Efficient Distributed R Dataframes on Apache Flink

Andreas Kunft, Jens Meiners, Tilmann Rabl, Volker Markl

- R got huge traction
  - Open source
  - Rich support for analytics & statistics
- But, standalone not well suited for out of core data loads
- Multiple extensions for distributed execution
  - Hadoop + R
  - Spark + R
  - SystemML

Our Goals

1. Provide API with natural feeling

```
df    <- select(df, f = df$flights, df$distance)
df$km <- df$miles * 1.6
df    <- apply(df, key = id, aggFunc)
```

2. Achieve comparable performance as native dataflow system

General Approach

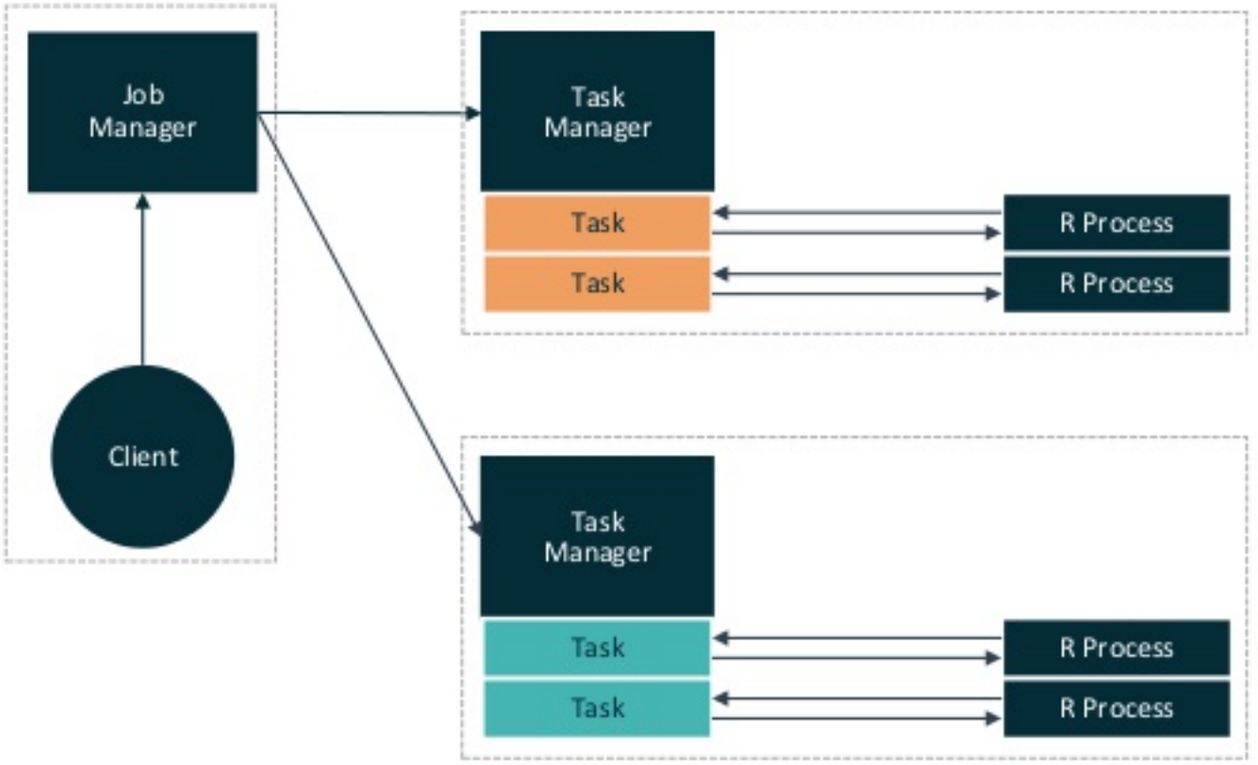
- R dataframe(T_1,T_2,…,T_N) as DataSet<Tuple*N*<T_1,T_2,…,T_N>>
- Create execution plan
  - Map R dataframe functions to the native API whenever possible
    e.g., select *to* projections
  - Call **user defined R functions** within the worker nodes

General Approach

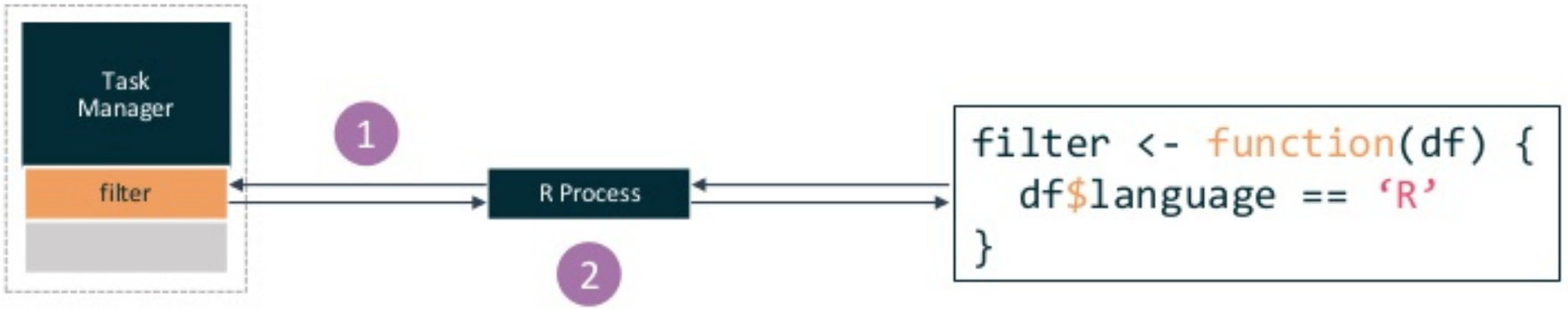
- R dataframe(T_1,T_2,…,T_N) as DataSet<Tuple*N*<T_1,T_2,…,T_N>>
- Create execution plan
  - Map R dataframe functions to the native API whenever possible
    e.g., select *to* projections
  - Call **user defined R functions** within the worker nodes

Handling user defined R functions

Inter Process Communication



Inter Process Communication



1. Communication + Serialization
2. Java and R compete for memory

Source-to-Source Translation

- Translate restrict set of operations to native dataflow API

- Operations are executed natively

```
df <- filter(
  df,
  df$language == 'R'
)
```

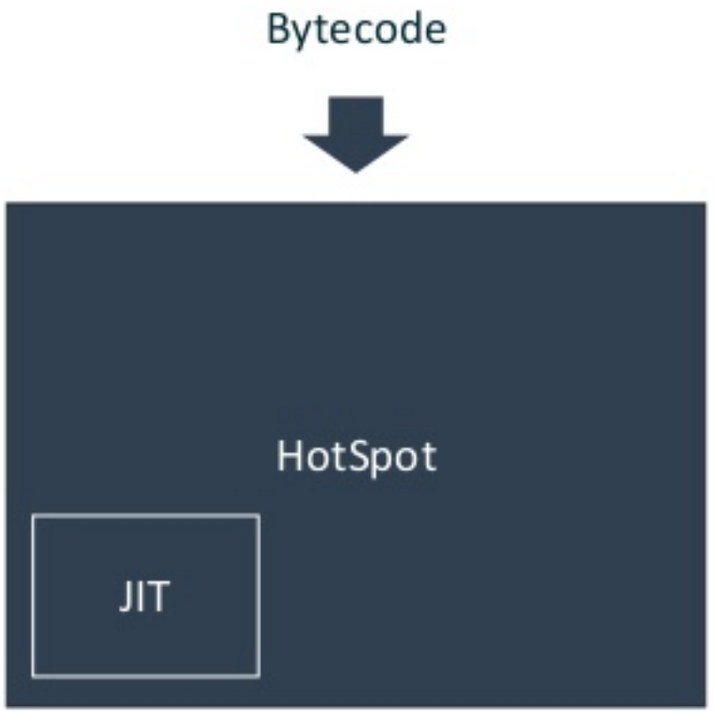
```
val df = df.filter($"language" === "R")
```

```
df$km <- df$miles * 1.6
```

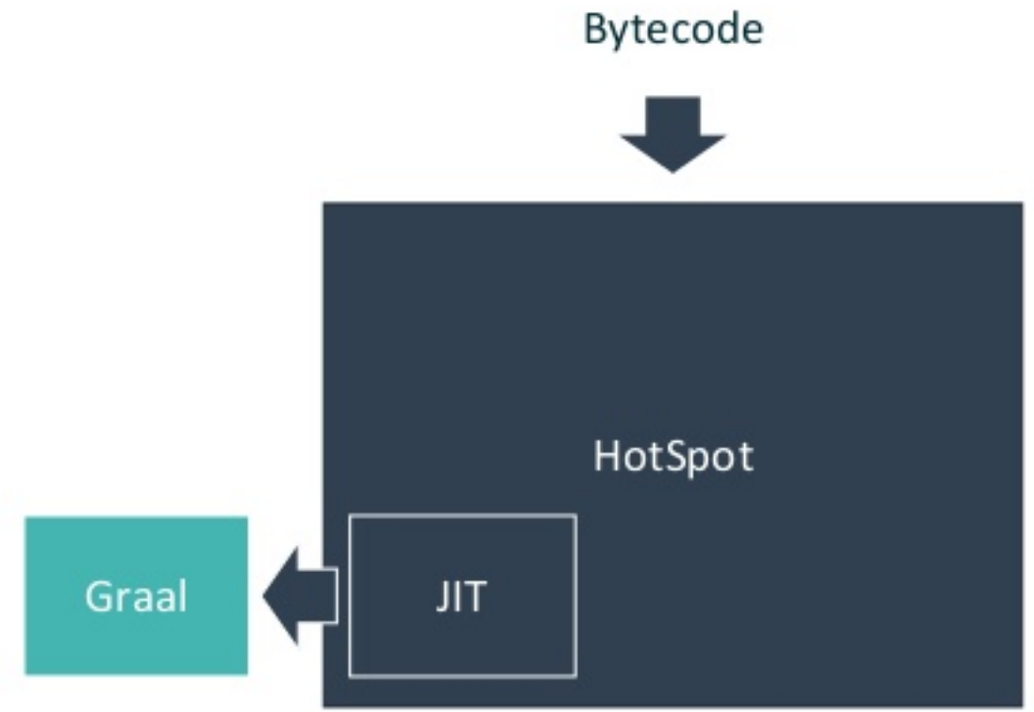
```
val df = df.withColumn("km", $"miles" * 1.6)
```

Flink + fastR

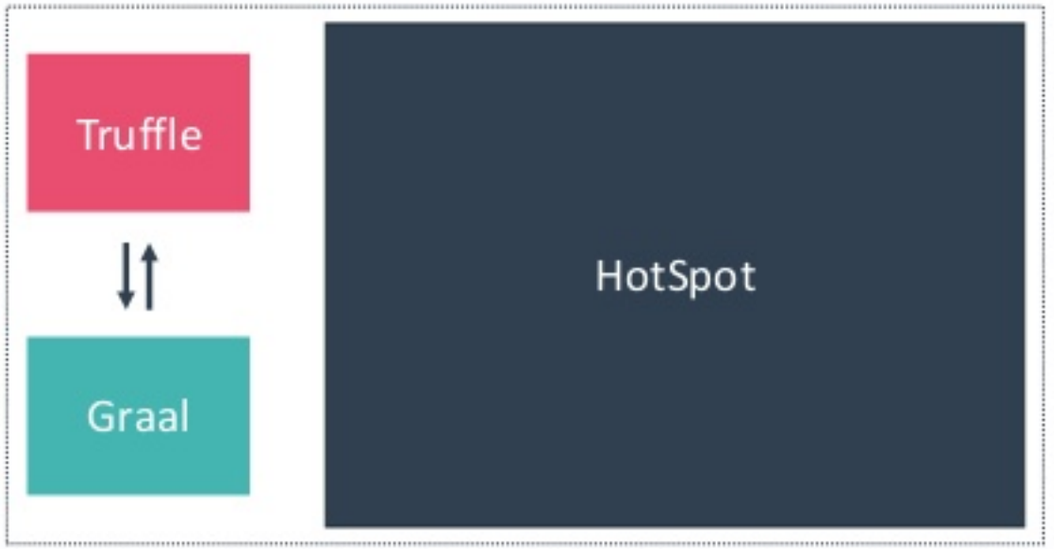
Truffle/Graal



Truffle/Graal



Truffle/Graal



GraalVM

Truffle/Graal

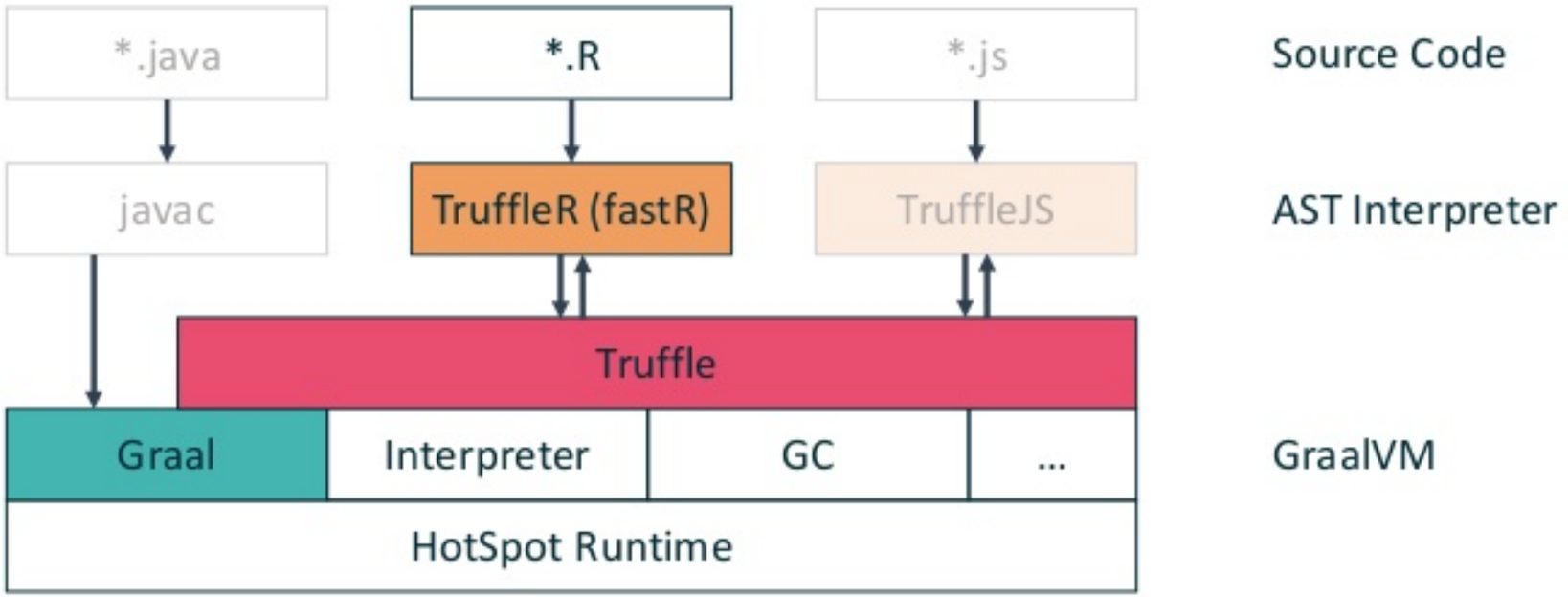


Figure based on: Grimmer, Matthias, et al. "High-performance cross-language interoperability in a multi-language runtime." *ACM SIGPLAN Notices*. Vol. 51. No. 2. ACM, 2015.

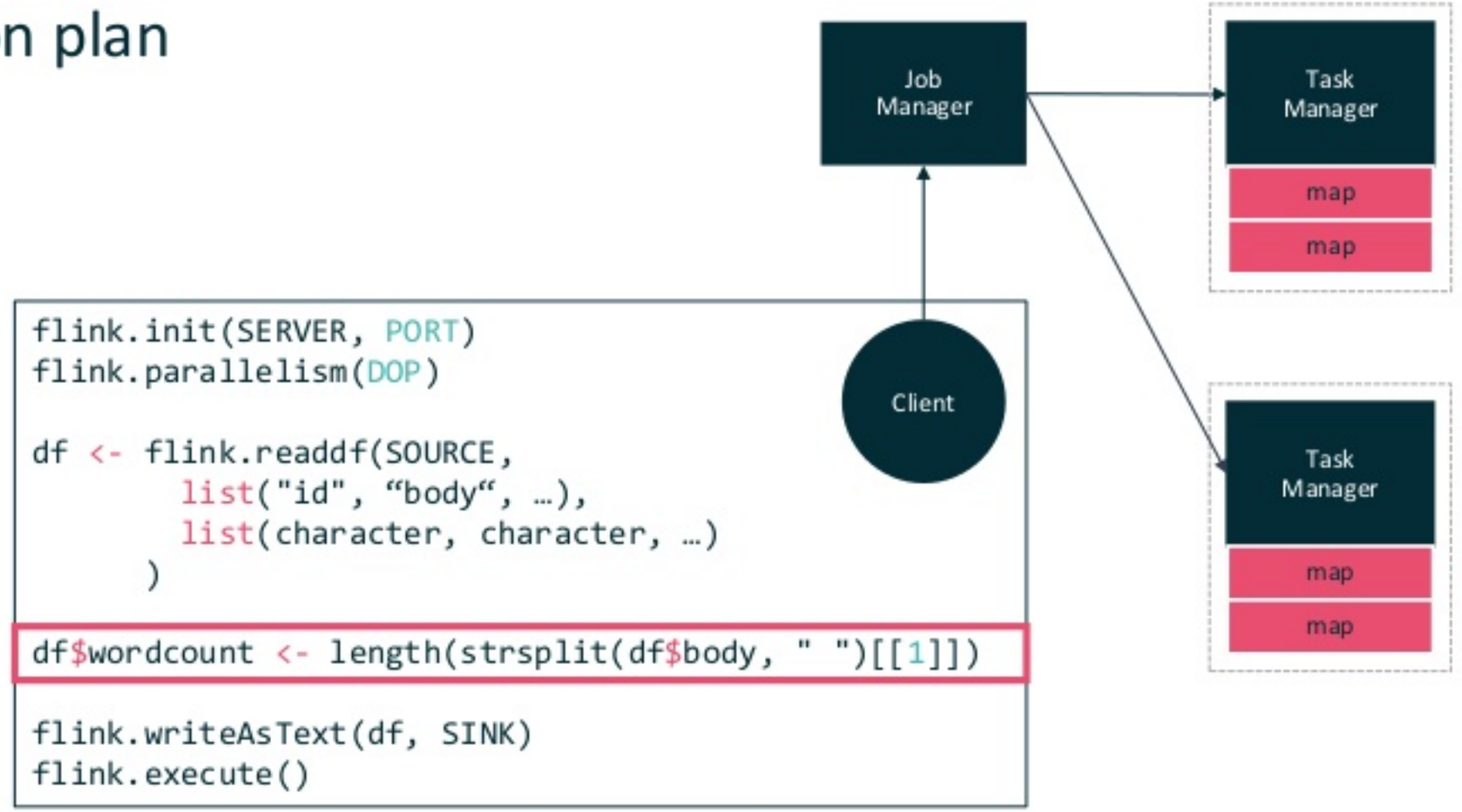
Flink + fastR

fastR: R implementation on top of Truffle/Graal

- Allows us to execute R code in the same VM as Flink
- Infer result types of R functions
- Access Java (Flink) data types in R

Client:

1. Dataframe rows *to* Flink tuples
2. Determine return types of UDFs
3. Create execution plan

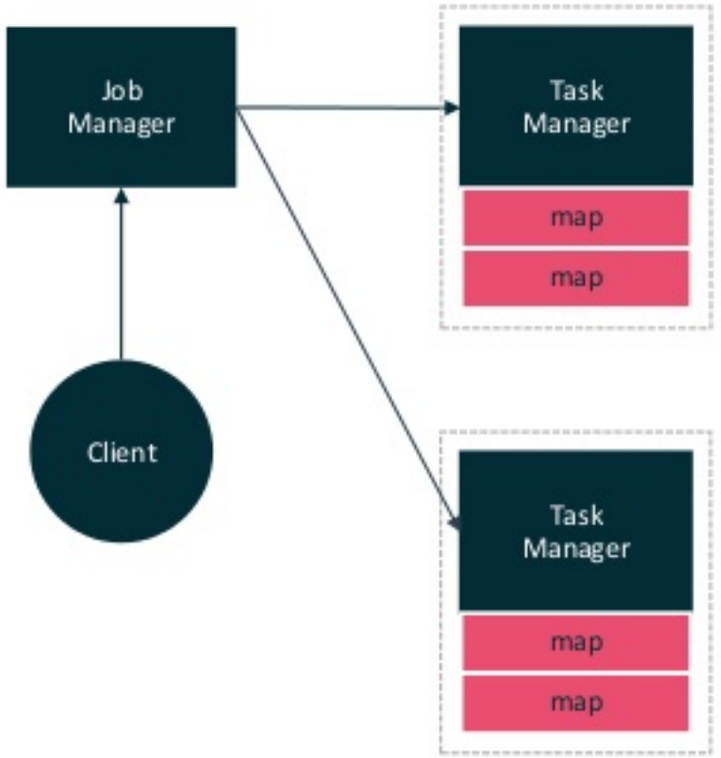


```
df$wordcount <- length(strsplit(df$body, " ")[[1]])
```

```
function(tuple) {
  .fun <- function(tuple) { length(strsplit(tuple[[2]], " ")[[1]] }
  flink.tuple(tuple[[1]], tuple[[2]], .fun(tuple))
}
```

1. Dataframe proxy keeps track of columns, provides efficient access
2. Can be extended with new columns
3. Rewrite to directly use Flink tuples

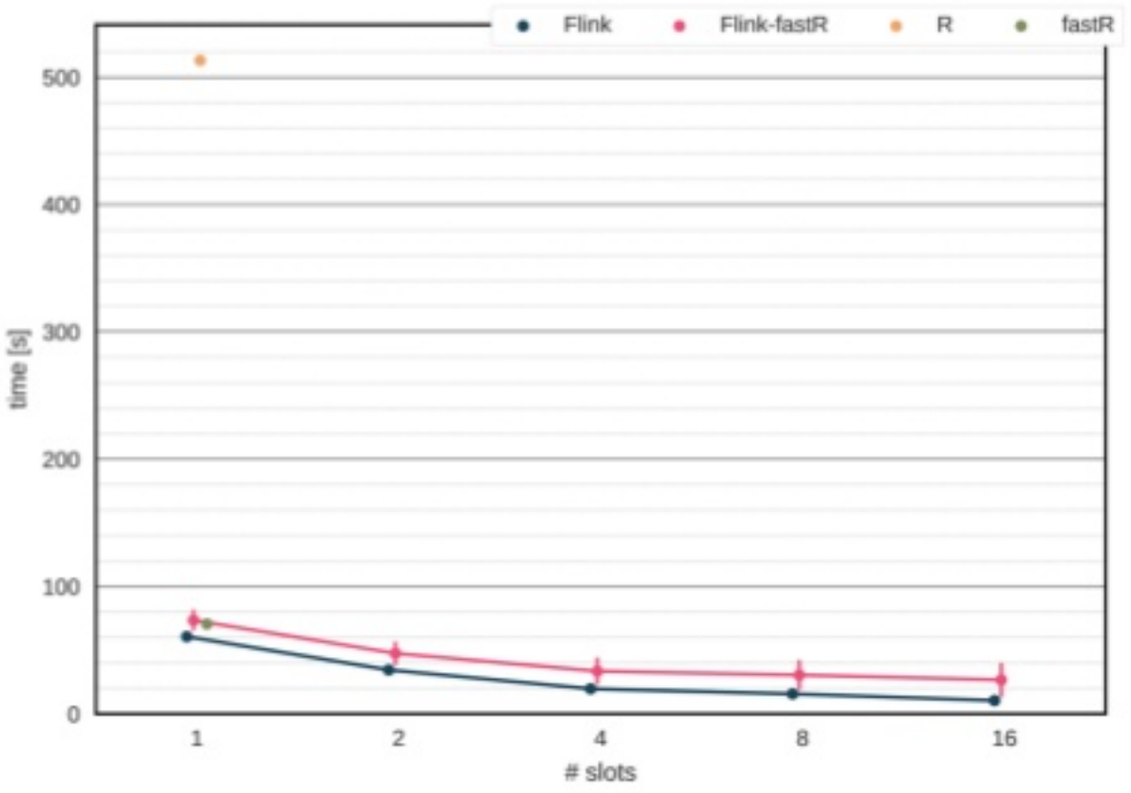
- Task Manager:
  Evaluate R UDF & Execute

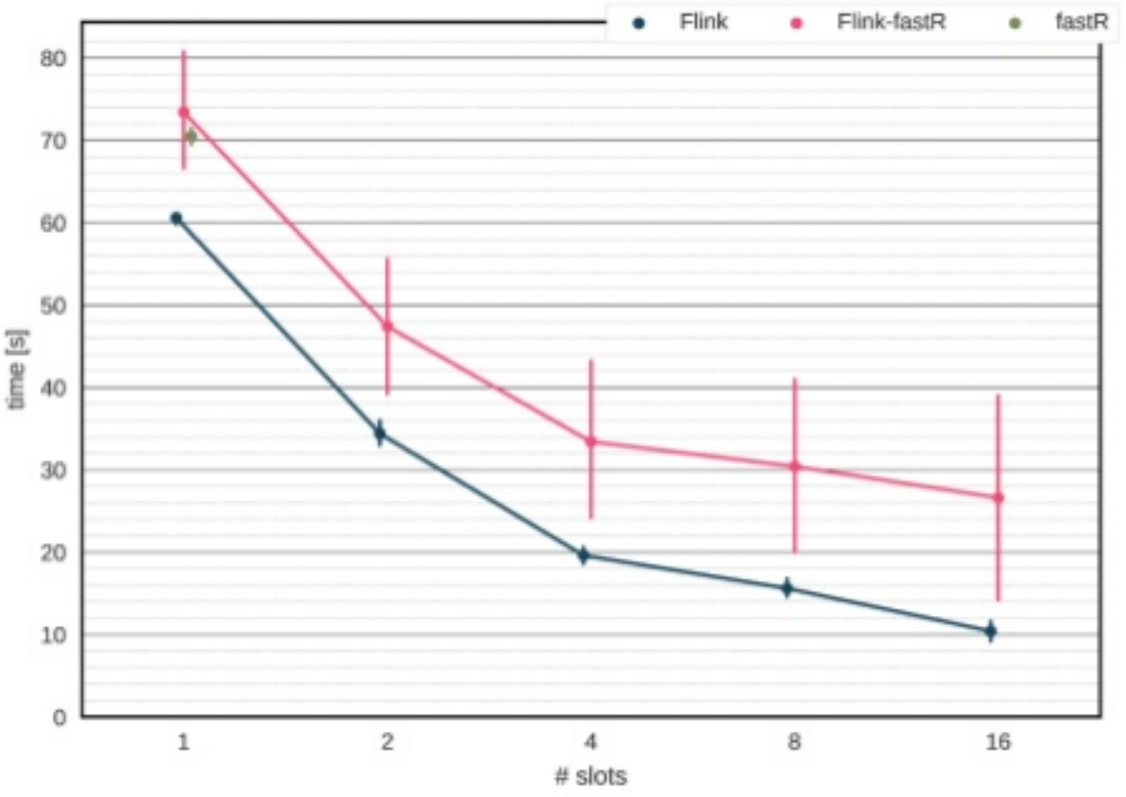



```
map { tuple =>
  executeRFunction(func, tuple)
}
```

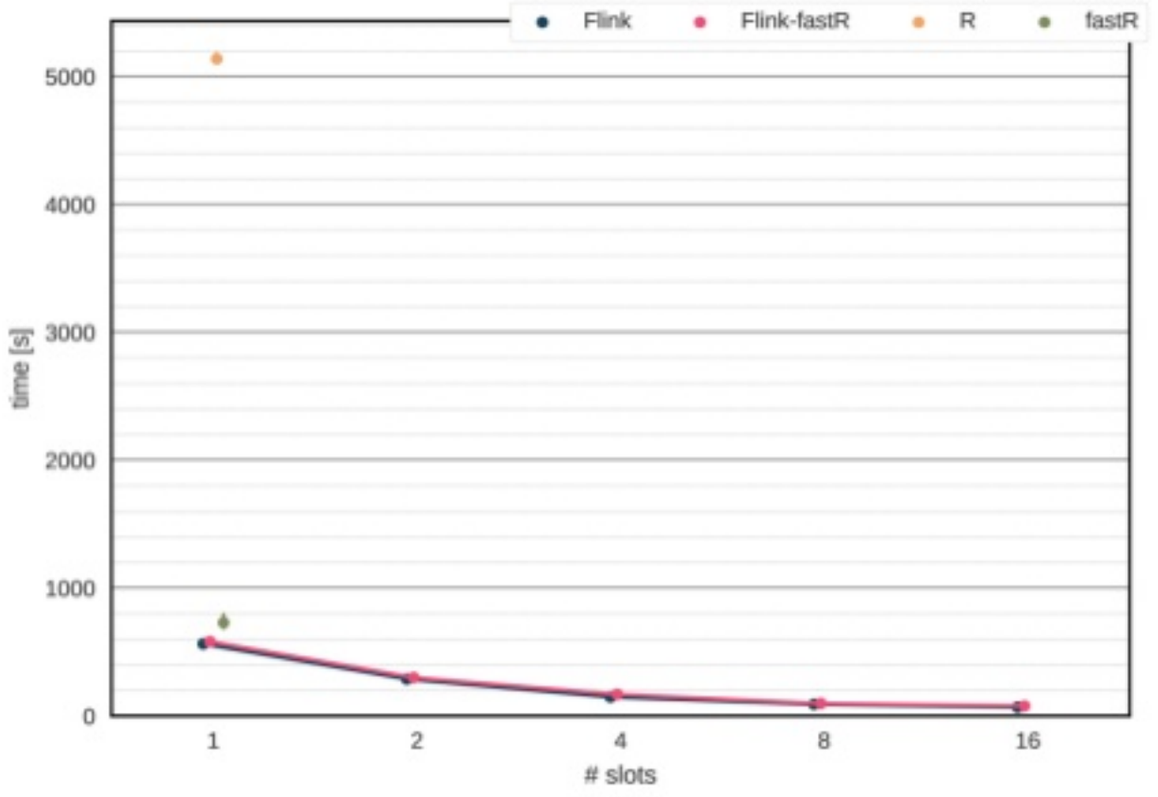
```
map { tuple =>
  executeRFunction(func, tuple)
}
```

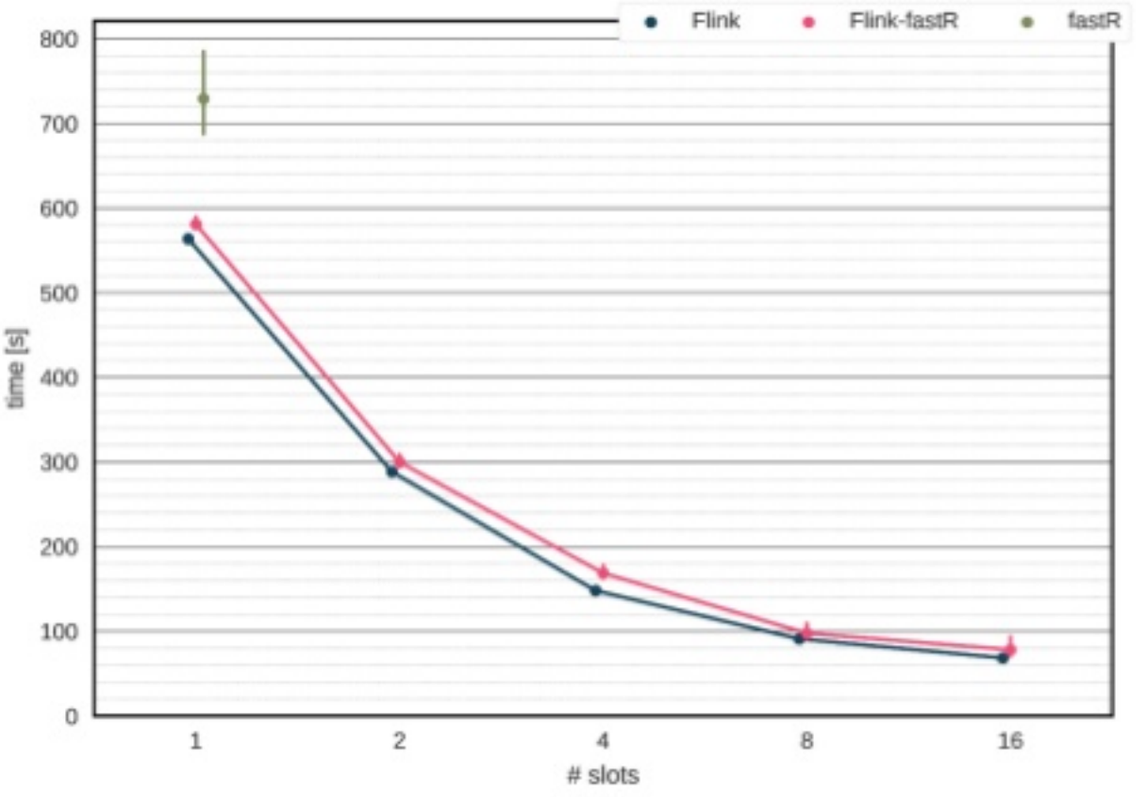
Local - 1.4GB


Flink
Flink-fastR
R
fastR
500
400
300
200
100
0
time [s]
1
2
4
8
16
slots

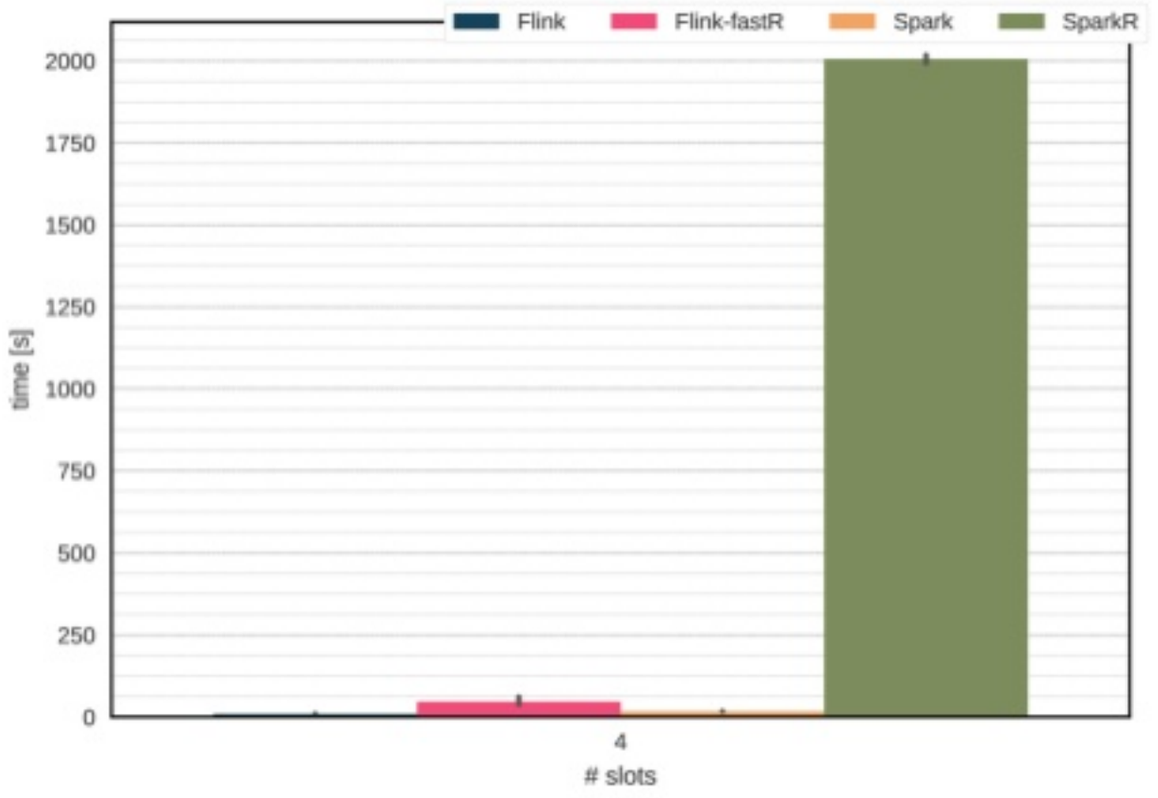

Flink
Flink-fastR
fastR
80
70
60
50
40
30
20
10
0
time [s]
1
2
4
8
16
slots

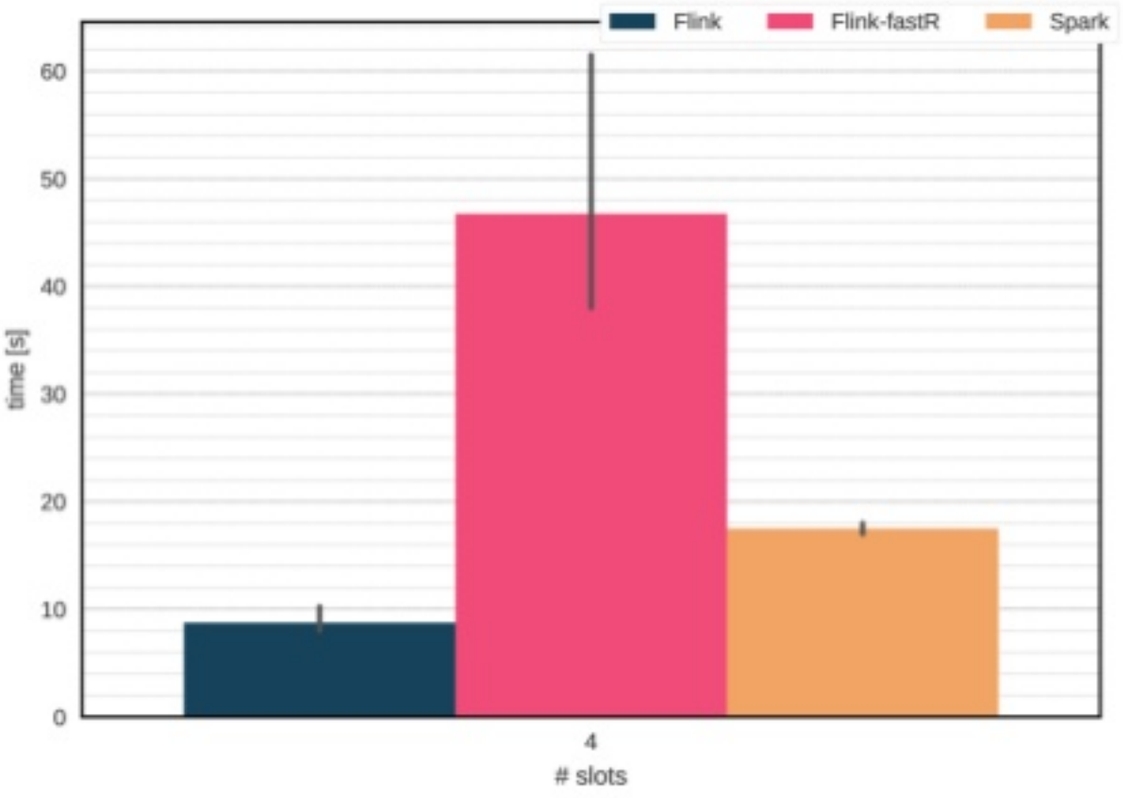
Local - 14GB


Flink
Flink-fastR
R
fastR
5000
4000
3000
2000
1000
0
time [s]
1
2
4
8
16
slots

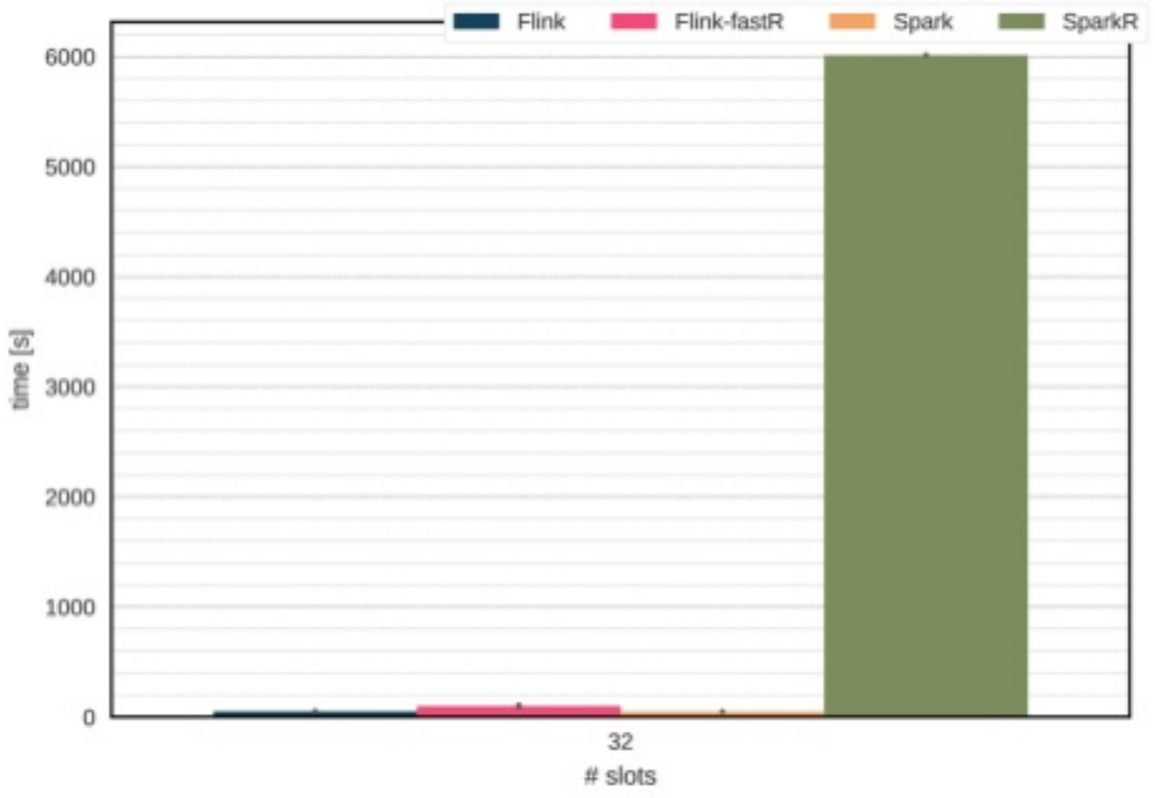

Flink
Flink-fastR
fastR
800
700
600
500
400
300
200
100
0
time [s]
1
2
4
8
16
slots

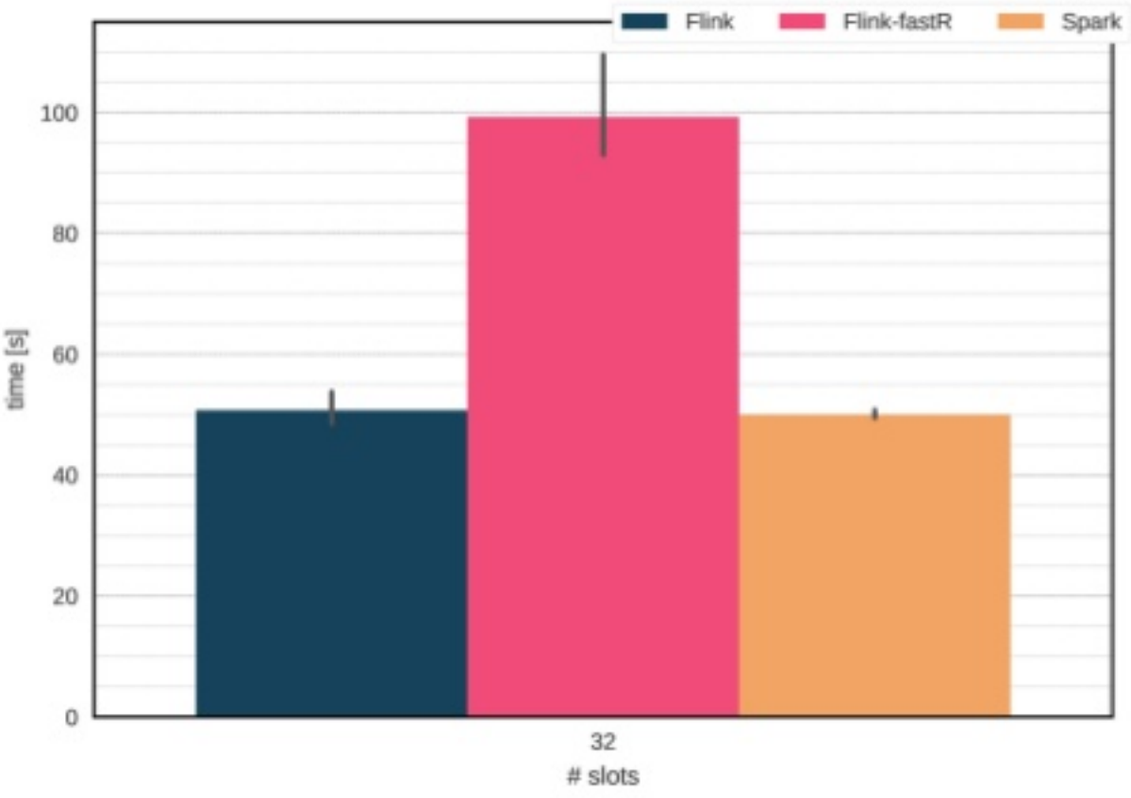
Local – 1GB


Flink
Flink-fastR
Spark
SparkR
2000
1750
1500
1250
1000
750
500
250
0
time [s]
4
slots


Flink
Flink-fastR
Spark
60
50
40
30
20
10
0
time [s]
4
slots

Cluster – 10GB





fastR + Flink

- R dataframe abstraction for distributed computation
- Performance gains even on single node (local mode)
- Approaches native performance even for R UDFs
- Interesting opportunities for:
  - Streaming
  - Other dynamic languages
  - Dynamic Re-optimization

Thank you for your attention!