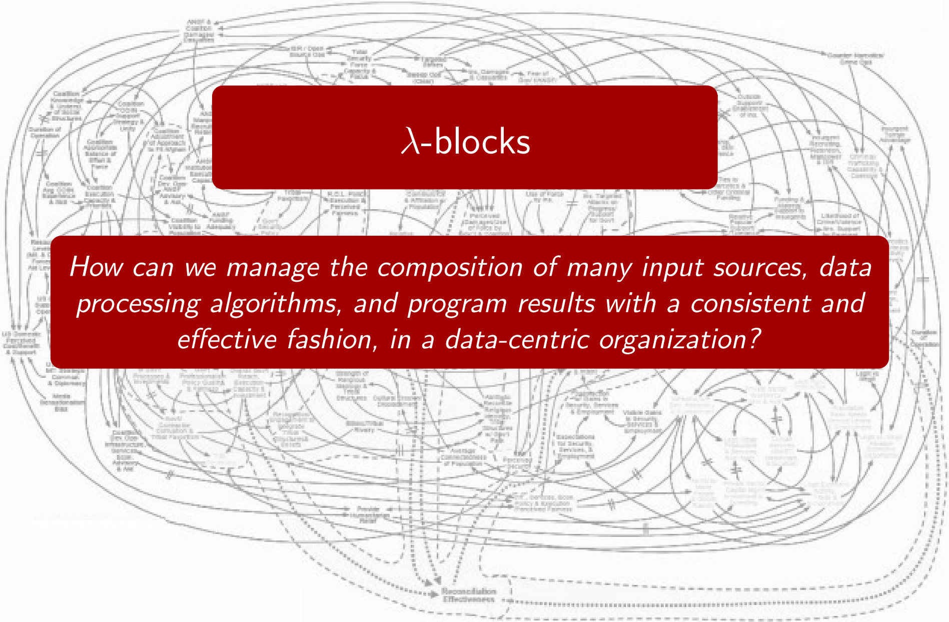
λ-blocks: Data Processing with Topologies of Blocks

Matthieu Caneill, Noël De Palma

July 3, 2018

IEEE Services — Big Data Congress

GOTO E SPAGHETTI CODE



λ-blocks

How can we manage the composition of many input sources, data processing algorithms, and program results with a consistent and effective fashion, in a data-centric organization?

Outline

Introduction

Architecture

Topologies and blocks

Graph manipulations

Evaluation

Conclusion

Outline

Introduction

Architecture

Topologies and blocks

Graph manipulations

Evaluation

Conclusion

Design goals

- A data processing abstraction

Design goals

Design goals

- A data processing abstraction
- A graph of code blocks to represent an end-to-end processing system

Design goals

Design goals

- A data processing abstraction
- A graph of code blocks to represent an end-to-end processing system
- Separation of concerns: low-level data operations, high-level data processing programs

Design goals

Design goals

- A data processing abstraction
- A graph of code blocks to represent an end-to-end processing system
- Separation of concerns: low-level data operations, high-level data processing programs
- Maximize reuse of code

Design goals

Design goals

- A data processing abstraction
- A graph of code blocks to represent an end-to-end processing system
- Separation of concerns: low-level data operations, high-level data processing programs
- Maximize reuse of code
- Compatible with existing (specialized) frameworks and possibility to mix them

Design goals

Design goals

- A data processing abstraction
- A graph of code blocks to represent an end-to-end processing system
- Separation of concerns: low-level data operations, high-level data processing programs
- Maximize reuse of code
- Compatible with existing (specialized) frameworks and possibility to mix them
- Graph manipulation toolkit

Design goals

Design goals

- A data processing abstraction
- A graph of code blocks to represent an end-to-end processing system
- Separation of concerns: low-level data operations, high-level data processing programs
- Maximize reuse of code
- Compatible with existing (specialized) frameworks and possibility to mix them
- Graph manipulation toolkit
- Bring simplicity to large-scale data processing

Outline

Introduction

Architecture

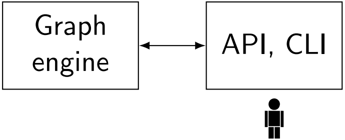
Topologies and blocks

Graph manipulations

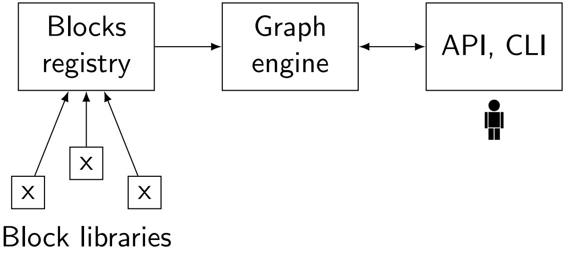
Evaluation

Conclusion

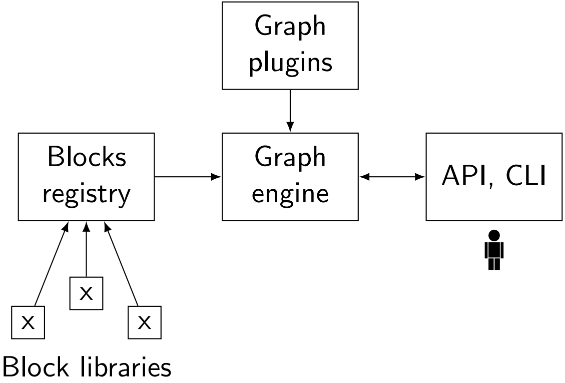
Architecture



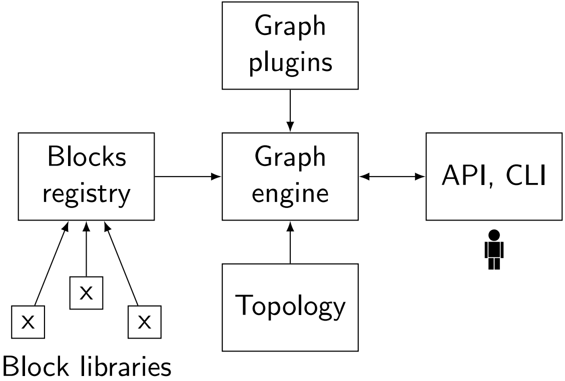
Architecture



Architecture



Architecture



Outline

Introduction

Architecture

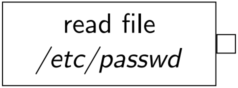
Topologies and blocks

Graph manipulations

Evaluation

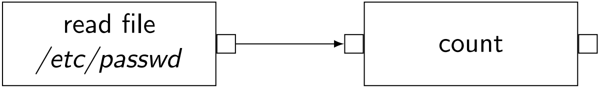
Conclusion

Topologies

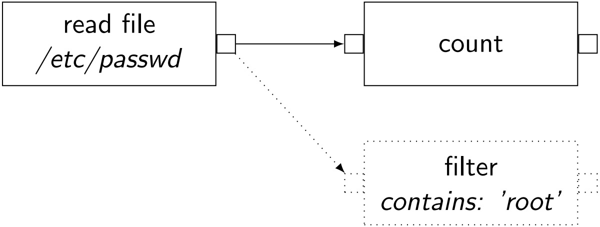


read file
/etc/passwd

Topologies



Topologies



Topologies

```python
"""Counts system users.
"""

def main():
    with open('/etc/passwd') as f:
        return len(f.readlines())

if __name__ == '__main__':
    print(main())
```

Topologies

```python
"""Counts system users.
"""


def main():
    with open('/etc/passwd') as f:
        return len(f.readlines())

if __name__ == '__main__':
    print(main())
```

```
$ wc -l /etc/passwd
```

Topologies

```python
"""Counts system users.
"""

def main():
    with open('/etc/passwd') as f:
        return len(f.readlines())

if __name__ == '__main__':
    print(main())
```

~~$ wc -l /etc/passwd~~

Topologies

```
---
name: count_users
description: Count number of system users
modules: [lb.blocks.foo]
---
- block: readfile
  name: my_readfile
  args:
    filename: /etc/passwd

- block: count
  name: my_count
  inputs:
    data: my_readfile.result
```

Blocks

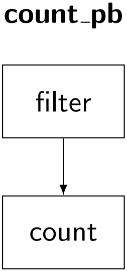
- read_http
- plot_bars
- show_console
- write_line
- write_lines
- split
- concatenate
- map_list
- flatMap
- flatten_list
- group_by_count
- sort
- get_spark_context
- spark_readfile
- spark_text_to_words
- spark_map
- spark_filter
- spark_flatMap
- spark_mapPartitions
- spark_sample
- spark_union
- spark_intersection
- spark_distinct
- spark_groupByKey
- spark_reduceByKey
- spark_aggregateByKey
- spark_sortByKey
- spark_join
- spark_cogroup
- spark_cartesian
- spark_pipe
- spark_coalesce
- spark_repartition
- spark_reduce
- spark_collect
- spark_count
- spark_first
- spark_take
- spark_takeSample
- spark_takeOrdered
- spark_saveAsTextFile
- spark_countByKey
- spark_foreach
- spark_add
- spark_swap
- twitter_search
- cat
- grep
- cut
- head
- tail

Blocks

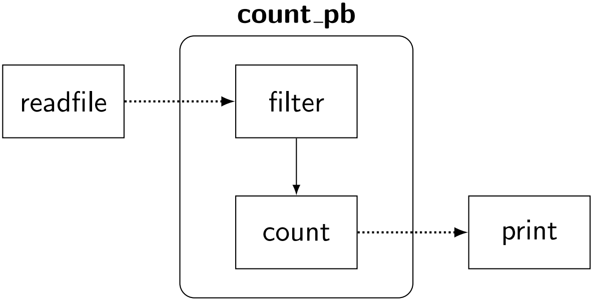
```python
@block(engine='localpython')
def take(n: int=0):
    """Truncates a list of integers.

    :param int n: The length of the desired result.
    :input List[int] data: The list of items to truncate.
    :output List[int] result: The truncated result.
    """
    def inner(data: List[int])->ReturnType[List[int]]:
        assert n <= len(data)
        return ReturnEntry(result=data[:n])
    return inner
```

Sub-topologies



Sub-topologies



Sub-topologies

```
---
name: count_pb
---
- block: filter
  name: filter
  args:
    contains: error
  inputs:
    data: $inputs.data
- block: count
  name: count
  inputs:
    data: filter.result
```

Sub-topologies

```
---
name: count_pb
---
- block: filter
  name: filter
  args:
    contains: error
  inputs:
    data: $inputs.data
- block: count
  name: count
  inputs:
    data: filter.result
```

```
---
name: foo_errors
---
- block: readfile
  name: readfile
  args:
    filename: foo.log

- topology: count_pb
  name: count_pb
  bind_in:
    data: readfile.result
  bind_out:
    result: count.result

- block: print
  name: print
  inputs:
    data: count_pb.result
```

Outline

Introduction

Architecture

Topologies and blocks

Graph manipulations

Evaluation

Conclusion

Graph manipulations

- Verification (e.g. type checking)

Graph manipulations

- Verification (e.g. type checking)
- Instrumentation

Graph manipulations

- Verification (e.g. type checking)
- Instrumentation
- Caching

Graph manipulations

- Verification (e.g. type checking)
- Instrumentation
- Caching
- Debugging tools

Graph manipulations

- Verification (e.g. type checking)
- Instrumentation
- Caching
- Debugging tools
- Optimizations

Graph manipulations

- Verification (e.g. type checking)
- Instrumentation
- Caching
- Debugging tools
- Optimizations
- Monitoring

Graph manipulations

- Verification (e.g. type checking)
- Instrumentation
- Caching
- Debugging tools
- Optimizations
- Monitoring
- Program reasoning and semantics

Graph manipulations

- Reasoning on the computation graph as a high-level object

Graph manipulations

- Reasoning on the computation graph as a high-level object
- Plugin system

Graph manipulations

- Reasoning on the computation graph as a high-level object
- Plugin system
- Hooks:
  - `before_graph_execution`
    pre-processing, optimizations, verifications

Graph manipulations

- Reasoning on the computation graph as a high-level object
- Plugin system
- Hooks:
  - `before_graph_execution`
    pre-processing, optimizations, verifications
  - `after_graph_execution`
    post-processing

Graph manipulations

- Reasoning on the computation graph as a high-level object
- Plugin system
- Hooks:
  - `before_graph_execution`
    pre-processing, optimizations, verifications
  - `after_graph_execution`
    post-processing
  - `before_block_execution`
    observation, optimizations

Graph manipulations

- Reasoning on the computation graph as a high-level object
- Plugin system
- Hooks:
  - `before_graph_execution`
    pre-processing, optimizations, verifications
  - `after_graph_execution`
    post-processing
  - `before_block_execution`
    observation, optimizations
  - `after_block_execution`
    observation

Graph manipulation example: instrumentation (excerpt)

```
by_block = {} # timing by block: begin, duration

@before_block_execution
def store_begin_time(block):
    name = block.fields['name']
    by_block[name]['begin'] = time.time()
```

Graph manipulation example: instrumentation (excerpt)

```
by_block = {} # timing by block: begin, duration

@before_block_execution
def store_begin_time(block):
    name = block.fields['name']
    by_block[name]['begin'] = time.time()


@after_block_execution
def store_end_time(block, results):
    name = block.fields['name']
    by_block[name]['duration'] = \
      time.time() - by_block[name]['begin']
```

Graph manipulation example: instrumentation (excerpt)

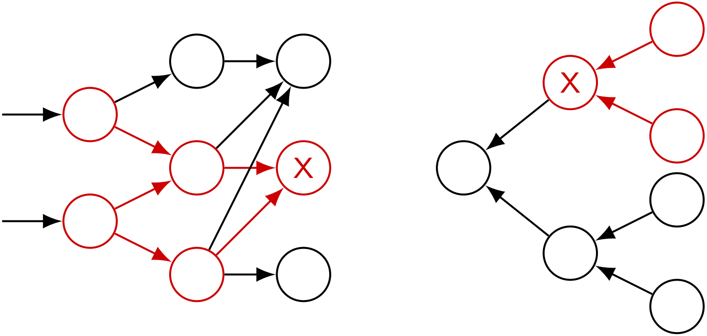
```python
@after_graph_execution
def show_times(results):
    longest_first = sorted(by_block, reverse=True)
    for blockname in longest_first:
        print('{}\t{}'.format(
            blockname,
            by_block[blockname]['duration'])
```

Graph manipulation example: instrumentation

block	duration (ms)
read http	818
write lines	54
grep	49
split	20

λ-blocks

Graph manipulation example: caching



$$\begin{aligned} H(B) = h(&B.name, && \text{block name (not instance name)} \\ &B.args, && \text{list of (name, value) tuples} \\ &B.inputs) && \text{list of (name, H(block), connector) tuples} \end{aligned}$$

Outline

Introduction

Architecture

Topologies and blocks

Graph manipulations

Evaluation

Conclusion

Evaluation

Setup

- Wordcount over https: local machine, 8 cores, 16 GB RAM
- Wordcount over disk: local machine, 8 cores, 16 GB RAM
- PageRank on Spark: Spark on 1 server (24 cores, 128 GB RAM)

Evaluation

Performances

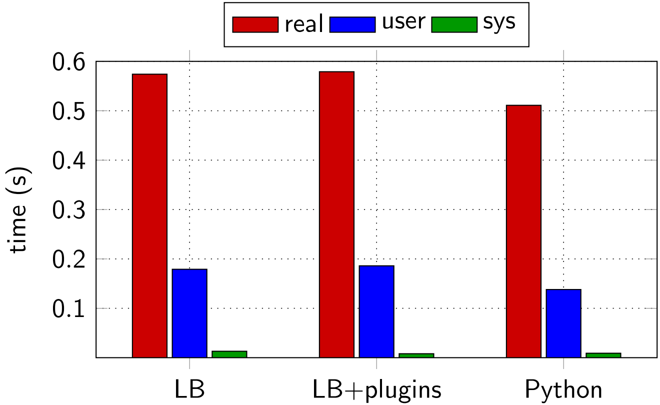


Figure: Wordcount over https: Twitter feed.

Performances

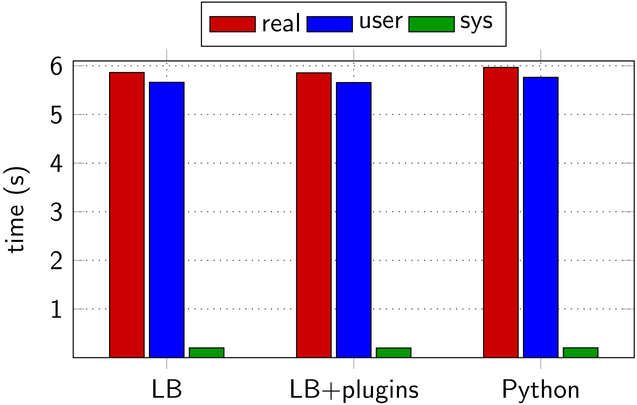


Figure: Wordcount over disk: Wikipedia dataset.

Evaluation

Performances

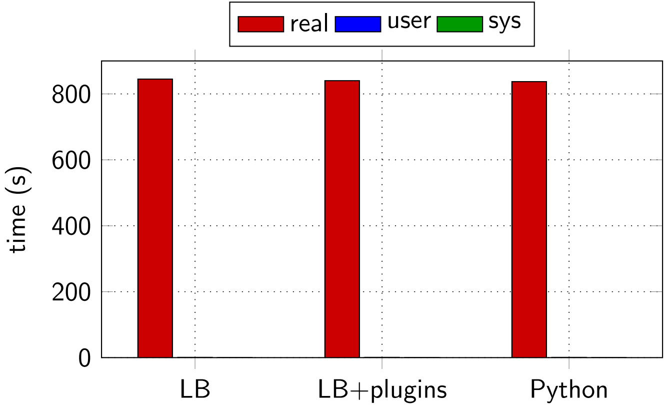
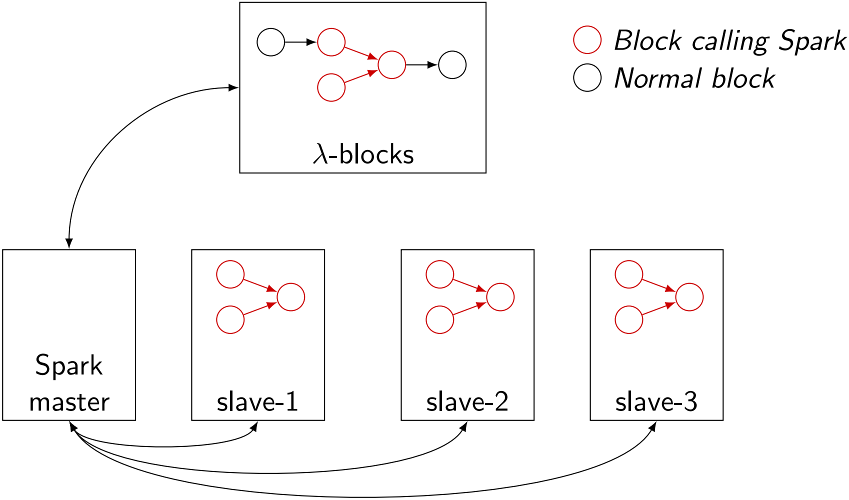


Figure: PageRank on Wikipedia hyperlinks with Spark.

Evaluation: using a Spark cluster



Evaluation

Maximum overhead measured per topology: 50 ms

Outline

Introduction

Architecture

Topologies and blocks

Graph manipulations

Evaluation

Conclusion

Conclusion

λ-blocks enables:

- decoupling between standalone pieces of code which transform data, and data processing algorithms;
- reasoning on a high-level abstraction of a data processing program;
- reusing everything (code, topologies, specialized frameworks).

Related work

Dataflow programming

- ML pipelines: scikit-learn [PVG^{+}11], Spark [The17a], Orange framework [DCE^{+}13]
- Real-time: Apache Beam [apa], StreamPipes [RKHS15]

Related work

Dataflow programming

- ML pipelines: scikit-learn [PVG+11], Spark [The17a], Orange framework [DCE+13]
- Real-time: Apache Beam [apa], StreamPipes [RKHS15]

Blocks programming

- Recognition over recall, immediate feedback [BGK+17]

Related work

Dataflow programming

- ML pipelines: scikit-learn [PVG+11], Spark [The17a], Orange framework [DCE+13]
- Real-time: Apache Beam [apa], StreamPipes [RKHS15]

Blocks programming

- Recognition over recall, immediate feedback [BGK+17]

Graphs from configuration

- Pyleus [Yel16], Storm Flux [The17b]

Related work

Dataflow programming

- ML pipelines: scikit-learn [PVG$^+$11], Spark [The17a], Orange framework [DCE$^+$13]
- Real-time: Apache Beam [apa], StreamPipes [RKHS15]

Blocks programming

- Recognition over recall, immediate feedback [BGK$^+$17]

Graphs from configuration

- Pyleus [Yel16], Storm Flux [The17b]

Other

- “Serverless” architectures and stateless functions [JVSR17]

Future work

- Explore more graph manipulation abstractions (complexity analysis, serialization, verification. . .)
- Streaming and online operations
- Tight integration with clusters (data storage, caches, etc)

Thanks! Questions?

Images credits

- *Goto e spaghetti code*, `http://blogbv2.altervista.org/HD/il-goto-e-la-buona-programmazione-parte-ii/`

Bibliography I

- Apache Beam.
  `https://beam.apache.org/`.

- David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak.
  Learnable programming: Blocks and beyond.
  Commun. ACM, 60(6):72–80, May 2017.

- Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan.
  Orange: Data mining toolbox in python.
  Journal of Machine Learning Research, 14:2349–2353, 2013.

Bibliography II

Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht.
Occupy the cloud: Distributed computing for the 99%.
arXiv preprint arXiv:1702.04024, 2017.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay.
Scikit-learn: Machine learning in Python.
Journal of Machine Learning Research, 12:2825–2830, 2011.

Bibliography III

Dominik Riemer, Florian Kaulfersch, Robin Hutmacher, and Ljiljana Stojanovic.
Streampipes: solving the challenge with semantic stream processing pipelines.
In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pages 330–331. ACM, 2015.

The Apache Spark developers.
ML Pipelines.
https://spark.apache.org/docs/latest/ml-pipeline.html, 2017.

Bibliography IV

The Apache Storm developers.
Flux.
http://storm.apache.org/releases/2.0.0-SNAPSHOT/flux.html, 2017.

YelpArchive.
Pyleus.
https://github.com/YelpArchive/pyleus, 2016.