Contributions to Large-Scale Data Processing Systems
PhD Defense

Matthieu Caneill

February 5, 2018

Daniel Hagimont
INP Toulouse
ENSEEIHT

Jean-Marc Menaud
IMT Atlantique

Sihem Amer-Yahia
CNRS / Université Grenoble Alpes

Noël De Palma
Université Grenoble Alpes
Worldwide data production

- 2008: 0.3 zB
- 2010: 0.3 zB
- 2012: 0.3 zB
- 2014: 5.8 zB

1 zetabyte = 1000 exabytes = 10^6 petabytes = 10^9 terabytes

(1 zetabyte is 2 billion times my hard drive)
Motivation

Worldwide data production

2008 | 2010 | 2012 | 2014
--- | --- | --- | ---
0.3 | 0.3 | 0.3 | 5.8

started PhD

1 zettabyte = 1000 exabytes
1 exabyte = 10^6 petabytes
1 petabyte = 10^9 terabytes

(1 zettabyte is 2 billion times my hard drive)
Motivation

Worldwide data production

- 2008: 0.3 zB
- 2010: 1 zB
- 2012: 3.8 zB
- 2014: 5.8 zB
- 2016: 15 zB
- 2018: 31 zB (est.)

started PhD

1 zB = 1,000,000,000,000,000,000 bytes
1 exabyte = 1,000,000,000,000,000,000,000 bytes
1 petabyte = 1,000,000,000,000,000,000,000,000 bytes
(1 zetabyte is 2 billion times my hard drive)
Worldwide data production

1 zetabyte = 1000 exabytes = 10^6 petabytes = 10^9 terabytes
Motivation

Worldwide data production

1 zetabyte = 1000 exabytes = 10^6 petabytes = 10^9 terabytes

(1 zetabyte is 2 billion times my hard drive)
Applications

- Genome sequencing and querying (human: 3 B base pairs)
Applications

- Genome sequencing and querying (human: 3 B base pairs)
- Web and social networks (Facebook: 600 TB/day in 2014)
Motivation

Applications

- Genome sequencing and querying (human: 3 B base pairs)
- Web and social networks (Facebook: 600 TB/day in 2014)
- Particle physics (CERN: 1 PB/s of collision data)
- etc.
Applications

- Genome sequencing and querying (human: 3 B base pairs)
- Web and social networks (Facebook: 600 TB/day in 2014)
- Particle physics (CERN: 1 PB/s of collision data)
- etc.

Problems

- Data management at scale
Applications

- Genome sequencing and querying (human: 3 B base pairs)
- Web and social networks (Facebook: 600 TB/day in 2014)
- Particle physics (CERN: 1 PB/s of collision data)
- etc.

Problems

- Data management at scale
- Data processing in reasonable time
Motivation

Applications

- Genome sequencing and querying (human: 3 B base pairs)
- Web and social networks (Facebook: 600 TB/day in 2014)
- Particle physics (CERN: 1 PB/s of collision data)
- etc.

Problems

- Data management at scale
- Data processing in reasonable time
- ... and reasonable price
Research questions

How to design...

- An industrial system to handle monitoring data and make predictions about future failures?
How to design…

- An industrial system to handle monitoring data and make predictions about future failures?
- An algorithm to improve locality in distributed streaming engines?
Research questions

How to design...

- An industrial system to handle monitoring data and make predictions about future failures?
- An algorithm to improve locality in distributed streaming engines?
- A framework to compose data processing algorithms in a descriptive fashion, while reasoning on high level abstractions?
Structure of this presentation

1. Online metrics prediction in monitoring systems
2. Locality data routing
3. λ-blocks
4. Conclusion
How to design an industrial system to handle monitoring data and make predictions about future failures?
Actors and roles of Smart Support Center

- **Coservit**: Monitoring services
- **HP**: Cloud computing, hardware
- **LIG – AMA**: Machine learning
- **LIG – ERODS**: Cloud computing, systems
Scope of Smart Support Center

- Monitoring data

- Monitoring insights
Scope of Smart Support Center

- Monitoring insights
- Failure prediction
Scope of Smart Support Center

- Monitoring insights
- Failure prediction
- Infrastructure scaling
Scope of Smart Support Center

- Monitoring insights
- Failure prediction
- Infrastructure scaling
Scope of Smart Support Center

- Monitoring insights
- Failure prediction
- Infrastructure scaling
- More server uptime
Challenges

- Scale monitoring infrastructure (from 1 to $N$ nodes)
Challenges

- Scale monitoring infrastructure (from 1 to $N$ nodes)
- System design for low latency analytics
Challenges

- Scale monitoring infrastructure (from 1 to $N$ nodes)
- System design for low latency analytics
- Fault tolerance
Metrics

- Monitoring metric: observation point on a server in a datacenter
Metrics

- Monitoring metric: observation point on a server in a datacenter
- CPU load, memory, service status
Metrics

- Monitoring metric: observation point on a server in a datacenter
- CPU load, memory, service status
- Reported by agents, processed, and stored
Metrics prediction in monitoring systems

Metrics

- Monitoring metric: observation point on a server in a datacenter
- CPU load, memory, service status
- Reported by agents, processed, and stored
- Computed as time-series
Metrics prediction in monitoring systems

Metrics

- Monitoring metric: observation point on a server in a datacenter
- CPU load, memory, service status
- Reported by agents, processed, and stored
- Computed as time-series
- Associated to thresholds: warning and critical
Metrics prediction in monitoring systems

Metrics behaviour: 6 scenarios

- Critical zone
- Warning zone
- Quick rise
- Slow rise
- Transient rise
- Perplexity point

Value vs. Time graph showing different scenarios of metric behaviour.
Linear regression

- Ability to identify local trends (few hours)
- Fast to compute
- Good candidate to avoid false positives (peaks)
- Library: MLlib (part of Apache Spark)
Metrics prediction in monitoring systems

Linear regression

- Ability to identify local trends (few hours)
- Fast to compute
- Good candidate to avoid false positives (peaks)
- Library: MLlib (part of Apache Spark)
Linear regression

- Ability to identify local trends (few hours)

![Graph showing linear regression with data points and a trend line.](image)
Linear regression

- Ability to identify local trends (few hours)
- Fast to compute
Metrics prediction in monitoring systems

Linear regression

- Ability to identify local trends (few hours)
- Fast to compute
- Good candidate to avoid false positives (peaks)
Linear regression

- Ability to identify local trends (few hours)
- Fast to compute
- Good candidate to avoid false positives (peaks)
- Library: MLlib (part of Apache Spark)
System architecture

Monitoring agents
Metrics prediction in monitoring systems

System architecture

Monitoring agents

Monitoring broker
Metrics prediction in monitoring systems

System architecture

Monitoring agents

Monitoring broker

Cassandra database
Metrics prediction in monitoring systems

System architecture

Monitoring agents

Monitoring broker

Cassandra database

Spark + MLlib
Metrics prediction in monitoring systems

System architecture

Monitoring agents → Monitoring broker

Monitoring broker → Cassandra database

Cassandra database → Alert manager
Cassandra database → GUI
Cassandra database → Spark + MLlib

...
Desired properties

- Scalable: up to a few servers (150 CPU cores) to handle Coservit’s load
Desired properties

- Scalable: up to a few servers (150 CPU cores) to handle Coservit’s load
- End-to-end fault tolerance: metrics can never be lost
Desired properties

- Scalable: up to a few servers (150 CPU cores) to handle Coservit’s load
- End-to-end fault tolerance: metrics can never be lost
- Performances: “fast” to compute metrics predictions
Evaluation

Setup

- Hardware: 4 servers (16–28 cores, 128–256 GB RAM)
- Dataset: Replay on production data recorded at Coservit
- 424,206 metrics, 1.5 billion data points monitored on 25,070 servers
Evaluation

Figure: swap memory
Evaluation

Figure: swap memory
Evaluation

Figure: swap memory
Evaluation

Figure: physical memory
Metrics prediction in monitoring systems

Evaluation

Figure: physical memory
Evaluation

Figure: physical memory
Metrics prediction in monitoring systems

Evaluation

![Graph showing metric prediction over time](image)

**Figure:** disk partition
Evaluation

Figure: disk partition
Evaluation

Figure: disk partition
Some metrics are too volatile and hard to predict

To avoid false positives/negatives, and save resources, they are blacklisted

Root Mean Square Error evaluated weekly

Metrics (temporarily) blacklisted if their RMSE > threshold

58.5% of the metrics have a low RMSE → good predictions
Evaluation

Metric blacklisting

- Some metrics are too volatile and hard to predict
- To avoid false positives/negatives, and save resources, they are blacklisted
Metrics prediction in monitoring systems

Evaluation

Metric blacklisting

- Some metrics are too volatile and hard to predict
- To avoid false positives/negatives, and save resources, they are blacklisted
- Root Mean Square Error evaluated weekly
Metrics prediction in monitoring systems

Evaluation

Metric blacklisting

- Some metrics are too volatile and hard to predict
- To avoid false positives/negatives, and save resources, they are blacklisted
- Root Mean Square Error evaluated weekly
- Metrics (temporarily) blacklisted if their RMSE > threshold
Evaluation

Metric blacklisting

- Some metrics are too volatile and hard to predict
- To avoid false positives/negatives, and save resources, they are blacklisted
- Root Mean Square Error evaluated weekly
- Metrics (temporarily) blacklisted if their RMSE $\geq$ threshold
- 58.5% of the metrics have a low RMSE $\rightarrow$ good predictions
Evaluation

CPU load and memory consumption

Figure: Running on 4 machines and 100 cores for 15 minutes.
Evaluation

Time repartition

Figure: Time repartition for predicting a metric.
Evaluation

Load handling

- End-to-end process for the prediction of 1 metric: 1 second.
Load handling

- End-to-end process for the prediction of 1 metric: 1 second.
- One monitoring server (with 24 cores) can handle the load of 1440 metrics (at worst), which is 85 servers on average.
Evaluation

Load handling: linear scaling

Figure: Amount of metrics handled in 15 minutes.
Load handling: linear scaling

Figure: Amount of metrics handled in 15 minutes.
Evaluation

Load handling: linear scaling

Figure: Amount of metrics handled in 15 minutes.
Positioning

No published work exhibits the same system (end-to-end system for monitoring metrics prediction, storage and blacklisting).

Prediction models

- Hardware failures [CAS12]
Related work

Positioning
No published work exhibits the same system (end-to-end system for monitoring metrics prediction, storage and blacklisting).

Prediction models
- Hardware failures [CAS12]
- Capacity planning (e.g. Microsoft Azure [mic])
Positioning

No published work exhibits the same system (end-to-end system for monitoring metrics prediction, storage and blacklisting).

Prediction models

- Hardware failures [CAS12]
- Capacity planning (e.g. Microsoft Azure [mic])
- Datacenter temperature (e.g. Thermocast [LLL+11])
Positioning

No published work exhibits the same system (end-to-end system for monitoring metrics prediction, storage and blacklisting).

Prediction models

- Hardware failures [CAS12]
- Capacity planning (e.g. Microsoft Azure [mic])
- Datacenter temperature (e.g. Thermocast [LLL^+11])
- Monitoring metrics (e.g. Zabbix [zab] with manual tuning)
Locality data routing

How to design an algorithm to improve locality in distributed streaming engines?
Actors
Collaboration with Vincent Leroy (SLIDE) and Ahmed El-Rheddane (ERODS).
Locality data routing

Distributed streaming engines

Goals

- Real-time message handling
Goals

- Real-time message handling
- Real-time metric calculations
Locality data routing

Distributed streaming engines

Goals

- Real-time message handling
- Real-time metric calculations
- Parallelization
Goals

- Real-time message handling
- Real-time metric calculations
- Parallelization
- Fault-tolerance
Locality data routing

Distributed streaming engines

Apache Storm → topologies.
Apache Storm → topologies.

Figure: Trending hashtags topology.

$S$ sends tweets, operator $A$ extract hashtags, $B$ converts them to lowercase, and $C$ counts the frequency of each hashtag.
Locality data routing

Distributed streaming engines

Apache Storm → topologies.

Figure: Trending hashtags topology.

$S$ sends tweets, operator $A$ extract hashtags, $B$ converts them to lowercase, and $C$ counts the frequency of each hashtag.

Division into tasks → distribution and parallelization made easy.
States are associated to keys

For example, the operator C can keep the list of trending hashtags (values) per location (keys).
Parallelization
To keep a consistent state, same keys must be routed to the same instance.

Figure: Tasks A and B are stateless, C is stateful.
Situation
Let’s have two stateful operators, each with two instances.
Situation
Let's have two stateful operators, each with two instances.

Goal
Minimize the traffic between the machines: $A_1 \rightarrow B_2$ and $A_2 \rightarrow B_1$. By default, $\text{locality} = 1/\text{parallelism}$
Situation
Let’s have two stateful operators, each with two instances.

Goal
Minimize the traffic between the machines: $A_1 \rightarrow B_2$ and $A_2 \rightarrow B_1$.
By default, $\text{locality} = 1/\text{parallelism}$

Constraint
Keep a good load balance between the machines.
Locality data routing

Keys correlation

Dynamically instrument the keys couples and represent them with a bipartite graph.

![Graph partitioning](image_url)
Dynamically instrument the **keys couples** and represent them with a bipartite graph.
Locality data routing

Keys correlation

Dynamically instrument the keys couples and represent them with a bipartite graph.

Routing tables

- **$S$:** Asia $\rightarrow A_1$
  
  Oceania $\rightarrow A_2$

- **$A_1$:**
  
  - #java $\rightarrow B_1$
  
  - #ruby $\rightarrow B_1$
  
  - #python $\rightarrow B_2$

- **$A_2$:**
  
  - #python $\rightarrow B_2$
  
  - #java $\rightarrow B_1$
  
  - #ruby $\rightarrow B_1$
Locality data routing

Keys correlation

Dynamically instrument the keys couples and represent them with a bipartite graph.

Routing tables

- $S$: Asia $\rightarrow A_1$
  Oceania $\rightarrow A_2$
- $A_1$: #java $\rightarrow B_1$
  #ruby $\rightarrow B_1$
  #python $\rightarrow B_2$
- $A_2$: #python $\rightarrow B_2$
  #java $\rightarrow B_1$
  #ruby $\rightarrow B_1$

Graph partitioning $\rightarrow$ optimized routing, favorizing local links.
Locality data routing

Message:
Posted from:

<table>
<thead>
<tr>
<th>S</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Locality data routing

Message: #python doesn’t have braces
Posted from: Oceania

<table>
<thead>
<tr>
<th>S</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Locality data routing

Message: #python doesn’t have braces
Posted from: Oceania

<table>
<thead>
<tr>
<th>Key</th>
<th>Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
</tr>
<tr>
<td>python</td>
<td>B2</td>
</tr>
</tbody>
</table>

Server 1

Server 2
Message: #java is a verbose language
Posted from: Asia
Locality data routing

Message: #java is a verbose language
Posted from: Asia

<table>
<thead>
<tr>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
</tr>
<tr>
<td>Asia</td>
<td>A2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>B2</td>
</tr>
<tr>
<td>java</td>
<td>B1</td>
</tr>
</tbody>
</table>
Locality data routing

Message:  
Posted from:

<table>
<thead>
<tr>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
</tr>
<tr>
<td>Asia</td>
<td>A2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>B2</td>
</tr>
<tr>
<td>java</td>
<td>B1</td>
</tr>
</tbody>
</table>

Reconfiguration is computed and applied
Locality data routing

Server 1

Server 2

Message:
Posted from:

<table>
<thead>
<tr>
<th>S</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
</tr>
<tr>
<td>Asia</td>
<td>A2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>B1</td>
</tr>
<tr>
<td>java</td>
<td>B2</td>
</tr>
</tbody>
</table>

Reconfiguration is computed and applied

Correlation between Oceania/python and Asia/java
Locality data routing

Message: 

posted from: Oceania

```message
#python is pretty cool!
```

Table:

<table>
<thead>
<tr>
<th>S key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oceania</td>
<td>A1</td>
</tr>
<tr>
<td>Asia</td>
<td>A2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A key</th>
<th>route</th>
</tr>
</thead>
<tbody>
<tr>
<td>python</td>
<td>B1</td>
</tr>
<tr>
<td>java</td>
<td>B2</td>
</tr>
</tbody>
</table>
Message: #python is pretty cool!
Posted from: Oceania
Locality data routing

Trends evolve with time
Correlations between keys change frequently.
Trends evolve with time
Correlations between keys change frequently.

Figure: #nevertrump, in March 2016
Locality data routing

Locality decay

- Keys correlations evolve with time.
Locality data routing

Locality decay

- Keys correlations evolve with time.
- Routing tables optimized by examining old data lead to decreased locality.
Locality data routing

Locality decay

- Keys correlations evolve with time.
- Routing tables optimized by examining old data lead to decreased locality.

Reconfiguration

- We re-compute the tables every $N$ minutes.
Locality data routing

Locality decay

- Keys correlations evolve with time.
- Routing tables optimized by examining old data lead to decreased locality.

Reconfiguration

- We re-compute the tables every $N$ minutes.
- Difficulty: keep the state consistent.
Solution: **online reconfiguration protocol**

- update the routing tables in a live system
- without losing any message and state
Locality data routing

Reconfiguration protocol

<table>
<thead>
<tr>
<th>M</th>
<th>A₁</th>
<th>A₂</th>
<th>B₁</th>
<th>B₂</th>
</tr>
</thead>
</table>

- 1. Get statistics
- 2. Send statistics
- 3. Partition graph, compute routing tables
- 4. Send reconfiguration
- 5. Send ACK
- 6. Propagate
- 7. Transfer key states
- 8. Propagate to next operator
Locality data routing

Reconfiguration protocol

\[ \begin{align*}
&\text{1. Get statistics} \\
\end{align*} \]}
Locality data routing

Reconfiguration protocol

1. Get statistics
2. Send statistics
Locality data routing

Reconfiguration protocol

1. Get statistics
2. Send statistics

Partition graph, compute routing tables
Locality data routing

Reconfiguration protocol

1. Get statistics
2. Send statistics

Partition graph, compute routing tables
3. Send reconfiguration
Locality data routing

Reconfiguration protocol

1. Get statistics
2. Send statistics
3. Send reconfiguration
4. Send ACK

Partition graph, compute routing tables
Locality data routing

Reconfiguration protocol

1. Get statistics
2. Send statistics

*Partition graph, compute routing tables*

3. Send reconfiguration
4. Send ACK
5. Propagate

compute routing tables
Locality data routing

Reconfiguration protocol

1. Get statistics
2. Send statistics

*Partition graph, compute routing tables*

3. Send reconfiguration
4. Send ACK
5. Propagate
6. Transfer key states
Locality data routing

Reconfiguration protocol

1. Get statistics
2. Send statistics

*Partition graph, compute routing tables*

3. Send reconfiguration
4. Send ACK

5. Propagate
6. Transfer key states

*Propagate to next operator*
Evaluation

Datasets

- From Flickr and Twitter
- Fields: location (country or place), hashtag
- Size: 173M records (Flickr), 100M (Twitter)
Evaluation

Datasets

- From Flickr and Twitter
- Fields: location (country or place), hashtag
- Size: 173M records (Flickr), 100M (Twitter)

Setup

- 8× 128 GB RAM, 20 cores.
- Computation of aggregated statistics (stateful workers).
- Parallelism (2..6), network speed (1Gb/s | 10Gb/s), message size (0..20kB).
Great speed-up when network is the bottleneck.
Locality data routing

Evaluation

Great speed-up when network is the bottleneck.

Highly dependent on message size.
Locality data routing

Evaluation – Flickr

Throughput (Ktuples/s) on 10Gb/s network, parallelism 6

(a) message size=4kB
(b) message size=8kB
Locality data routing

Evaluation – Flickr

Throughput (Ktuples/s) on 1Gb/s network, parallelism 6

(a) message size=4kB

(b) message size=8kB
Average throughput with 1Gb/s network, 4kB message size

**Figure:** Average throughput, measured after the first reconfiguration.
Locality data routing

Evaluation – Flickr

Locality, with parallelism 6

![Graph showing locality with weeks on the x-axis and percentage on the y-axis. The graph includes a green line marked "hash-based." ]
Locality data routing

Evaluation – Flickr

Locality, with parallelism 6

![Graph showing locality over weeks]

- **Weeks**: 0 to 25
- **Locality**: 0% to 60%

Legend:
- **Hash-based**
- **Offline**
Locality data routing

Evaluation – Flickr

Locality, with parallelism 6

![Graph](image-url)

- **Locality**
  - **weeks**
  - **hash-based**
  - **offline**
  - **online**
Locality data routing

Evaluation – Flickr

Locality when changing the number of collected key correlations

![Graph showing locality vs. number of edges for different values of collected key correlations. Each line represents a different number of collected key correlations, and the y-axis shows locality ranging from 0% to 80%, while the x-axis shows edges on a logarithmic scale ranging from $10^1$ to $10^7$.]

- Line 2 (green square) represents locality for 2 collected key correlations.
- Line 3 (red circle) represents locality for 3 collected key correlations.
- Line 4 (blue triangle) represents locality for 4 collected key correlations.
- Line 5 (orange triangle) represents locality for 5 collected key correlations.
- Line 6 (pink triangle) represents locality for 6 collected key correlations.
Locality data routing

Related work

Scheduling: placement of operators on servers

- Using the topology [ABQ13]
- Using observed communication patterns [ABQ13]
- Using observed and/or estimated CPU and memory patterns [FB15, PHH+15]
Related work

Scheduling: placement of operators on servers

- Using the topology [ABQ13]
- Using observed communication patterns [ABQ13]
- Using observed and/or estimated CPU and memory patterns [FB15, PHH+15]

Load balancing: limit impact of data skew

- Partial key grouping [NMG+15]
- Special routing for frequent keys [RQA+15]
Related work

**Scheduling:** placement of operators on servers

- Using the topology [ABQ13]
- Using observed communication patterns [ABQ13]
- Using observed and/or estimated CPU and memory patterns [FB15, PHH+15]

**Load balancing:** limit impact of data skew

- Partial key grouping [NMG+15]
- Special routing for frequent keys [RQA+15]

**Co-location of correlated keys**

- Databases partitions [CJZM10], social networks [BJJL13]
How to design a framework to compose data processing algorithms in a descriptive fashion, while reasoning on high level abstractions?
Design goals

▶ A data processing abstraction
Design goals

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

- 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

- 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

- 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

- 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
read file
/etc/passwd
read file
/etc/passwd

count
read file
/etc/passwd

filter
contains: 'root'

count
"""Counts system users.
"""

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

if __name__ == '__main__':
    print(main())
"""Counts system users. """

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

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

$ wc -l /etc/passwd
"""Counts system users."
"""

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

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

$ wc -l /etc/passwd
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
λ-blocks

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
@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

\[\lambda\text{-blocks}\]

count_pb

\[
\begin{align*}
\text{filter} \\
\text{count}
\end{align*}
\]
---
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
Architecture

Graph engine

API, CLI
Architecture

- Blocks registry
- Graph engine
- API, CLI
- Block libraries
Architecture

λ-blocks

- Blocks registry
- Graph plugins
- Graph engine
- API, CLI
- Block libraries
Architecture

- Blocks registry
- Graph plugins
- Graph engine
- Topology
- Block libraries
- API, CLI

Block libraries: 

\[ \lambda \text{-blocks} \]
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
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()

@after_block_execution
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']
@after_graph_execution

def show_times(results):
    longest_first = sorted(by_block, reverse=True)
    for blockname in longest_first:
        print('{}		{}'.format(blockname,
                             by_block[blockname]['duration']))
Graph manipulation example: instrumentation

<table>
<thead>
<tr>
<th>block</th>
<th>duration (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>read http</td>
<td>818</td>
</tr>
<tr>
<td>write lines</td>
<td>54</td>
</tr>
<tr>
<td>grep</td>
<td>49</td>
</tr>
<tr>
<td>split</td>
<td>20</td>
</tr>
</tbody>
</table>
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)
Performances

**Figure:** Wordcount over https: Twitter feed.
Performances

Figure: Wordcount over disk: Wikipedia dataset.
Performances

Figure: PageRank on Wikipedia hyperlinks with Spark.
Maximum overhead measured per topology: 50 ms
λ-blocks

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 \textsuperscript{[PVG+11]}, Spark \textsuperscript{[The17a]}, Orange framework \textsuperscript{[DCE+13]}
- Real-time: Apache Beam \textsuperscript{[apa]}, StreamPipes \textsuperscript{[RKHS15]}

Blocks programming

- Recognition over recall, immediate feedback \textsuperscript{[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]
Context

Computer systems to process large quantities of data.
Conclusion

Context

Computer systems to process large quantities of data.

Problems: how to design... 

- An industrial system to handle monitoring data and make predictions about future failures?
- An algorithm to improve locality in distributed streaming engines?
- A framework to compose data processing algorithms in a descriptive fashion, while reasoning on high level abstractions?
Conclusion

Contributions

<table>
<thead>
<tr>
<th>Metrics prediction</th>
<th>Locality routing</th>
<th>λ-blocks</th>
</tr>
</thead>
</table>
## Conclusion

### Contributions

<table>
<thead>
<tr>
<th>What it is</th>
<th>Metrics prediction</th>
<th>Locality routing</th>
<th>λ-blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industrial system</td>
<td>Online routing library</td>
<td>Data processing abstraction</td>
<td></td>
</tr>
</tbody>
</table>
## Conclusion

### Contributions

<table>
<thead>
<tr>
<th>What it is</th>
<th>Metrics prediction</th>
<th>Locality routing</th>
<th>λ-blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industrial system</td>
<td>Online routing library</td>
<td>Data processing abstraction</td>
<td></td>
</tr>
<tr>
<td>Layer</td>
<td>End-to-end</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
## Conclusion

## Contributions

<table>
<thead>
<tr>
<th>Metrics prediction</th>
<th>Locality routing</th>
<th>λ-blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What it is</strong></td>
<td>Industrial system</td>
<td>Online routing library</td>
</tr>
<tr>
<td><strong>Layer</strong></td>
<td>End-to-end</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Improves</strong></td>
<td>Uptimes</td>
<td>Throughput</td>
</tr>
</tbody>
</table>
Conclusion

Future work

Metrics prediction in monitoring systems

- Predictions on long-term global trends
- Ticketing mechanism
Conclusion

Future work

Metrics prediction in monitoring systems

- Predictions on long-term global trends
- Ticketing mechanism

Locality data routing

- Replace binary locality/non-locality with distance
- Smarter way to determine when to reschedule
- Extend to more complex topologies
Conclusion

Future work

\( \lambda \)-blocks

- Explore more graph manipulation abstractions (complexity analysis, serialization, verification…)
- Streaming and online operations
- Tight integration with clusters (data storage, caches, etc)
Thanks! Questions?
Using a Spark cluster

- Spark master
- slave-1
- slave-2
- slave-3

- Block calling Spark
- Normal block
$H(B) = h(B.name, \text{ block name (not instance name)})$

$B.args,$ \text{ list of (name, value) tuples}$

$B.inputs$ \text{ list of (name, H(block), connector) tuples}$
Figure: Wordcount program running under different setups. 
(1) Startup (modules import, etc); (2) Blocks registry creation, block modules import; (3) Plugin import; (4) YAML parsing and graph creation; (5) Graph checks; (6) Graph execution.
## Database schema

### metrics
- metric_id: uuid
- metric_name: text
- group_id: uuid

### measurements
- metric_id: uuid
- timestamp: int
- warn: text
- crit: text
- max: double
- min: double
- value: double
- metric_name: text
- metric_unit: text

### predictions
- metric_id: uuid
- timestamp: int
- predicted_values: list
Images credits

- *Data Center operators verifying network cable integrity*, CC-BY-SA,
  https://commons.wikimedia.org/wiki/File:Dc_cabling_50.jpg

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


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

Lei Li, Chieh-Jan Mike Liang, Jie Liu, Suman Nath, Andreas Terzis, and Christos Faloutsos.
Thermocast: A cyber-physical forecasting model for datacenters.
In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’11, pages 1370–1378, New York, NY, USA, 2011. ACM.
Microsoft cloud azure.
https://docs.microsoft.com/en-us/azure/
machine-learning/
machine-learning-algorithm-choice.

Muhammad Anis Uddin Nasir, Gianmarco De Francisci Morales, David García-Soriano, Nicolas Kourtellis, and Marco Serafini.
The power of both choices: Practical load balancing for distributed stream processing engines.
In 31st IEEE International Conference on Data Engineering, ICDE, pages 137–148, 2015.
Boyang Peng, Mohammad Hosseini, Zhihao Hong, Reza Farivar, and Roy Campbell.

R-storm: Resource-aware scheduling in storm.


Scikit-learn: Machine learning in Python.
Dominik Riemer, Florian Kauflersch, Robin Hutmacher, and Ljiljana Stojanovic. 
Streampipes: solving the challenge with semantic stream processing pipelines. 

Nicoló Rivetti, Leonardo Querzoni, Emmanuelle Anceaume, Yann Busnel, and Bruno Sericola. 
Efficient key grouping for near-optimal load balancing in stream processing systems. 
The Apache Spark developers.
ML Pipelines.

The Apache Storm developers.
Flux.

YelpArchive.
Pyleus.
Zabbix prediction triggers.
https://www.zabbix.com/documentation/3.0/manual/config/triggers/prediction.