XStream

Declarative authoring of distributed stream processing pipelines

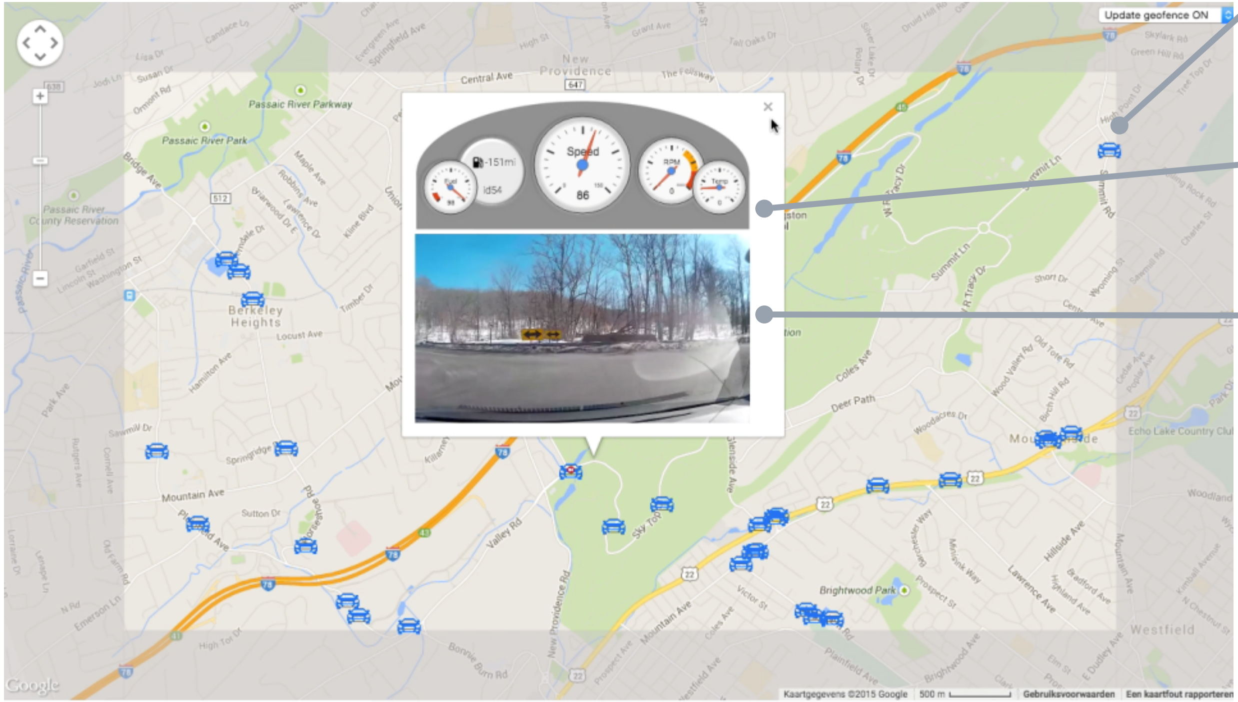
(Or, embedded DSLs make for great stream processing APIs)

- Tom Van Cutsem
- Nokia Bell Labs, Antwerp, Belgium

Demo

Real-time car fleet tracking

ACM DEBS 2017
Best Demo Award



GPS receiver
(position data, 1 update/s)

OBD via CAN bus
(engine data, 1 update/s)

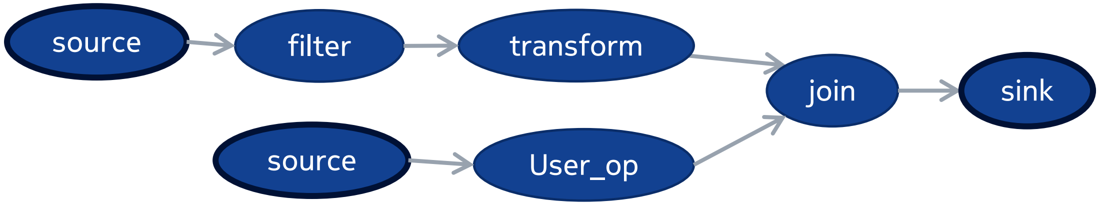
Dashcam
(HD video, H.264 encoded, 500kbps, on-demand)

On-board Unit
(Quad-core ARM Cortex-A9 1Ghz, 1GB RAM + 4G USB Modem)

2 real cars,
10 hours footage
400 virtual cars

Applications in World-wide Streams

- Applications = continuous queries (a.k.a. “flows”) + dashboards (UI widgets)
- Queries are created using a **flow-based programming** approach
- **Library** for JavaScript & TypeScript: **XStream/JS**
- Script generates a dataflow that is optimized by WWS **dataflow compiler**

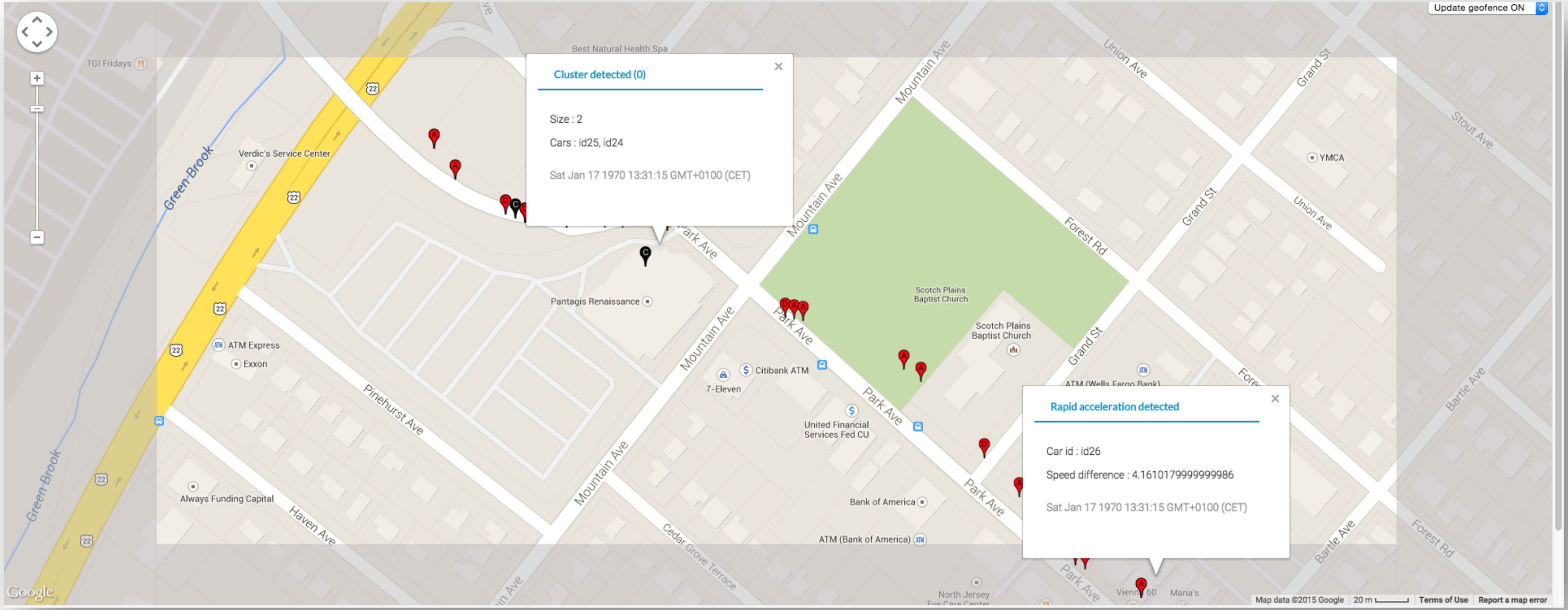


Related Work

- Microsoft Dryad, **DryadLINQ**, Nectar, Naiad
- IBM **JAQL**, System S
- MIT WaveScript
- Google Cloud Dataflow, MapReduce, **FlumeJava**, Sawzall, Millwheel
- Distributed stream processing: Borealis, Stanford STREAM
- Streaming SQL dialects: Esper EQL, StreamSQL, Oracle CQL
- Data-parallel stream processing: Apache Storm, Spark Streaming, Flink, Samza
- Apache Hive
- Apache Quarks

Example flow

Alert me of congested areas near me



  Public

NOKIA Bell Labs

Example flow

Alert me of congested areas near me

```
let all = stream<Array<CarStream>>({"filter": {stream: 'engine'}})
  .pipe(union_streams<Array<CarStream>>())
  .transform(([e]) => [Object.assign({}, e, { id: e.car_id, lng: e.lon })])
  .pipe(resample({ "sample_period": 1 }));

let car = stream({'filter': {stream: 'location', id: $MYCAR }})
  .pipe(union_streams<Array<CarStream>>());

car.transform(([e]) => [Object.assign({}, e, { lng: e.lon })])
  .expand()
  .sink("fenceCenter");

let fence = geofence<Array<CarStream>>({ perimeter: 100 });

all.pipe(fence);
car.pipe(fence.center);

let detector = jerkDetector();
fence.set.pipe(detector);

let clusters = detector.rapid_decel.pipe(geocluster());
clusters.transform((c) =>
  ({ set: c.top_clusters.map(({centroid: {lon, lat}, cluster_id}) => ({lon, lat, cluster_id})) }))
.sink("congestedAreas");
```

Example flow

Alert me of congested areas near me

```
let all = stream<Array<CarStream>>({"filter": {stream: 'engine'}})
  .pipe(union_streams<Array<CarStream>>())
  .transform(([e]) => [Object.assign({}, e, { id: e.car_id, lng: e.lon })])
  .pipe(resample({ "sample_period": 1 }));

let car = stream({'filter': {stream: 'location', id: $MYCAR }})
  .pipe(union_streams<Array<CarStream>>());

car.transform(([e]) => [Object.assign({}, e, { lng: e.lon })])
  .expand()
  .sink("fenceCenter");

let fence = geofence<Array<CarStream>>({ perimeter: 100 });

all.pipe(fence);
car.pipe(fence.center);

let detector = jerkDetector();
fence.set.pipe(detector);

let clusters = detector.rapid_decel.pipe(geocluster());
clusters.transform((c) =>
  ({ set: c.top_clusters.map(({centroid: {lon, lat}, cluster_id}) => ({lon, lat, cluster_id})) }))
.sink("congestedAreas");
```

Stream sources

Stream sinks

Example flow

Alert me of congested areas near me

```
let all = stream<Array<CarStream>>({"filter": {stream: 'engine'}})
  .pipe(union_streams<Array<CarStream>>())
  .transform(([e]) => [Object.assign({}, e, { id: e.car_id, lng: e.lon })])
  .pipe(resample({ "sample_period": 1 }));

let car = stream({'filter': {stream: 'location', id: $MYCAR }})
  .pipe(union_streams<Array<CarStream>>());

car.transform(([e]) => [Object.assign({}, e, { lng: e.lon })])
  .expand()
  .sink("fenceCenter");

let fence = geofence<Array<CarStream>>({ perimeter: 100 });

all.pipe(fence);
car.pipe(fence.center);

let detector = jerkDetector();
fence.set.pipe(detector);

let clusters = detector.rapid_decel.pipe(geocluster());
clusters.transform((c) =>
  ({ set: c.top_clusters.map(({centroid: {lon, lat}, cluster_id}) => ({lon, lat, cluster_id})) }))
.sink("congestedAreas");
```

Built-in operators on streams

  Public

NOKIA Bell Labs

Example flow

Alert me of congested areas near me

```
let all = stream<Array<CarStream>>({"filter": {stream: 'engine'}})
  .pipe(union_streams<Array<CarStream>>())
  .transform(([e]) => [Object.assign({}, e, { id: e.car_id, lng: e.lon })])
  .pipe(resample({ "sample_period": 1 }));

let car = stream({'filter': {stream: 'location', id: $MYCAR }})
  .pipe(union_streams<Array<CarStream>>());

car.transform(([e]) => [Object.assign({}, e, { lng: e.lon })])
  .expand()
  .sink("fenceCenter");

let fence = geofence<Array<CarStream>>({ perimeter: 100 });

all.pipe(fence);
car.pipe(fence.center);

let detector = jerkDetector();
fence.set.pipe(detector);

let clusters = detector.rapid_decel.pipe(geocluster());
clusters.transform((c) =>
  ({ set: c.top_clusters.map(({centroid: {lon, lat}, cluster_id}) => ({lon, lat, cluster_id})) }))
.sink("congestedAreas");
```

"External" (user-defined) operators

Example flow

Alert me of congested areas near me

```
let all = stream<Array<CarStream>>({"filter": {stream: 'engine'}})
  .pipe(union_streams<Array<CarStream>>())
  .transform(([e]) => [Object.assign({}, e, { id: e.car_id, lng: e.lon })])
  .pipe(resample({ "sample_period": 1 }));

let car = stream({'filter': {stream: 'location', id: $MYCAR }})
  .pipe(union_streams<Array<CarStream>>());

car.transform(([e]) => [Object.assign({}, e, { lng: e.lon })])
  .expand()
  .sink("fenceCenter");

let fence = geofence<Array<CarStream>>({ perimeter: 100 });

all.pipe(fence);
car.pipe(fence.center);

let detector = jerkDetector();
fence.set.pipe(detector);

let clusters = detector.rapid_decel.pipe(geocluster());
clusters.transform((c) =>
  ({ set: c.top_clusters.map(({centroid: {lon, lat}, cluster_id}) => ({lon, lat, cluster_id})) }))
.sink("congestedAreas");
```

External operator “wiring”

  Public

NOKIA Bell Labs

Example flow

Alert me of congested areas near me

```
let all = stream<Array<CarStream>>({"filter": {stream: 'engine'}})
  .pipe(union_streams<Array<CarStream>>())
  .transform(([e]) => [Object.assign({}, e, { id: e.car_id, lng: e.lon })])
  .pipe(resample({ "sample_period": 1 }));

let car = stream({'filter': {stream: 'location', id: $MYCAR }})
  .pipe(union_streams<Array<CarStream>>());

car.transform(([e]) => [Object.assign({}, e, { lng: e.lon })])
  .expand()
  .sink("fenceCenter");

let fence = geofence<Array<CarStream>>({ perimeter: 100 });

all.pipe(fence);
car.pipe(fence.center);

let detector = jerkDetector();
fence.set.pipe(detector);

let clusters = detector.rapid_decel.pipe(geocluster());
clusters.transform((c) =>
  ({ set: c.top_clusters.map(({centroid: {lon, lat}, cluster_id}) => ({lon, lat, cluster_id})) }))
.sink("congestedAreas");
```

Delayed/remote code execution

  Public

NOKIA Bell Labs

Launching XStream Flows

XStream Developer

The "Flow"

Functional "How"

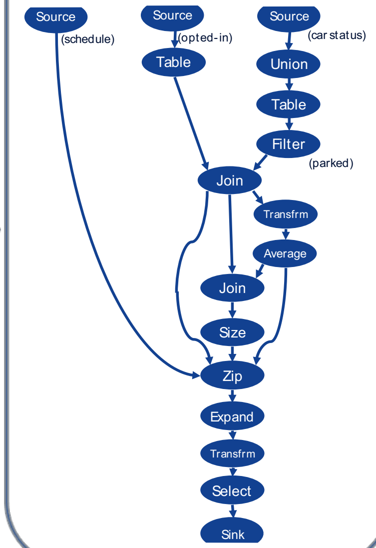
Implementation "How"

Deployment "How"

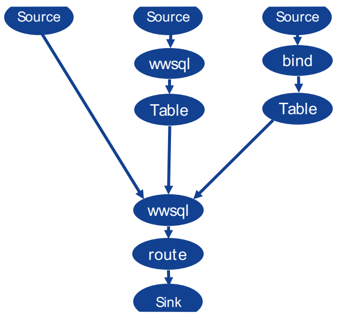
Functional dataflow specification

```
let fence =
  resolve_streams({type: "Car", port: "engine"})
  -> resample({ period: 1, primary_key: "id" })
  -> geofence({ perimeter: 100 });

let car = resolve_stream({
  id: $MYCAR,
  type: 'Car',
  port: 'location' });
car -> fence.center;

let detector = fence.set -> detectSpeedDrop({
  threshold: 10, // 10 mph drop
  time_window: 2 // within 2 seconds
});

detector
  -> geocluster({
       proximity_radius: 100, // meters
       min_cluster_size: 2, // at least 2 cars
       time_window: 5 // within 5 seconds
    })
  -> sink("brakeAlert");
```

Logical Query Plan

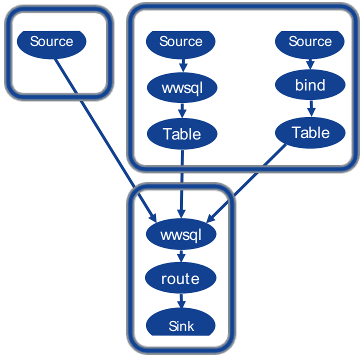


Physical Query Plan



(optimized plan, e.g. operators are fused)

Placed Query Plan



XStream: external or embedded DSL?

```
function relay({events: ev, switches: s}) {
    let left = ev -> transform [$,null];
    let right = s -> filter $ == true
                  -> transform [null,$];

    union(left, right)
    -> reduce [null,false] [[pe,pb], [e,b]]:
                           [e ?? pe, b ?? false]
    -> filter [e,b]: b
    -> transform [e,b]: e
};
```

XStream DSL

```
function relay({ events: ev, switches: s }) {
  let left = ev.transform($ => [$, null]);
  let right = s.filter(($) => $ === true)
               .transform($ => [null, $]);

  return union(left, right)
    .reduce([null, false], ([[pe, pb], [e, b]]) =>
                            [e || pe, b !== null])
    .filter(([e, b]) => b)
    .transform(([e, b]) => e);
}
```

XStream/JS

Embedded DSLs make for great stream processing APIs

“fluent APIs” in Java/Scala: Apache Storm, Apache Spark (incl. Spark Streaming), Apache Flink

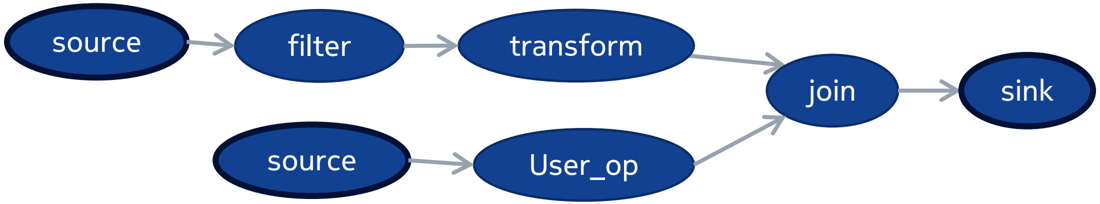
Craig Chambers et al. on FlumeJava’s predecessor called “Lumberjack” (PLDI 2010):

- “The implicitly parallel, mostly functional programming model was **not natural** for many of its intended users. FlumeJava’s explicitly parallel model [...] coupled with its “mostly imperative” model [...], is much more natural for most of these programmers."
- LumberJack: **static analysis** (hard and imprecise) vs FlumeJava: just run the program to generate the graph and then reason from that. Simpler *and* more precise.
- **Tooling**. “Building an efficient, complete, usable Lumberjack-based system is much more difficult [...] than building an equivalently efficient, complete, and usable FlumeJava system.”
- **“Novelty is an obstacle to adoption.** By being embedded in a well-known programming language, FlumeJava focuses the potential adopter’s attention on a few new features, namely the Flume abstractions and the handful of Java classes and methods implementing them.”

NOKIA Bell Labs

Summary: XStream

- A high-level query interface to compose end-to-end dataflows
- Embeddable as a library in existing programming languages
- Compiler optimizes generated query plan prior to deployment
- Deployer deploys operators across (wide-area) distributed execution environment



JS TS

  Public

NOKIA Bell Labs

NOKIA

Copyright and confidentiality