Migration of a realtime stats product from Storm to Flink

Patrick Gunia
Engineering Manager @ ResearchGate

ResearchGate

ResearchGate is built for scientists.

The social network gives scientists new tools to connect, collaborate, and keep up with the research that matters most to them.

Photography-based taxonomy is inadequate, unnecessary, and potentially harmful for biological sciences

FULL TEXT

Article in Zootaxa 4196(3):435-445 · November 2016

DOI: 10.11646/zootaxa.4196.3.9

1st Luis Miguel Pires Ceríaco
21.05 · Villanova University

2nd Eliécer E. Gutiérrez
37.07 · Universidade Federal de Santa ...

3rd Alain Dubois
37.97 · Muséum National ...

+ 486

Last George Zug
32.31 · Smithsonian Institution

Show more authors

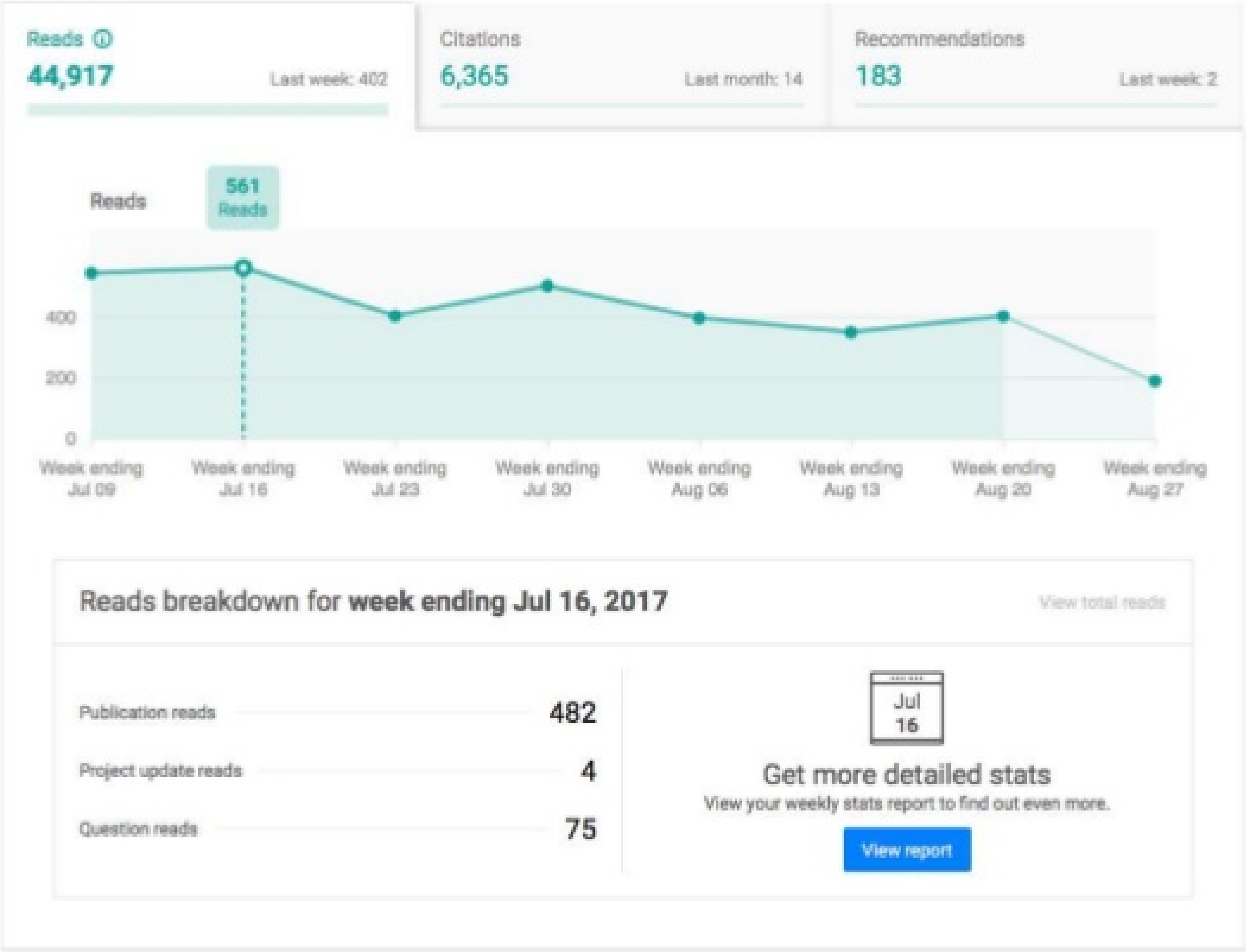
Abstract

The question whether taxonomic descriptions naming new animal species without type specimen(s) deposited in collections should be accepted for publication by scientific journals and allowed by the Code has already been discussed in Zootaxa (Dubois & Nemésio 2007; Donegan 2008, 2009; Nemésio 2009a–b; Dubois 2009; Gentile & Snell 2009; Minelli 2009; Cianferoni & Bartolozzi 2016; Amorim et al. 2016). This question was again raised...

Linked Data

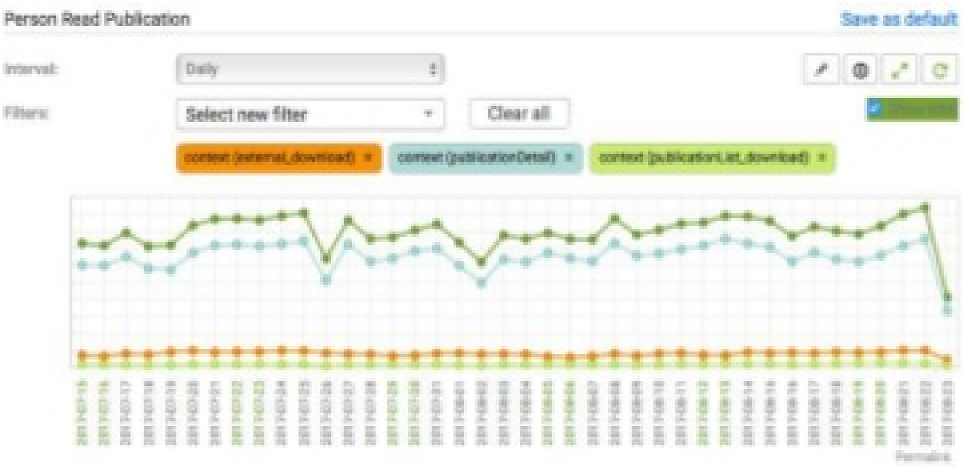
Photography-based taxonomy is inadequat...
Data · Nov 2016

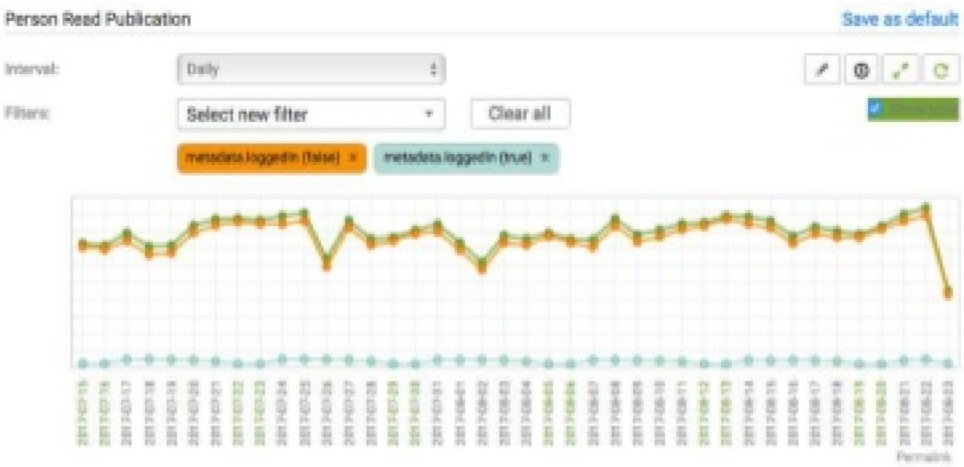
What to count –Reads

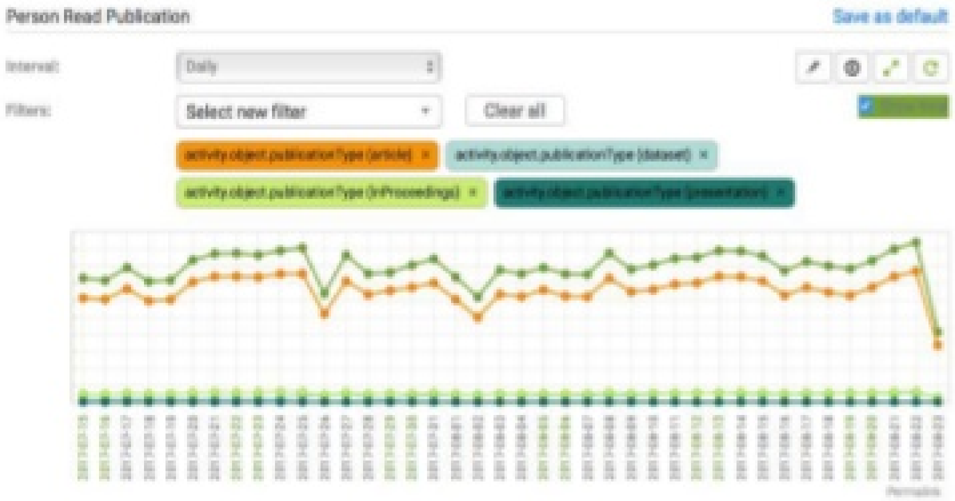


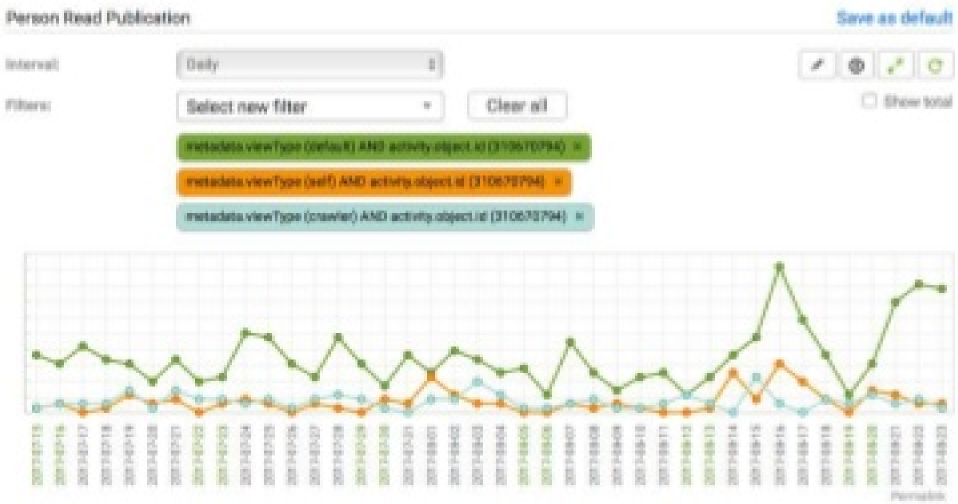
ResearchGate

What to count –Reads









ResearchGate

What to count – Requirements

1. **Correctness**

2. (Near) **realtime**

3. **Adjustable**

ResearchGate

How we did it in the past…

ResearchGate

Never change a running system….

…so why would we?

Drawbacks of the original solution

Counting Scheme

	Event	Event		Event		Event	Event	Event	
Counter A	+1	+1				+1			
Counter B		+1		+1			+1	+1	
Counter C	+1					+1	+1		

time

Drawbacks of the original solution

Performance / Efficiency

- Single increments are **inefficient** and cause a **high write load** on the database
- Roughly **~1,000 million counters** are incremented per day
- Load **grows linearly with platform activity** (e.g. x increment operations per publication read)

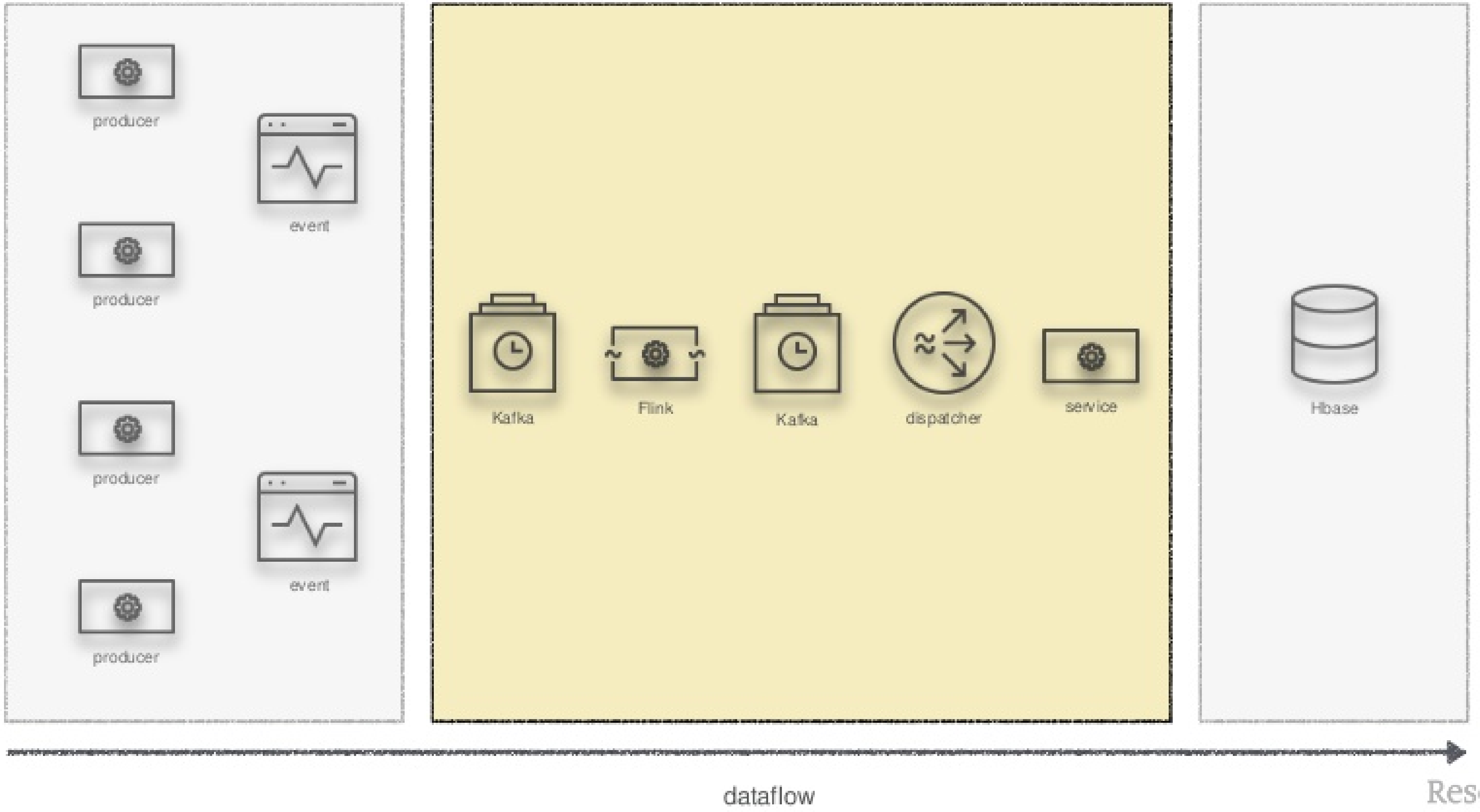
Drawbacks of the original solution

Operational difficulties

- **At-least-once semantics** in combination with the simple increment logic can cause **multiple increments** per event on the same counter
- To ensure **correctness**, additional components are needed (e.g. consistency checks and fix jobs)

From stateless to stateful counting

How we do it now…



As this is Flink Forward…

…let’s focus on the Flink part of the story

What we want to achieve

1. Migrate the Storm implementation to Flink
2. Improve efficiency and performance
3. Improve ease of operation

From stateless to stateful counting

Baseline implementation

Simple Counting

Migrate the Storm implementation to Flink

1. Read input event from Kafka
2. FlatMap input event into (multiple) entities
3. Output results (single counter increments) to Kafka sink
4. Perform DB operation based on Kafka messages

Simple Counting

Benefit

Lines of Code: Reduced lines of code by ~70% by using the High-Level language abstraction offered by Flink APIs

What we want to achieve

1. Migrate the Storm implementation to Flink ✓
2. Improve efficiency and performance
3. Improve ease of operation

From stateless to stateful counting

Challenge 1: Performance / Efficiency

"Statefuller" Counting

Doing it the more Flink(y) way…

1. Read input event from Kafka
2. FlatMap input event into (multiple) entities
3. **Key stream based on entity key**
4. **Use time window to pre-aggregate**
5. Output **aggregates** to Kafka sink
6. Perform DB operations based on Kafka messages

```
final DataStream<CounterDTO> output = source
    // create counters based on input events
    .flatMap(new MapEventToCounterKey(mapConfig))
    .name(MapEventToCounterKey.class.getSimpleName())

    // key the stream by minute aligned counter keys
    .keyBy(new CounterMinuteKeySelector())

    // and collect them for a configured window
   .timeWindow(Time.milliseconds(config.getTimeWindowInMs()))
    .fold(null, new MinuteWindowFold())
    .name(MinuteWindowFold.class.getSimpleName())
```

"Statefuller" Counting

Counting Scheme

	Event	Event				Event	Event	Event	
Counter A	+1	+1				+1			
Counter B		+1					+1	+1	

time

"Statefuller" Counting

Counting Scheme

	Event	Event				Event	Event	Event	
Counter A	+1	+1				+1			
Counter B		+1					+1	+1	

Using windows to pre-aggregate increments

time

"Statefuller" Counting

Counting Scheme

	Event	Event				Event	Event	Event	
Counter A	+1	+1				+1			
Counter B		+1					+1	+1	

Using windows to pre-aggregate increments

	Event	Event				Event	Event	Event	
Counter A		+2				+1			
Counter B		+1						+2	

time

"Statefuller" Counting

Benefits

1. **Easy** to implement (KeySelector, TimeWindow, Fold)

2. **Small resource footprint**: two yarn containers with 4GB of memory

3. Window-based pre-aggregation **reduced the write load on the database** from ~1,000 million increments per day to ~200 million (~80%)

What we want to achieve

1. Migrate the Storm implementation to Flink ✓
2. Improve efficiency and performance ✓
3. Improve ease of operation

From stateless to stateful counting

Challenge 2: Improve ease of operation

Idempotent counting

Counting Scheme

	Event	Event				Event	Event	Event	
Counter A		+2				+1			
Counter B		+1						+2	

time

Idempotent counting

Counting Scheme

	Event	Event				Event	Event	Event	
Counter A		+2				+1			
Counter B		+1						+2	

Replace increments with PUT operations

time

Idempotent counting

Counting Scheme

	Event	Event				Event	Event	Event	
Counter A		+2				+1			
Counter B		+1						+2	

Replace increments with PUT operations

	Event	Event				Event	Event	Event	
Counter A		40				41			
Counter B		12						14	

time

Idempotent counting

Scheme

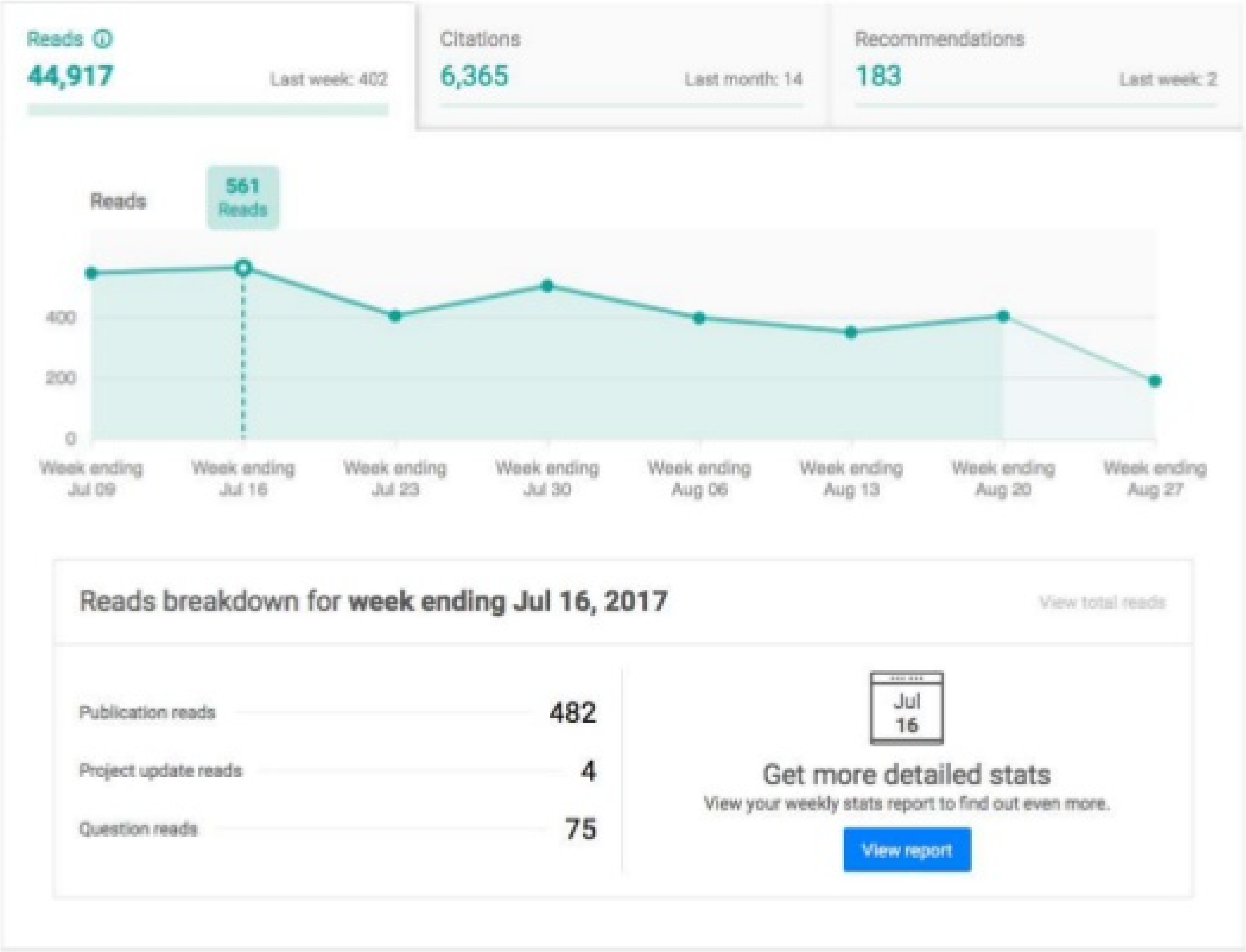
Windows

1st: Pre-aggregates increments

2nd: Accumulates the state for counters with day granularity

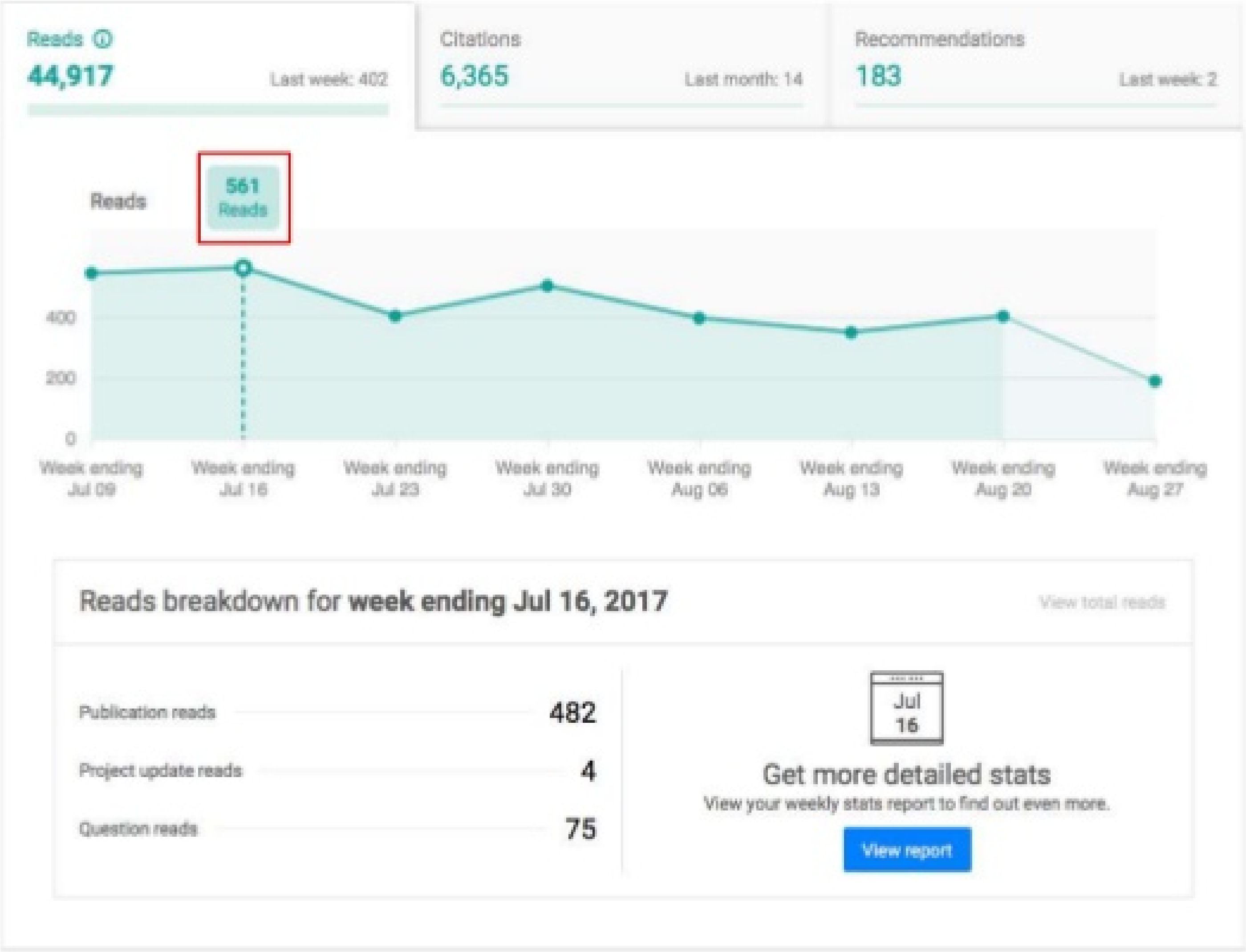
```
final DataStream<CounterDTO> output = source
    // create counters based on input events
    .flatMap(new MapEventToCounterKey(mapConfig))
    .name(MapEventToCounterKey.class.getSimpleName())

    // key the stream by minute aligned counter keys
    .keyBy(new CounterMinuteKeySelector())

    // and collect them for a configured window
    .timeWindow(Time.milliseconds(config.getTimeWindowInMs()))
    .fold(null, new MinuteWindowFold())
    .name(MinuteWindowFold.class.getSimpleName())

    // key the stream by day aligned counter keys
    .keyBy(new CounterDayKeySelector())
    .timeWindow(Time.minutes(config.getDayWindowTimeInMinutes()))

    // trigger on each element and purge the window
    .trigger(new OnElementTrigger())
    .fold(null, new DayWindowFold())
    .name(DayWindowFold.class.getSimpleName());
```

Back to our graphs...



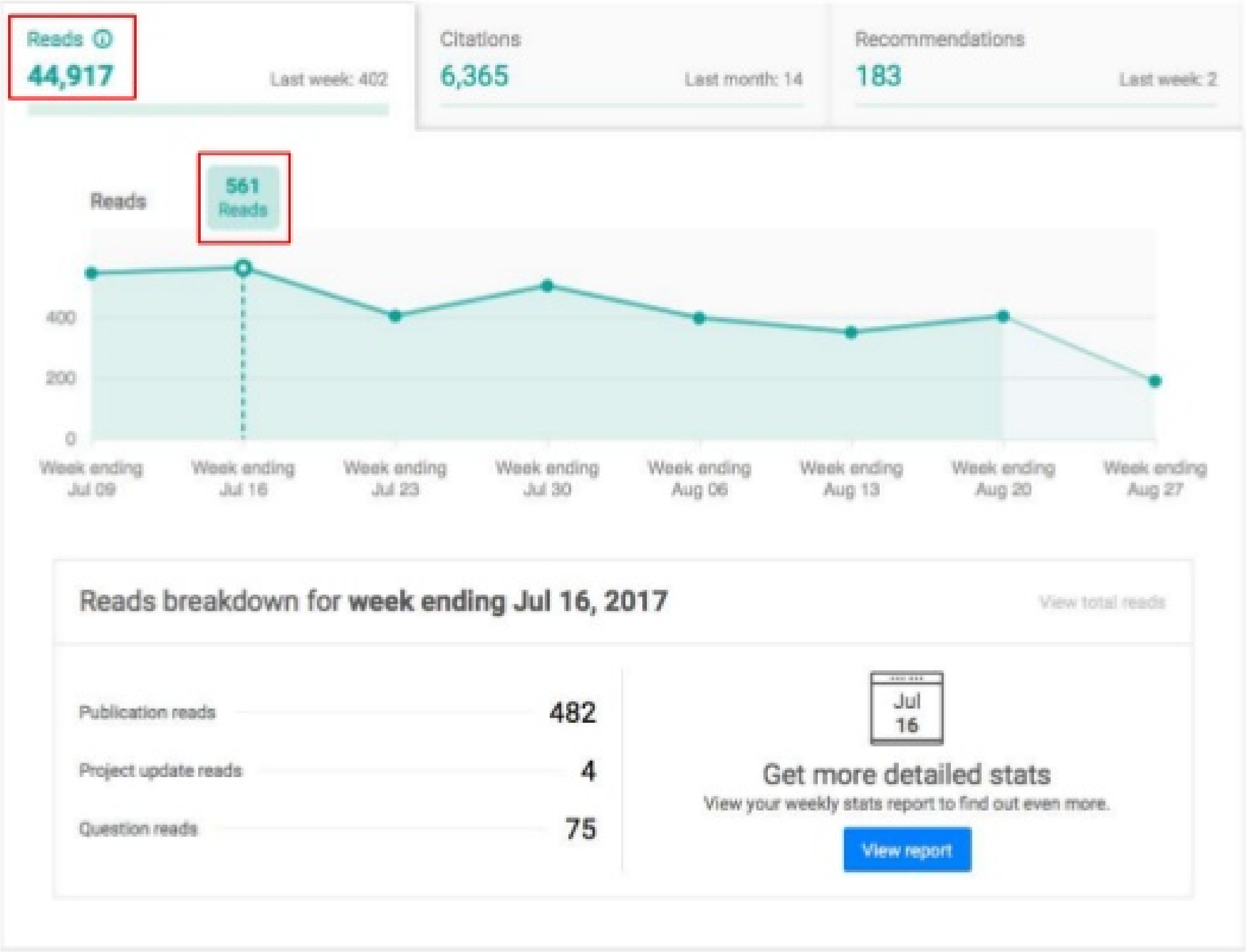
ResearchGate

Back to our graphs...



ResearchGate

Back to our graphs...



ResearchGate

All time counters

- Windows for counters with day granularity are **finite** and **can be closed**
- this enables Put operations instead of increments and thus eases operation and **increases correctness and trust**

But how to handle updates of counters, that are never „closed“?

All time counters – 1st idea

1. Key the stream based on a key that identifies the „all-time“ counter
2. **Update** the **counter state** for every message in the keyed stream
3. Output the **current state** and perform a **Put** operation on the database

All time counters – 1st idea

Problems

- Creating the **initial state** for all existing counters is **not trivial**
- The key space is **unlimited**, thus the state will grow **infinitely**

Thus we‘ve come up with something else...

All time counters – 2nd idea

- Day counter updates can be used to also update the all-time counter idempotently
- Simplified:
  1. **Remember** what you did before
  2. **Revert** the previous operation
  3. **Apply** the update

All time counters – 2nd idea

Update Scheme

	All-time	Current Day
Counter	245	5

All time counters – 2nd idea

Update Scheme

	All-time	Current Day
Counter	245	5

Incoming Update for current day, new value: 7

All time counters – 2nd idea

Update Scheme

	All-time	Current Day
Counter	245	5

Incoming Update for current day, new value: 7

Step 1: Revert the previous operation 245 – 5 = 240

Step 2: Apply the update 240 + 7 = 247

All time counters – 2nd idea

Update Scheme

	All-time	Current Day
Counter	245	5

Incoming Update for current day, new value: 7

Step 1:	Revert the previous operation	245 – 5 = 240
Step 2:	Apply the update	240 + 7 = 247

Put the row back into the database

All time counters – 2nd idea

Update Scheme

	All-time	Current Day
Counter	245	5

Incoming Update for current day, new value: 7

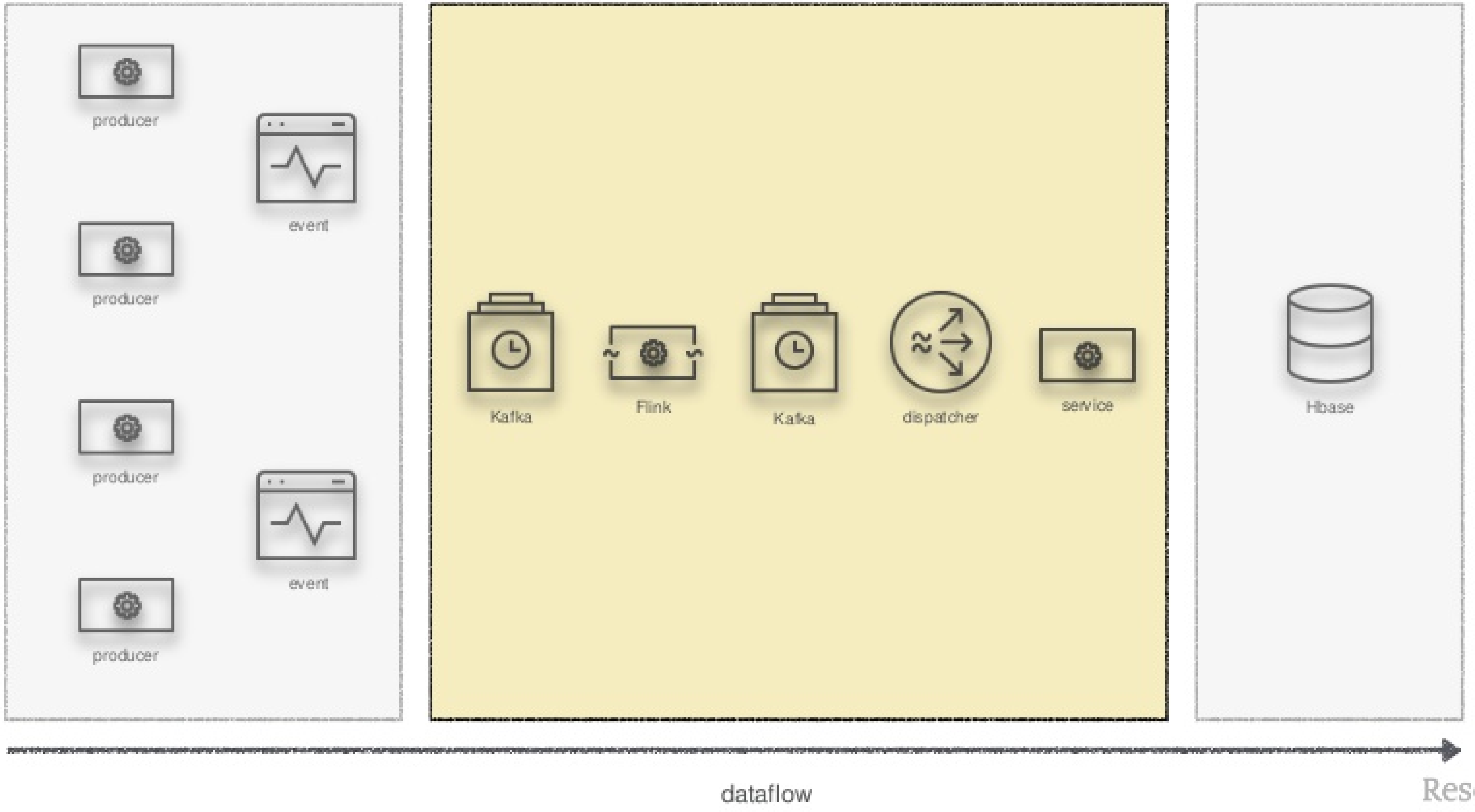
Step 1: Revert the previous operation 245 – 5 = 240

Step 2: Apply the update 240 + 7 = 247

Put the row back into the database

	All-time	Current Day
Counter	247	7

Stats processing pipeline



ResearchGate

What we want to achieve

1. Migrate the Storm implementation to Flink ✓
2. Improve efficiency and performance ✓
3. Improve ease of operation ✓

How to integrate stream and batch processing?

Batch World

There are a couple of things that might

- be **unknown** at processing time (e.g. bad behaving crawlers)
- **change** after the fact (e.g. business requirements)

Batch World

- Switching to Flink enabled us to **re-use parts of the implemented business logic** in a nightly batch job
- new counters can easily be added and existing ones modified
- the described architecture allows us to **re-use code** and **building blocks** from our infrastructure

Thank you!

Questions?

patrick.gunia@researchgate.net
Twitter: @patgunia
LinkedIn: patrickgunia

ResearchGate