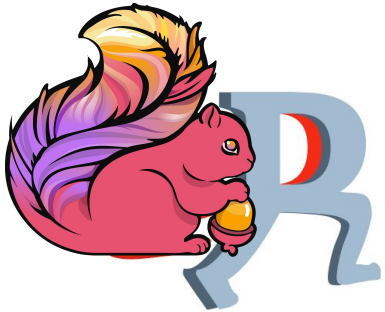


FastR + Apache Flink

Enabling distributed data processing in R

18 May 2016

Technische Universität Berlin

Juan Fumero
<juan.fumero@ed.ac.uk>

Oracle Labs

Disclaimer

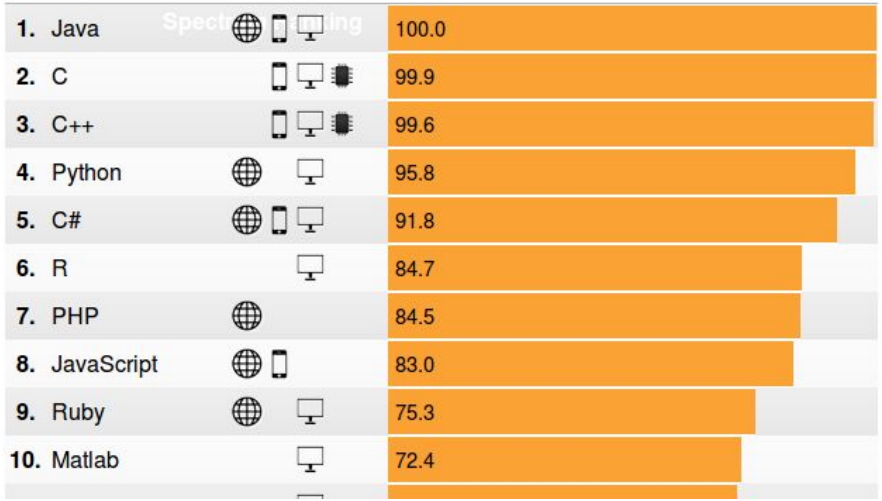
This is a work in progress

Outline

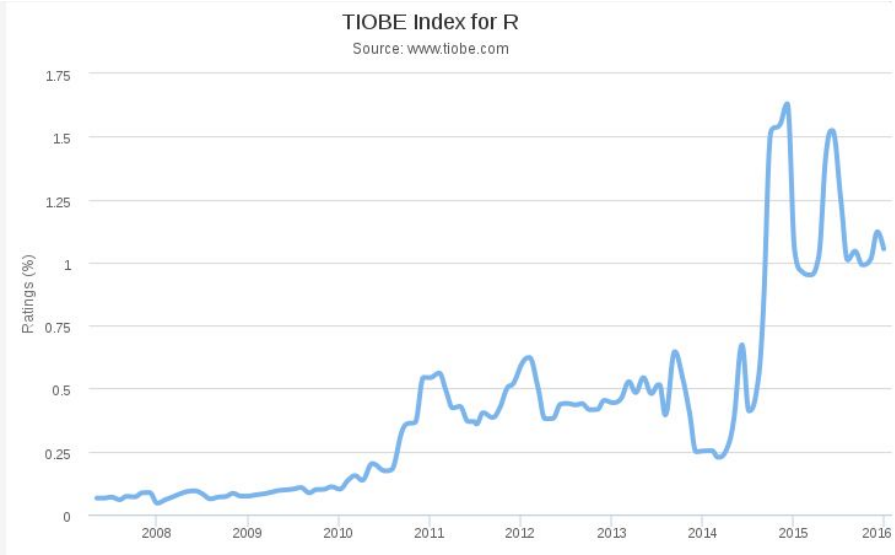
1. Introduction and motivation
   - Apache Flink and R
   - Oracle Truffle + Graal
2. **FastR + Flink**
   - **Supported Operations**
   - **Execution Model**
3. Preliminary Results
4. [DEMO] within a distributed configuration

Introduction and motivation

Why R?



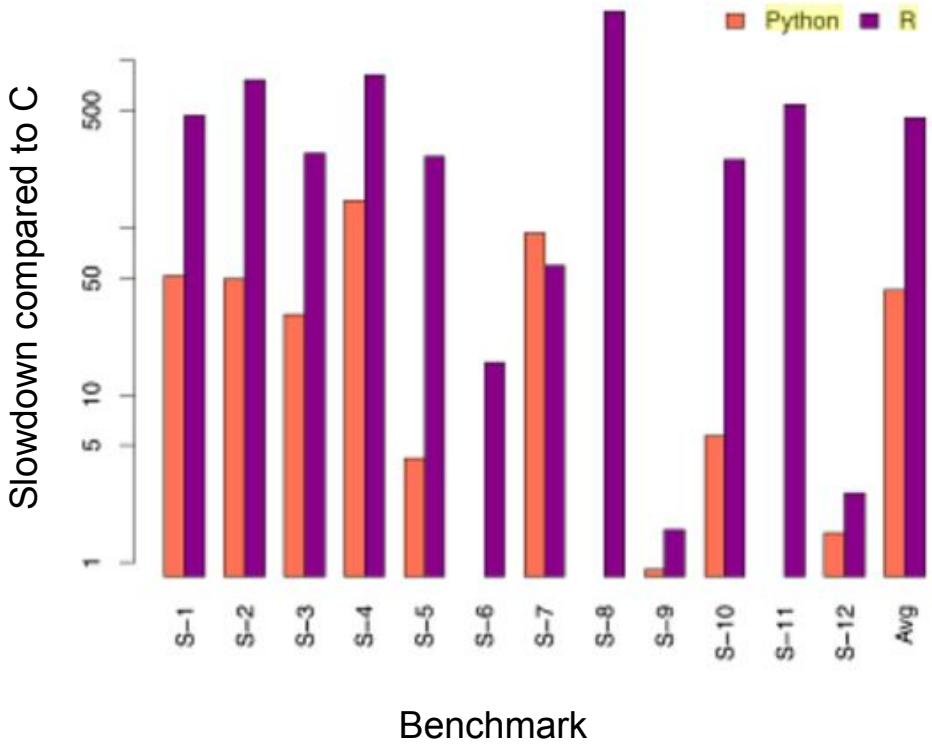
http://spectrum.ieee.org



R is one of the most popular languages for data analytics

R is:
- Functional
- Lazy
- Object Oriented
- Dynamic

But, R is slow


Python
R
Slowdown compared to C
1
5
10
50
500
S-1
S-2
S-3
S-4
S-5
S-6
S-7
S-8
S-9
S-10
S-11
S-12
Avg
Benchmark

Parallel R packages

- Snow
- OpenCL
- GPU-Tools
- ViennaCL
- CUDA
- ...

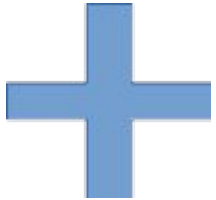
The R users need to adapt the code depending on the technology or parallel framework

Problem

GNU-R is neither fast nor distributed

Problem

GNU-R is neither fast nor distributed



Apache Flink

Framework for distributed and batch computing in Java and Scala

Apache Flink

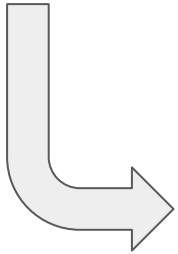
Framework for distributed stream and batch data processing

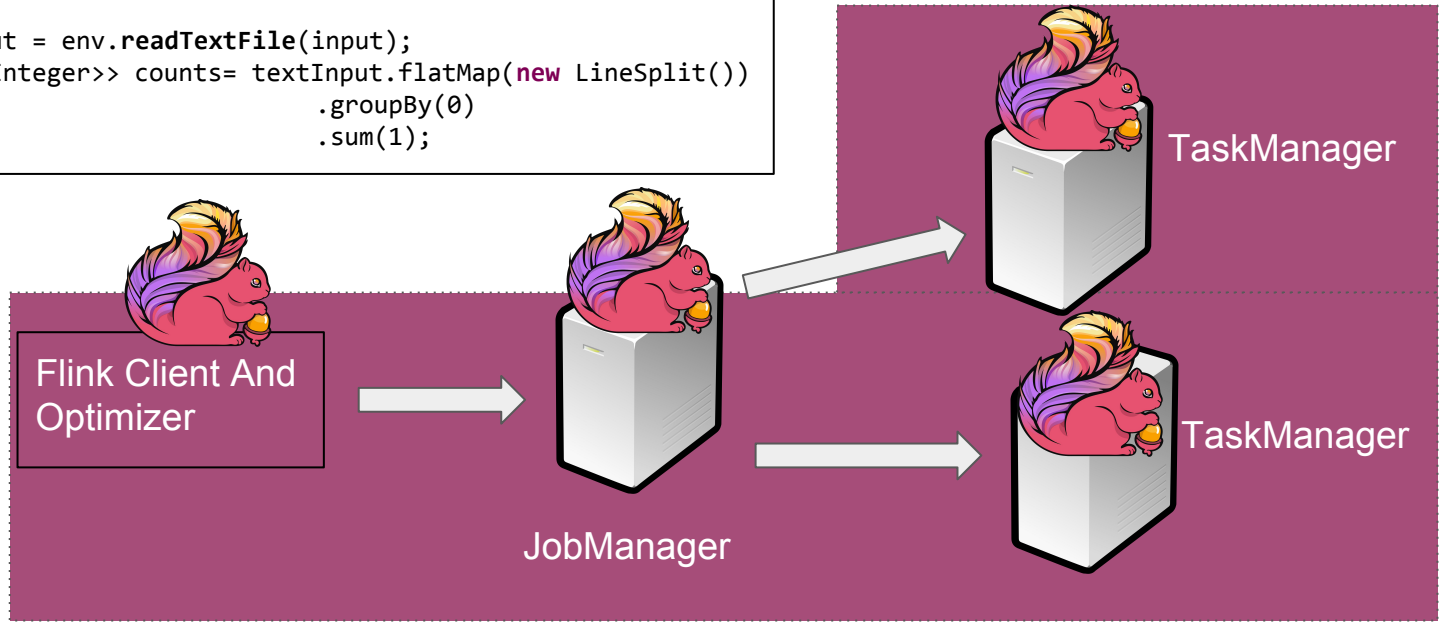
- Own task scheduler and data distribution
- Hadoop/Amazon plugins
- It runs on laptops, clusters and supercomputers with the same code
- High level APIs: Java, Scala and Python
- Free and Open Source

flink.apache.org

Apache Flink Example, Word Count

```
public class LineSplit … {
  public void flatMap(...) {
     for (String token : value.split("\\W+"))
        if (token.length() > 0) {
            out.collect(new Tuple2<String, Integer>(token, 1);
  }
}
DataSet<String> textInput = env.readTextFile(input);
DataSet<Tuple2<String, Integer>> counts= textInput.flatMap(new LineSplit())
                                                .groupBy(0)
                                                .sum(1);
```





Our Solution

We propose a compilation approach built within Truffle/Graal for distributed computing on top of Apache Flink.

FastR + Apache Flink

```
> sapply(input, function(x)
 {
    for ( … ) { … }
 } )
```

FastR: Truffle/Graal Overview

Compilers Overview

> x <- 1 + y

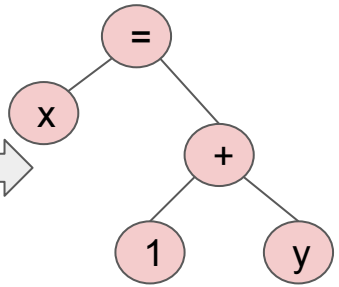
Lexer

ID: “x”
EQ: “<-”
NUMBER: “1”
PLUS: “+”
ID: “y”

Parser

Start ::= declaration
Declaration ::= ID EQ (ID | NUMBER) PLUS (ID | NUMBER)

AST



Byte-codes

1: const(1)
2: push y
3: iadd
4: store(x)

Assembly

```
mov ax, #1
mov bx, y
add ax,bx
mov di,offset result
mov [di], ax
```

Binary

How to Implement Your Own Language?

Current situation

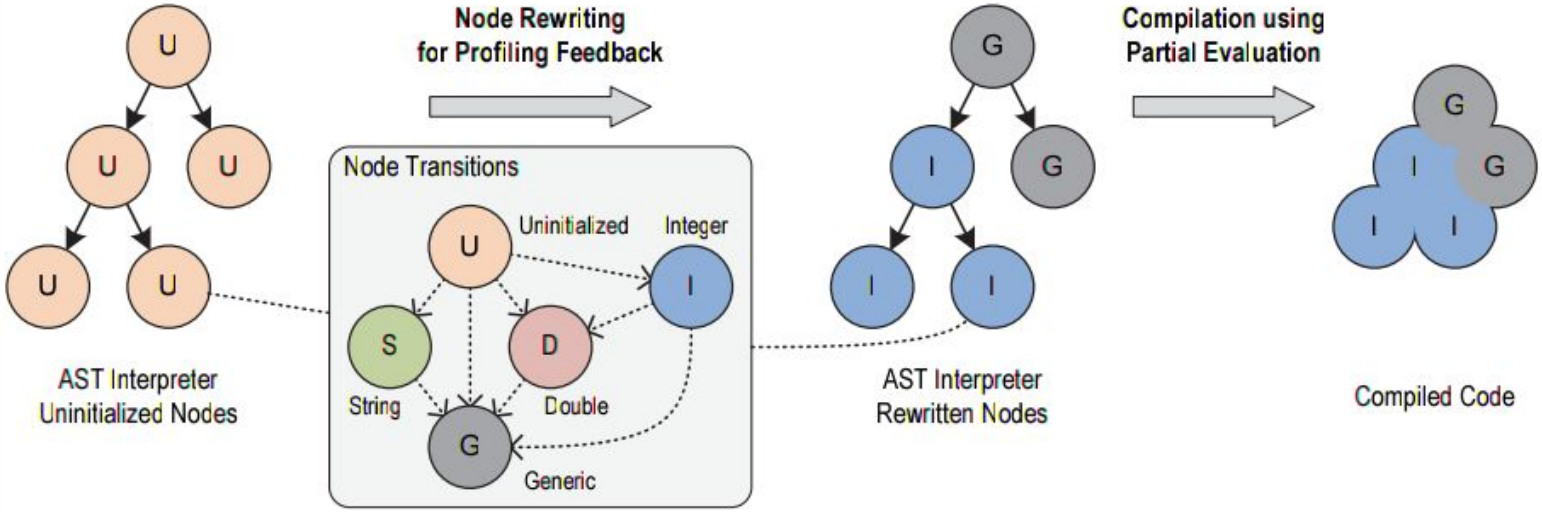
- Prototype a new language
  - Parser and language work to build syntax tree (AST), AST Interpreter
- Write a “real” VM
  - In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …
- People start using it
- People complain about performance
  - Define a bytecode format and write bytecode interpreter
- Performance is still bad
  - Write a JIT compiler
    Improve the garbage collector

How it should be

- Prototype a new language in Java
  - Parser and language work to build syntax tree (AST)
    Execute using AST interpreter
- People start using it
  - And it is already fast

Oracle Labs material

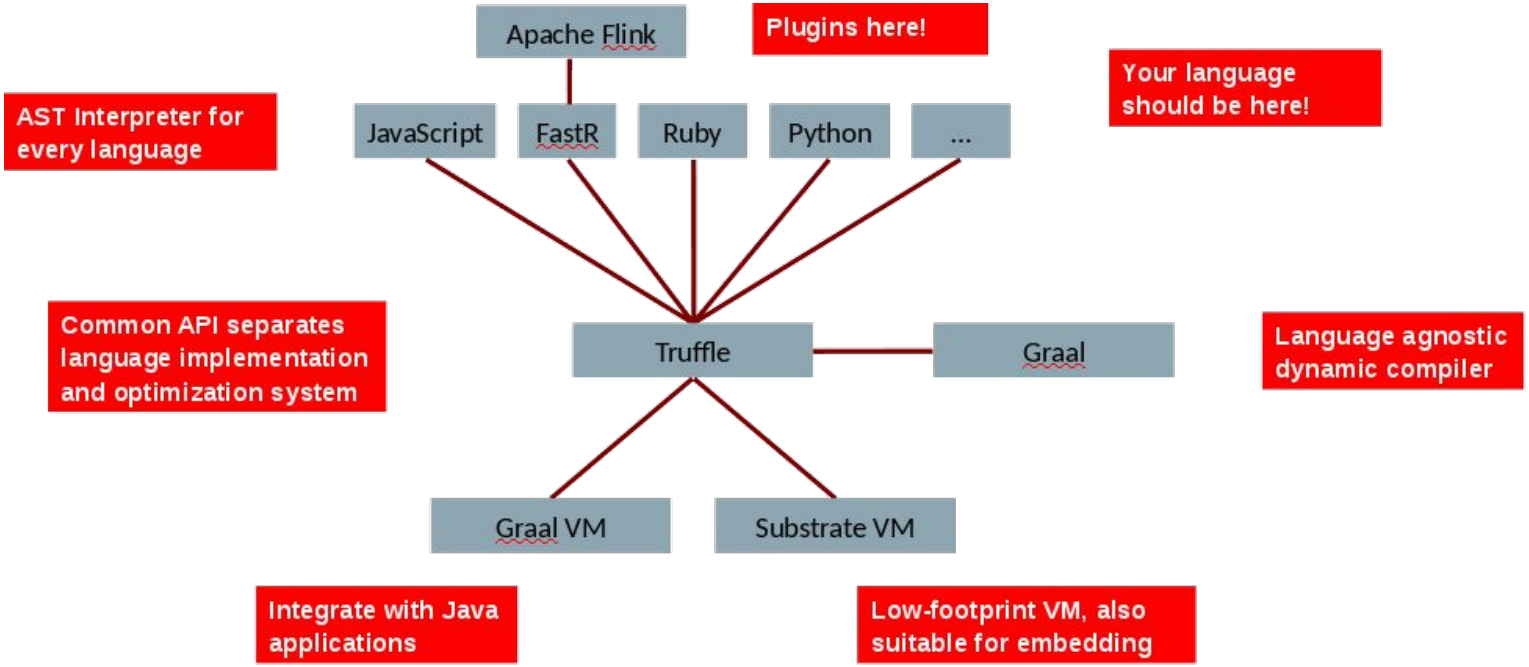
Truffle AST Specialization



Oracle Labs material

T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

Truffle/Graal Infrastructure



Oracle Labs material

FastR + Flink: implementation

Our Solution: FastR - Flink Compiler

- Our goal:
  - Run R data processing applications in a distributed system as easy as possible

- Approach
  - Custom FastR-Flink compiler builtins (library)
  - Minimal R/Flink setup and configuration
  - “Offload” hard computation if the cluster is available

FastR-Flink API Example - Word Count

```
# Word Count in R
bigText <- flink.readTextFile("hdfs://192.168.1.10/user/juan/quijote.txt")

createTuples <- function(text) {
        words <- strsplit(text, " ")[[1]]
        tuples = list()
        for (w in words) {
                tuples[[length(tuples)+1]] = list(w, 1)
        }
        return (tuples)
}

splitText <- flink.flatMap(bigText, createTuples)
groupBy   <- flink.groupBy(splitText, 0)
count     <- flink.sum(groupBy, 1)
hdfsPath <- "hdfs://192.168.1.10/user/juan/newQUIJOTE.txt"

flink.writeAsText(count, hdfsPath)
```

Supported Operations

Operation	Description
flink.sapply	Local and remove apply (blocking)
flink.execute	Execute previous operation (blocking)
flink.collect	Execute the operations (blocking)
flink.map	Apply local/remove map (non-blocking)
flink.sum	Sum arrays (non-blocking)
flink.groupBy	Group tuples (non-blocking)

Supported Operations

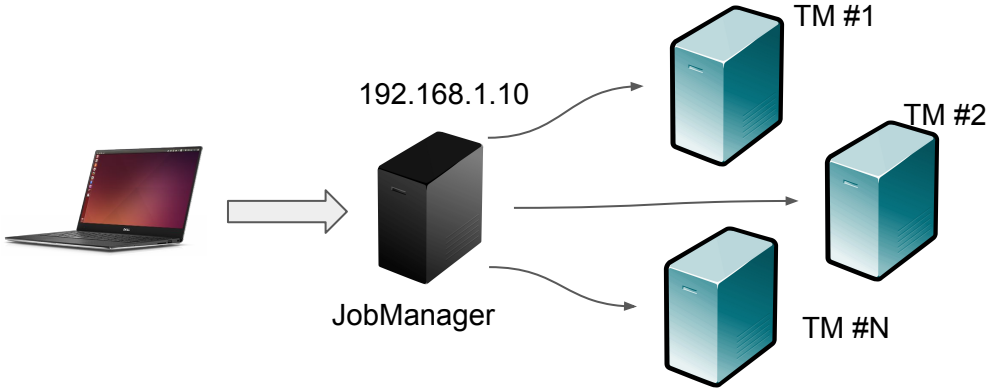
Operation	Description
flink.reduce	Reduction operation (blocking and non-blocking versions)
flink.readTextFile	Read from HDFS (non-blocking)
flink.filter	Filter data according to the function (blocking and non-blocking)
flink.arrayMap	Map with bigger chunks per Flink thread
flink.connectToJobManager	Establish the connection with the main server
flink.setParallelism	Set the parallelism degree for future Flink operations

FastR + Flink : Execution model

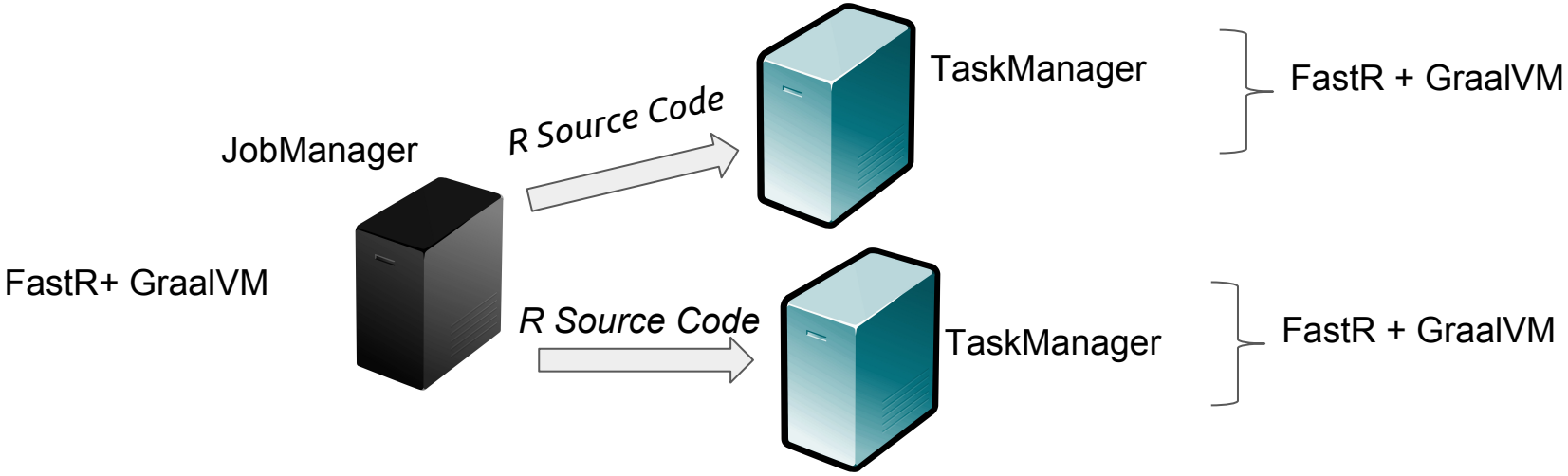
Cool, so how does it work?

```
ipJobManager <- "192.168.1.10"
flink.connectToJobManager(ipJobManager)
flink.setParallelism(2048)
userFunction <- function(x) {
      x * x
}
result <- flink.sapply(1:10000, userFunction)
```

Where R you?

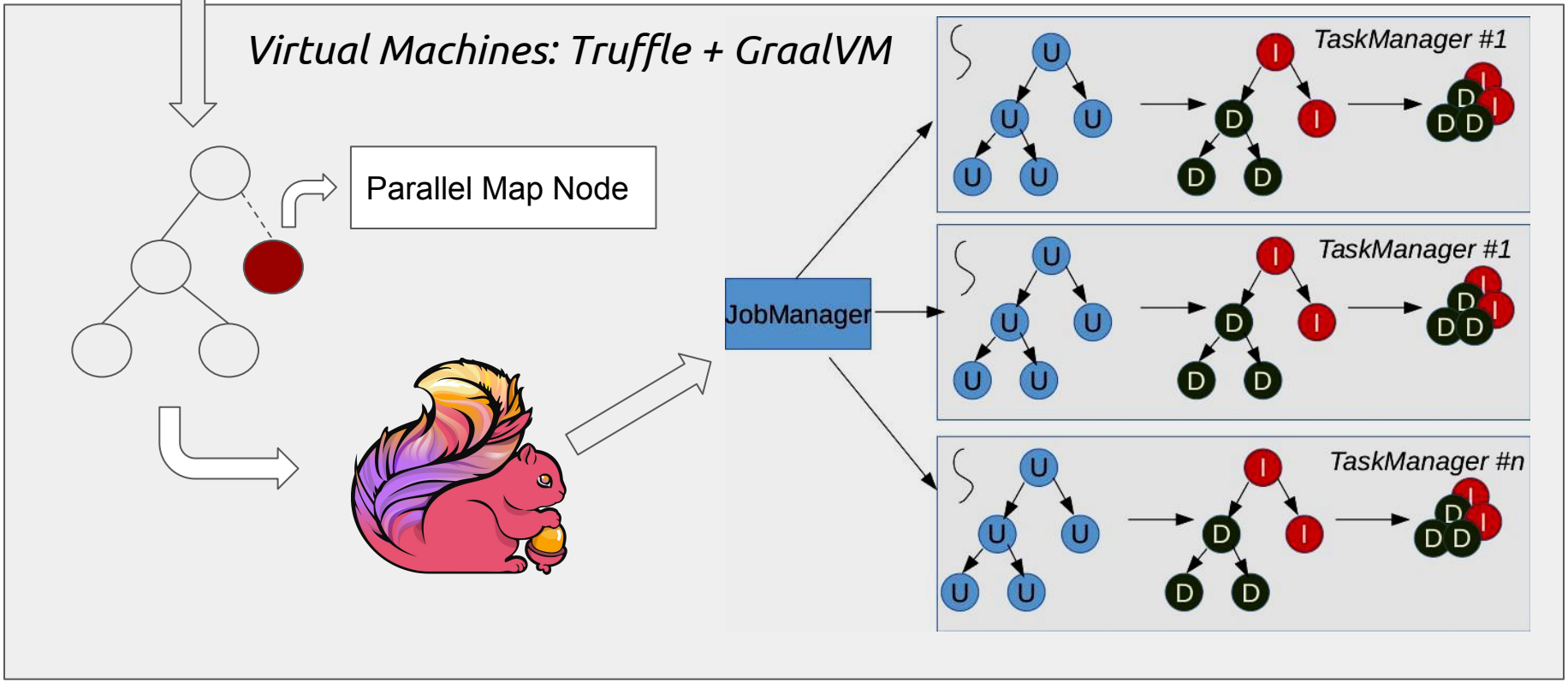


VMs cluster organization

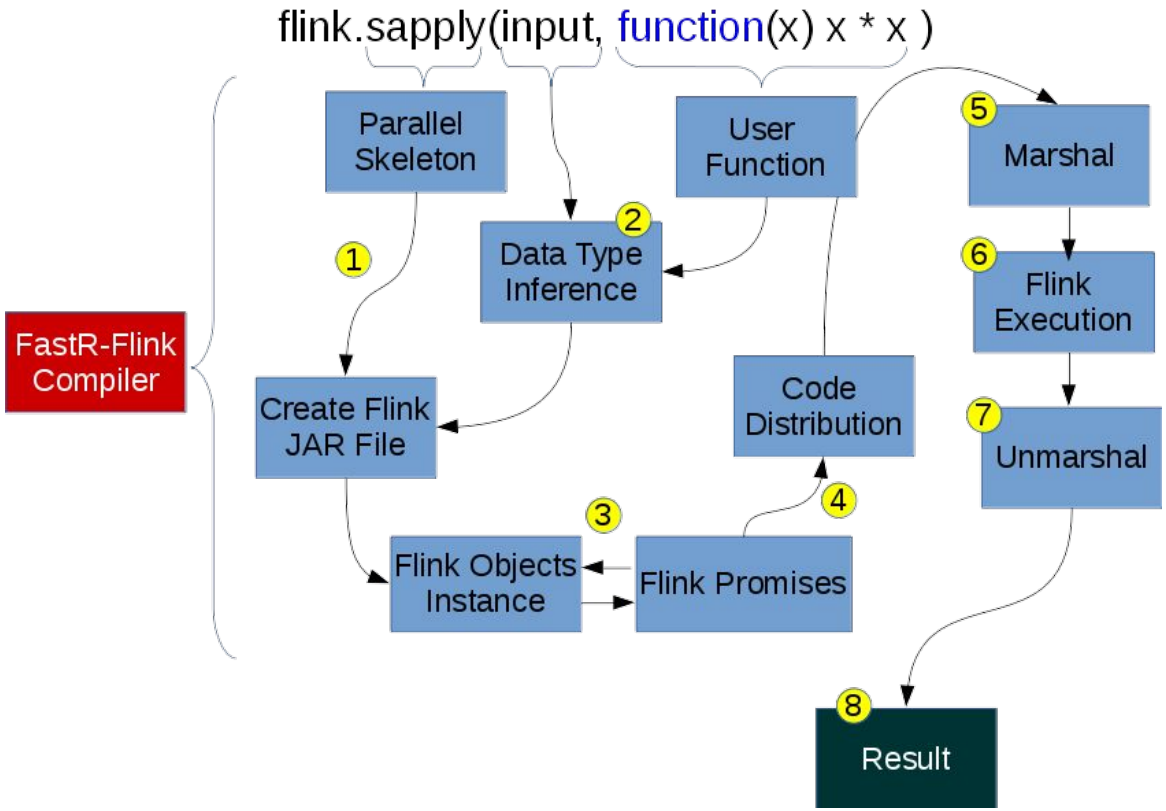


R source code

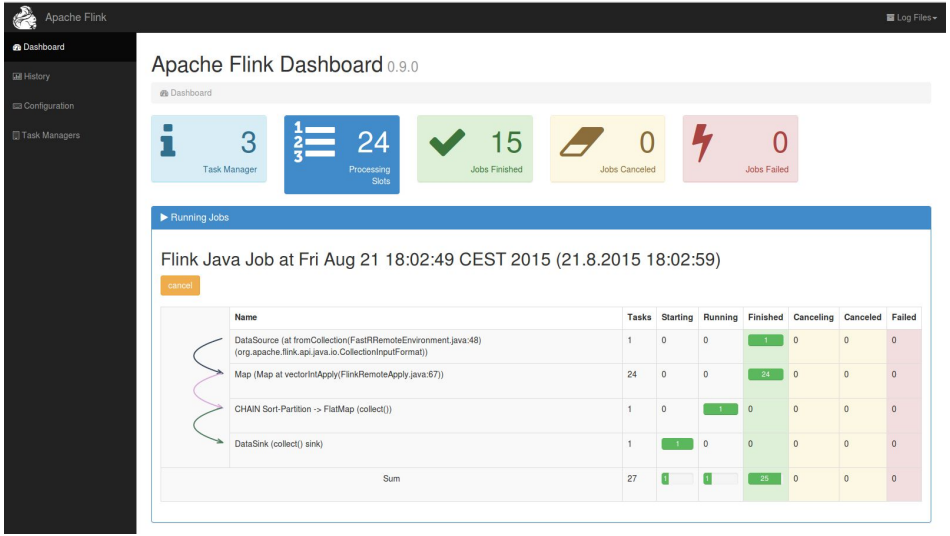
```
> map < – f l i n k . sapply ( input , function(x) x * x ))
```



FastR-Flink Execution Workflow



Flink Web Interface



- It works with FastR!!!
- Track Jobs
- Check history
- Check configuration

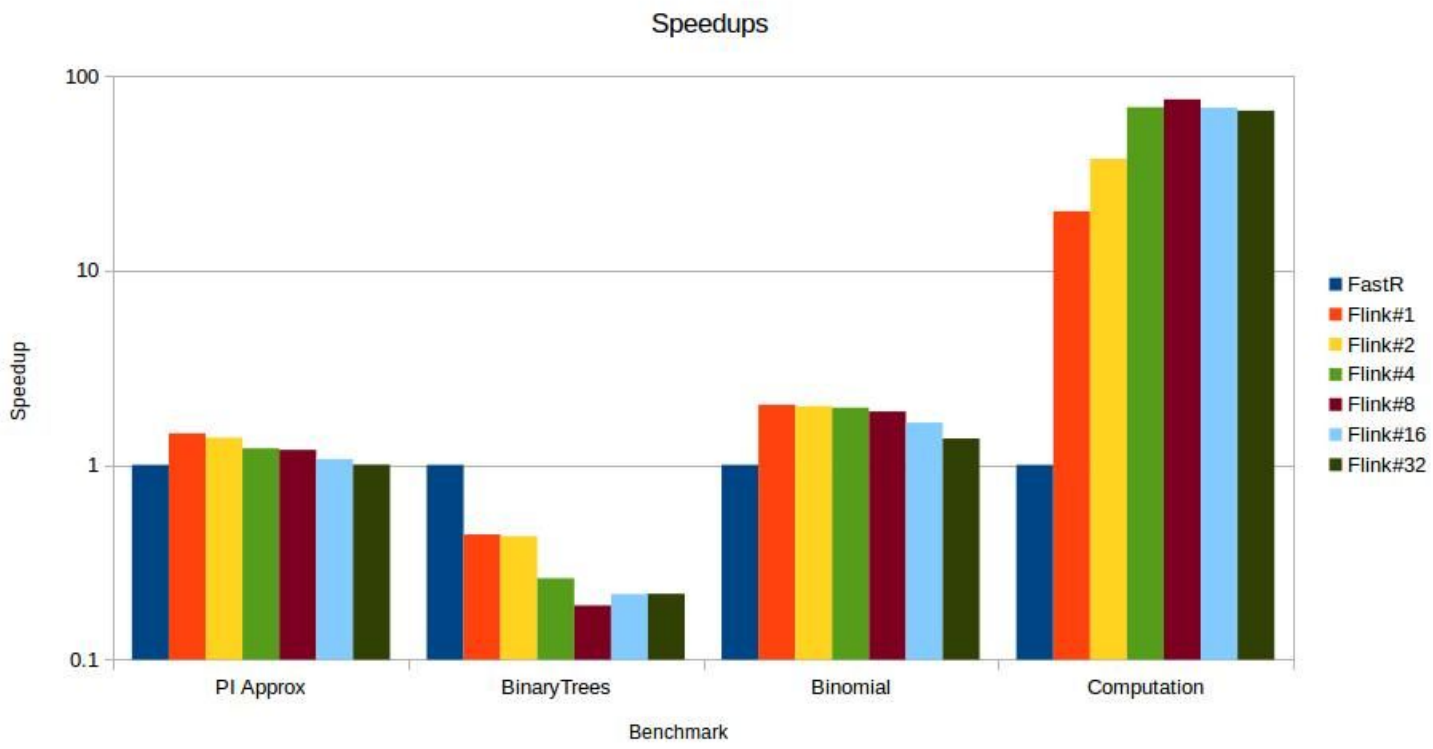
But, Java trace.

- Maybe not very useful for R users.

Preliminary Results

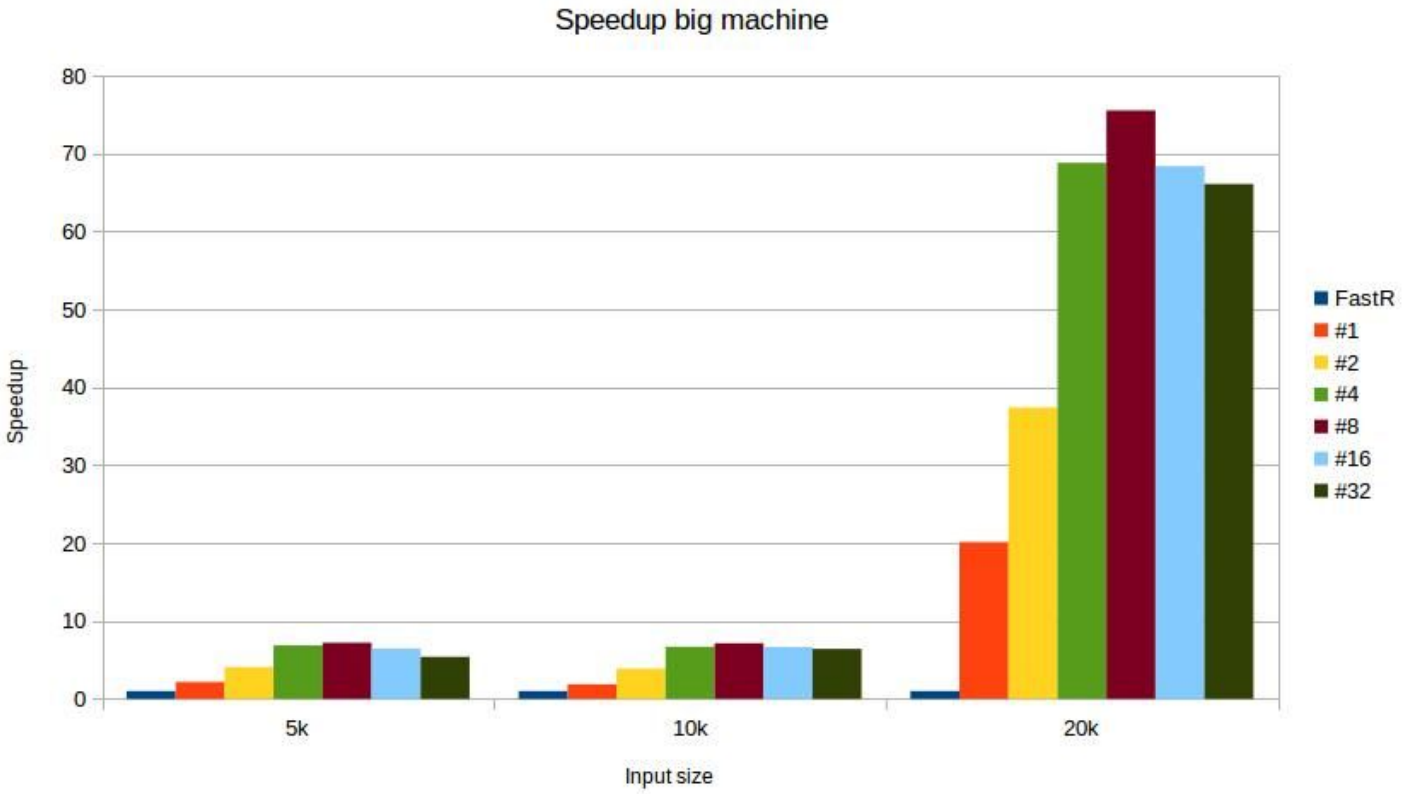
Preliminary Results

- Xeon(R) 32 real cores
- Heap: 12GB
- FastR compiled with OpenJDK 1.8_60



Preliminary Results

- Xeon(R) 32 real cores
- Heap: 12GB
- FastR compiled with OpenJDK 1.8_60



Conclusions and Future work

> sum(Conclusions)

- Prototype: R implementation with + some Flink operations included in the compiler
- FastR-Flink in the compiler
- Easy installation and easy programmability
- Good speedups in shared memory for high computation
- Investigating distributed memory to increase speedups

> sapply(Future Work)

- A world to explore:
  - Solve distribute performance issues
  - More realistic benchmarks (big data)
  - Distributed benchmarks
  - Support more Flink operations
  - Support Flink for other Truffle languages using same approach
  - Define a stable API for high level programming languages
  - Optimisations (caching)
  - GPU support based on our previous work [1]

[1] Juan Fumero, Toomas Remmelg, Michel Steuwer and Christophe Dubach. ***Runtime Code Generation and Data Management for Heterogeneous Computing in Java.*** *2015 International Conference on Principles and Practices of Programming on the Java Platform*

Check it out! It is Open Source

Clone and compile:

```
$ mx sclone https://bitbucket.org/allr/fastr-flink

$ hg update -r rflink-0.1

$ mx build
```

Run:

```
$ mx R      # start the interpreter with Flink local environment
```

Thanks for your attention

> flink.ask(questions)

AND DEMO !

Contact: Juan Fumero <juan.fumero@ed.ac.uk>

Thanks to Oracle Labs, TU Berlin and The University of Edinburgh

Oracle Labs

Technische Universität Berlin

THE UNIVERSITY of EDINBURGH