on Azure:
- Input data in Azure Storage
- Compute on Azure Compute
- 8 jobs with both code and data
  - 200 jobs code only (in paper)
- Same data-center
  - Also, cross data-center (in paper)

Job Selectivity



Many jobs are very selective …
either on rows, columns, or both

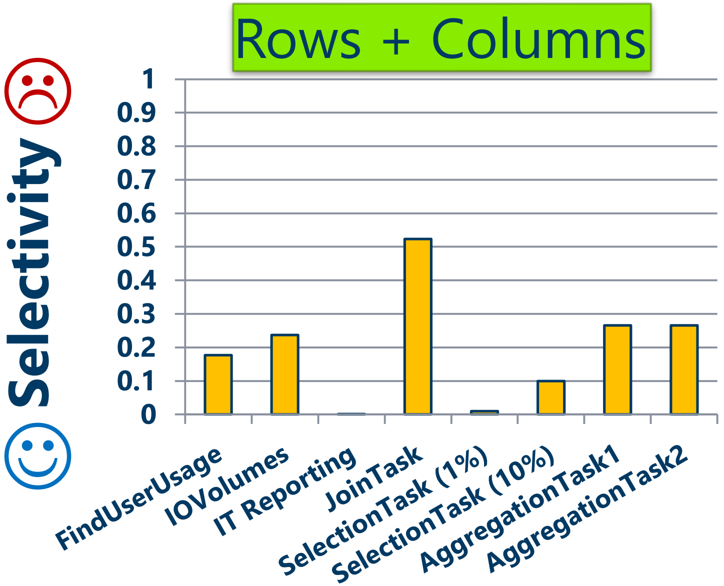
Job Selectivity

High selectivity ➔
less bytes to transfer

- ✓ Good for operators
- ✓ Cheaper for users for cross-data centers scenarios [see paper]

reduce runtime

- ✓ Good for users



Many jobs are very selective …
either on rows, columns, or both

Measuring runtime benefits

- We cannot extend Azure Storage or Amazon S3 with filters ☹
- Instead, **we use pre-filtered data** and compare with unfiltered data
- We assume storage with: (a) scalable I/O, and (b) enough processing power for filtering

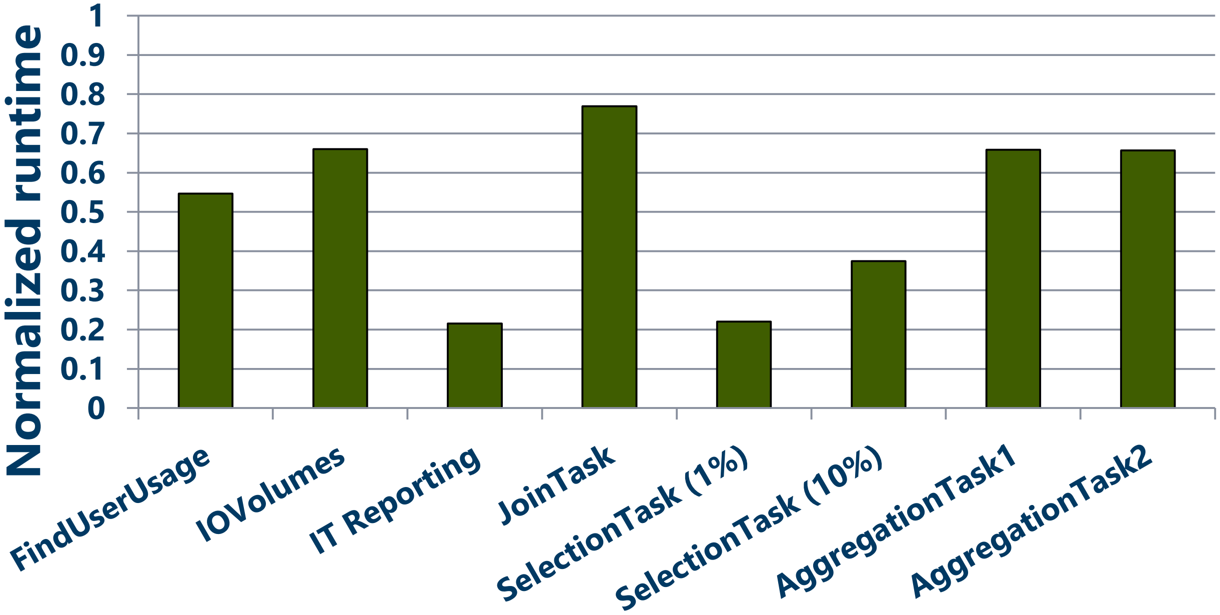
Diversion:
Do we have enough processing power?

- Row & Column filtering in **Java**: **~100MBytes/sec per core**
- Scales linearly with multiple cores

- ≤2 cores for filtering enough for all but 1 job
- Runtime **always reduces** runtime, even with fewer cores

- Performance dominated by string input/output, not filter
- Column filtering in optimized **C**: **5-17x faster** than Java

Runtime benefits



- ❑30-80% reduction in runtime
- ❑Runtime reductions less than selectivity due to Hadoop overheads

Conclusions

- Hadoop in the cloud: separation of storage and compute.

- Rhea minimizes transfers from storage to compute
  - Uses static analysis on the job bytecode
  - Extracts **selection** and **projection** operators from code
  - Generates filters to run in the storage layer
  - Runs transparently to user (and is safe for provider)
  - Potential benefits to the user (time, money) and cloud provider (bandwidth)

Microsoft