Three steps is all you need
fast, accurate, automatic scaling decisions
for distributed streaming dataflows

Vasiliki Kalavri†, John Liagouris†, Moritz Hoffmann†,
Desislava Dimitrova†, Matthew Forshaw‡†, Timothy Roscoe†

†Systems Group, Department of Computer Science, ETH Zürich, firstname.lastname@inf.ethz.ch
‡ ††Newcastle University, firstname.lastname@newcastle.ac.uk
Any streaming job will inevitably become over- or under-provisioned in the future
Any streaming job will inevitably become over- or under-provisioned in the future.
Any streaming job will inevitably become over- or under-provisioned in the future.
Any streaming job will inevitably become over- or under-provisioned in the future.
CONFIGURING PARALLELISM FOR A STREAMING JOB

parallel dataflow

input streams

output stream
CONFIGURING PARALLELISM FOR A STREAMING JOB

1. monitor event rates
2. configure parallelism
3. deploy and test performance

until the target throughput is met
THE SCALING PROBLEM

Given a logical dataflow with sources $S_1, S_2, \ldots S_n$ and rates $\lambda_1, \lambda_2, \ldots \lambda_n$ identify the minimum parallelism $\pi_i$ per operator $i$, such that the physical dataflow can sustain all source rates.
AUTOMATIC SCALING OVERVIEW
AUTOMATIC SCALING OVERVIEW

detect symptoms
metrics
scaling controller
policy
scaling action
AUTOMATIC SCALING OVERVIEW

- **detect symptoms**
- **metrics**
- **scaling controller**
- **policy**
- **decide whether to scale**
- **scaling action**
AUTOMATIC SCALING OVERVIEW

detect symptoms

metrics

scaling controller

policy

decide whether to scale

decide how much to scale

scaling action
Existing approaches

Borealis
StreamCloud
Seep
IBM Streams
Spark Streaming
Google Dataflow
Dhalion

...
Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- ...

**metrics**
- CPU utilization
- backlog, tuples/s
- backpressure signal

**policy**

**scaling action**
Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- ...

**metrics**

- CPU utilization
- backlog, tuples/s
- backpressure signal

**policy**

- threshold and rule-based
  - if $CPU > 80\% \Rightarrow scale$

**scaling action**
Existing approaches

| Borealis   | CPU utilization backlog, tuples/s backlogpressure signal |
| StreamCloud |                                                        |
| Seep       |                                                        |
| IBM Streams|                                                        |
| Spark Streaming |                                    |
| Google Dataflow |                                           |
| Dhalion    |                                                        |

metrics

policy

threshold and rule-based
if CPU > 80% => scale

scaling action

small changes, one operator at a time
## Existing approaches

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Policy</th>
<th>Scaling action</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization, backlog, tuples/s, backpressure signal</td>
<td>if CPU &gt; 80% =&gt; scale</td>
<td>small changes, one operator at a time</td>
</tr>
</tbody>
</table>

### Problematic due to interference, multitenancy
Existing approaches

Borealis
StreamCloud
Seep
IBM Streams
Spark Streaming
Google Dataflow
Dhalion
...

metrics
CPU utilization
backlog, tuples/s
backpressure signal

policy
threshold and rule-based
if CPU > 80% => scale

scaling action
small changes, one operator at a time

problematic due to interference, multitenancy
sensitive to noise, manual, hard to tune
Existing approaches

Borealis
StreamCloud
Seep
IBM Streams
Spark Streaming
Google Dataflow
Dhalion
...

metrics
CPU utilization backlog, tuples/s backpressure signal

problematic due to interference, multitenancy

policy
threshold and rule-based
if CPU > 80% => scale

sensitive to noise, manual, hard to tune

scaling action
small changes, one operator at a time

non-predictive, speculative steps
Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- ...

**metrics**
- CPU utilization
- backlog, tuples/s
- backpressure signal

**policy**
- threshold and rule-based
- if $CPU > 80\%$ => scale

**scaling action**
- small changes, one operator at a time

- problematic due to interference, multitenancy
- sensitive to noise, manual, hard to tune
- non-predictive, speculative steps

X oscillations
Existing approaches

- Borealis
- StreamCloud
- Seep
- IBM Streams
- Spark Streaming
- Google Dataflow
- Dhalion
- ...
Existing approaches

**Borealis**

**StreamCloud**

**Seep**

**IBM Streams**

**Spark Streaming**

**Google Dataflow**

**Dhalion**

…

**metrics**

- CPU utilization
- backlog, tuples/s
- backpressure signal

**policy**

- threshold and rule-based
- if CPU > 80% => scale

**scaling action**

- small changes, one operator at a time

- problematic due to interference, multitenancy

- sensitive to noise, manual, hard to tune

- non-predictive, speculative steps

- oscillations

- temporary over- and under-provisioning

- slow convergence
effect of Dhalion’s scaling actions in an initially under-provisioned wordcount dataflow
effect of Dhalion’s scaling actions
in an initially under-provisioned wordcount dataflow
metrics
externally observed

policy
threshold-based

scaling action
non-predictive, single-operator
OUR APPROACH: DS2

- **metrics**: externally observed
- **policy**: threshold-based
- **scaling action**: non-predictive, single-operator

- true rates through instrumentation
- dataflow dependency model
- predictive, dataflow-wide actions
OUR APPROACH: DS2

- **metrics**: externally observed
- **policy**: threshold-based
- **scaling action**: non-predictive, single-operator

- true rates through instrumentation
- dataflow dependency model
- predictive, dataflow-wide actions

- no oscillations
- true rates as bounds to avoid over/under-shoot
- fast convergence

OUR APPROACH: DS2
backpressure
target: 40 rec/s
Which operator is the bottleneck?

What if we scale $o_1 \times 4$?

How much to scale $o_2$?
backpressure
target: 40 rec/s

observed view
The diagram illustrates the backpressure target: 40 rec/s and the observed view. The time line runs from 1 to 4 seconds, with markings at 10 rec/s and 100 rec/s. The nodes src, o1, and o2 are shown with the status of busy and waiting. The diagram also indicates backpressure due to target and observed view.
**O₁ is the bottleneck**

The diagram illustrates a network flow with three nodes: `src`, `O₁`, and `O₂`. The flow rates are shown as 10 rec/s into `O₁` and 100 rec/s between `O₁` and `O₂`. The bottleneck is identified as `O₁` due to the mismatch between its input rate of 10 rec/s and the output rate of 100 rec/s, leading to backpressure.

**Instrumentation**

The observed view indicates that the system is experiencing backpressure and is operating at its target rate of 40 rec/s. The observed rec/s are 10 rec/s into `O₁` and 100 rec/s between `O₁` and `O₂`, with a 0.5s delay between them, highlighting the inefficiency and the need for optimization.

**DS₂ View**

The diagram also includes a timeline from 1 to 4 seconds, with the time labeled in intervals of 2 seconds (1, 2, 3, 4). The busy and waiting states are denoted by solid and dotted lines, respectively.
DS2 view

- **src**
  - : busy
  - : waiting

- **O₁** is the bottleneck
- 2 **O₂** instances can keep up with the rate of 4 **O₁** instances

- Backpressure target: 40 rec/s
- True rate = 200 rec/s

Instrumentation

```
<table>
<thead>
<tr>
<th>Time (s)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O₁</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>O₂</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
```

O₁ instances can keep up with the rate of 4 O₁ instances.
THE DS2 MODEL
THE DS2 MODEL

**Useful time:** The time spent by an operator instance in deserialization, processing, and serialization activities.
THE DS2 MODEL

**Useful time**: The time spent by an operator instance in deserialization, processing, and serialization activities.

**True processing (resp. output) rate**: The number of records an operator instance can process (resp. output) per unit of useful time.
THE DS2 MODEL

Useful time: The time spent by an operator instance in deserialization, processing, and serialization activities.

True processing (resp. output) rate: The number of records an operator instance can process (resp. output) per unit of useful time.

Optimal parallelism for $o_i$: \[
\text{aggregated true output rate of upstream ops} \quad \frac{\text{average true processing rate of } o_i}{\text{per unit of useful time}}
\]
CONVERGENCE STEPS

- **no overshoot** when scaling up: ideal rate is an *upper* bound
- **no undershoot** when scaling down: ideal rate is a *lower* bound

if the actual scaling is linear, convergence takes **one** step
CONVERGENCE STEPS

- **no overshoot** when scaling up: ideal rate is an *upper* bound
- **no undershoot** when scaling down: ideal rate is a *lower* bound

If the actual scaling is linear, convergence takes **one** step.

![Diagram showing target, initial rate, and parallelism axes with a point labeled p0 and a dashed line representing prediction.](image-url)
CONVERGENCE STEPS

- no overshoot when scaling up: ideal rate is an upper bound
- no undershoot when scaling down: ideal rate is a lower bound

If the actual scaling is linear, convergence takes one step

\[ p_0 \to p_1 \]

\[ \text{target} \]

\[ \text{initial rate} \]

\[ \text{parallelism} \]
CONVERGENCE STEPS

- **no overshoot** when scaling up: ideal rate is an *upper* bound
- **no undershoot** when scaling down: ideal rate is a *lower* bound

if the actual scaling is linear, convergence takes **one** step

If the actual scaling is linear, convergence takes **one** step.

If the actual scaling is linear, convergence takes **one** step.
In practice, rates are commonly **sub-linear** due to other overheads (e.g. worker coordination).
In practice, rates are commonly **sub-linear** due to other overheads (e.g. worker coordination).
In practice, rates are commonly **sub-linear** due to other overheads (e.g. worker coordination).

when the actual scaling is sub-linear, convergence takes **more than one** steps
In practice, rates are commonly sub-linear due to other overheads (e.g. worker coordination).

In our experiments, DS2 took up to three steps to converge for complex queries.
DS2 operates online in a reactive setting

Apache Flink

Timely dataflow

Instrumented stream processor

Scaling Manager

Scaling Policy

Metrics Repository

report metrics

monitor

pull metrics

invoke

decision

re-scale job
EVALUATION
DS2 VS. DHALION ON HERON

wordcount

Target rate: 16.700 rec/s
DS2 converges in a single step for both operators.
DS2 converges in a single step for both operators.

DS2 converges in 60s, i.e. as soon as it receives the Heron metrics.

Target rate: 16.700 rec/s
DS2 converges in a **single step** for both operators.

DS2 converges in **60s**, i.e. as soon as it receives the Heron metrics.

Dhalion scales one operator at a time, resulting to a total of **six steps**.

Target rate: **16.700 rec/s**
DS2 converges in a **single step** for both operators. DS2 converges in **60s**, i.e. as soon as it receives the Heron metrics. Dhalion scales one operator at a time, resulting to a total of **six steps**.

Dhalion converges in **2000s**. Target rate: **16.700 rec/s**
DS2 VS. DHALION ON HERON

DS2 converges in a single step for both operators.

DS2 converges in 60s, i.e. as soon as it receives the Heron metrics.

Dhalion scales one operator at a time, resulting to a total of six steps.

Dhalion converges in 2000s.

Target rate: 16.700 rec/s
DS2 ON APACHE FLINK

wordcount

Records/s

Source Rate
Target Rate

#Instances

FlatMap
Count

Elapsed time [s]

Target rate: 2,000,000 rec/s
Apache Flink savepoint and reconfiguration takes ~30s

Target rate: 2,000,000 rec/s
DS2 ON APACHE FLINK

Apache Flink savepoint and reconfiguration takes ~30s

DS2 converges in two steps for both operators

Target rate: 2.000.000 rec/s
DS2 ON APACHE FLINK

Apache Flink savepoint and reconfiguration takes ~30s

DS2 converges in two steps for both operators

DS2 reacts 3s after the target rate has changed

Target rate: 2,000,000 rec/s
<table>
<thead>
<tr>
<th>initial parallelism</th>
<th>Q1: flatmap</th>
<th>Q2: filter</th>
<th>Q3: incremental join</th>
<th>Q5: tumbling window join</th>
<th>Q8: sliding window</th>
<th>Q11: session window</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 =&gt;</td>
<td>12 =&gt; 16</td>
<td>11 =&gt; 13 =&gt; 14</td>
<td>16 =&gt; 20</td>
<td>14 =&gt; 15 =&gt; 16</td>
<td>10</td>
<td>12 =&gt; 22 =&gt; 28</td>
</tr>
<tr>
<td>12 =&gt;</td>
<td>16</td>
<td>14</td>
<td>18 =&gt; 20</td>
<td>16</td>
<td>10</td>
<td>22 =&gt; 28</td>
</tr>
<tr>
<td>16 =&gt;</td>
<td>16</td>
<td>12 =&gt; 14</td>
<td>20</td>
<td>16</td>
<td>8 =&gt; 10</td>
<td>26 =&gt; 28</td>
</tr>
<tr>
<td>20 =&gt;</td>
<td>16</td>
<td>13 =&gt; 14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
</tr>
<tr>
<td>24 =&gt;</td>
<td>16</td>
<td>14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
</tr>
<tr>
<td>28 =&gt;</td>
<td>16</td>
<td>14</td>
<td>20</td>
<td>13 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
</tr>
</tbody>
</table>

=> : scaling action
## CONVERGENCE - NEXMARK

<table>
<thead>
<tr>
<th>initial parallelism</th>
<th>Q1: flatmap</th>
<th>Q2: filter</th>
<th>Q3: incremental join</th>
<th>Q5: tumbling window join</th>
<th>Q8: sliding window</th>
<th>Q11: session window</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 =&gt; 12 =&gt; 16</td>
<td>11 =&gt; 13 =&gt; 14</td>
<td>16 =&gt; 20</td>
<td>14 =&gt; 15 =&gt; 16</td>
<td>10</td>
<td>12 =&gt; 22 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>12 =&gt; 16</td>
<td>14</td>
<td>18 =&gt; 20</td>
<td>16</td>
<td>10</td>
<td>22 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>16 =&gt; 16</td>
<td>12 =&gt; 14</td>
<td>20</td>
<td>16</td>
<td>8 =&gt; 10</td>
<td>26 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>20 =&gt; 16</td>
<td>13 =&gt; 14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>24 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>28 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>13 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

=> : scaling action
## CONVERGENCE - NEXMARK

<table>
<thead>
<tr>
<th>initial parallelism</th>
<th>Q1: flatmap</th>
<th>Q2: filter</th>
<th>Q3: incremental join</th>
<th>Q5: tumbling window join</th>
<th>Q8: sliding window</th>
<th>Q11: session window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>scale-up</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 =&gt; 12 =&gt; 16</td>
<td>11 =&gt; 13 =&gt; 14</td>
<td>16 =&gt; 20</td>
<td>14 =&gt; 15 =&gt; 16</td>
<td>10</td>
<td>12 =&gt; 22 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>12 =&gt; 16</td>
<td>14</td>
<td>18 =&gt; 20</td>
<td>16</td>
<td>10</td>
<td>22 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>16 =&gt; 16</td>
<td>12 =&gt; 14</td>
<td>20</td>
<td>16</td>
<td>8 =&gt; 10</td>
<td>26 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>20 =&gt; 16</td>
<td>13 =&gt; 14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>24 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>28 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>13 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

=> : scaling action

- **scale-up**
- **scale-down**
<table>
<thead>
<tr>
<th>initial parallelism</th>
<th>Q1: flatmap</th>
<th>Q2: filter</th>
<th>Q3: incremental join</th>
<th>Q5: tumbling window join</th>
<th>Q8: sliding window</th>
<th>Q11: session window</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 =&gt; 12 =&gt; 16</td>
<td>scale-up</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 =&gt; 16</td>
<td>11 =&gt; 13 =&gt; 14</td>
<td>16 =&gt; 20</td>
<td>14 =&gt; 15 =&gt; 16</td>
<td>10</td>
<td>12 =&gt; 22 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>16 =&gt; 14</td>
<td>18 =&gt; 20</td>
<td>16</td>
<td>10</td>
<td>22 =&gt; 28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 =&gt; 16</td>
<td>13 =&gt; 14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>13 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

=> : scaling action

a single step for many queries and initial configurations

scale-up

scale-down
### Convergence - Nexmark

#### Initial Parallelism

<table>
<thead>
<tr>
<th>Initial Parallelism</th>
<th>Q1: Flatmap</th>
<th>Q2: Filter</th>
<th>Q3: Incremental Join</th>
<th>Q5: Tumbling Window Join</th>
<th>Q8: Sliding Window</th>
<th>Q11: Session Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 =&gt; 12 =&gt; 16</td>
<td>11 =&gt; 13 =&gt; 14</td>
<td>16 =&gt; 20</td>
<td>14 =&gt; 15 =&gt; 16</td>
<td>10</td>
<td>12 =&gt; 22 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>12 =&gt; 16</td>
<td>14</td>
<td>18 =&gt; 20</td>
<td>16</td>
<td>10</td>
<td>22 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>16 =&gt; 16</td>
<td>12 =&gt; 14</td>
<td>20</td>
<td>16</td>
<td>8 =&gt; 10</td>
<td>26 =&gt; 28</td>
<td></td>
</tr>
<tr>
<td>20 =&gt; 16</td>
<td>13 =&gt; 14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>24 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>14 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>28 =&gt; 16</td>
<td>14</td>
<td>20</td>
<td>13 =&gt; 16</td>
<td>8 =&gt; 10</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

**At most 3 steps**

- **Scale-up**
  - 8 => 12 => 16
  - 11 => 13 => 14
  - 16 => 20
  - 14 => 15 => 16
  - 10

- **Scale-down**
  - 28 => 16
  - 14
  - 20
  - 13 => 16
  - 8 => 10

**A single step** for many queries and initial configurations

=> : Scaling action
Observed metrics threshold-based policies can lead to oscillations, misconfiguration and slow convergence.

DS2 uses instrumentation to measure true processing and output rates and estimate parallelism for all operators at once.

DS2 makes fast and accurate scaling decisions and converges in up to three steps even for non-linear, complex dataflows.

https://github.com/strymon-system/ds2
Three steps is all you need
fast, accurate, automatic scaling decisions
for distributed streaming dataflows

Vasiliki Kalavri†, John Liagouris†, Moritz Hoffmann†,
Desislava Dimitrova†, Matthew Forshaw‡†, Timothy Roscoe†

†Systems Group, Department of Computer Science, ETH Zürich, firstname.lastname@inf.ethz.ch
‡†Newcastle University, firstname.lastname@newcastle.ac.uk