Lucidworks

# Building Analytics Applications with Streaming Expressions in Apache Solr

Amrit Sarkar

Cloud Search Reliability Engineer

Lucidworks Inc

@sarkaramrit2

#BerlinBuzzwords19

Lucidworks

# Who are we?

Based in San Francisco

Offices in Cambridge, Bangalore, Bangkok, New York City, Raleigh, Munich

Over 300 customers across the Fortune 1000

Fusion, a Solr-powered platform for search-driven apps

Consulting and support for organizations using Solr

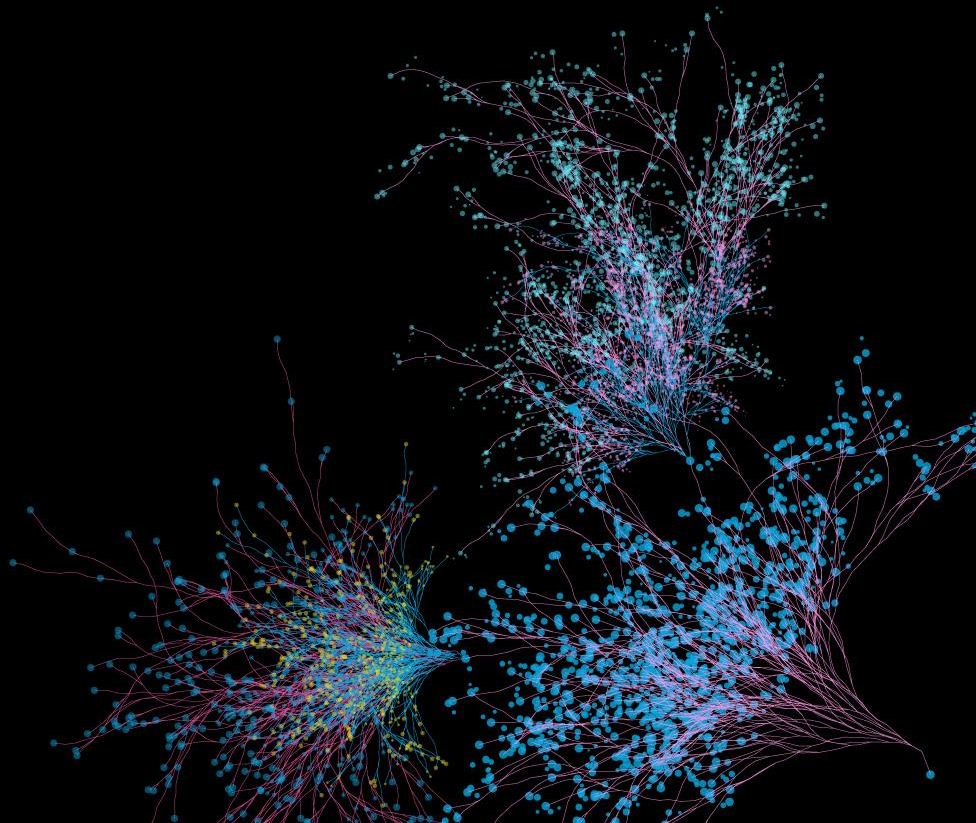
# Agenda

Lucidworks

- Parallel Computing Framework

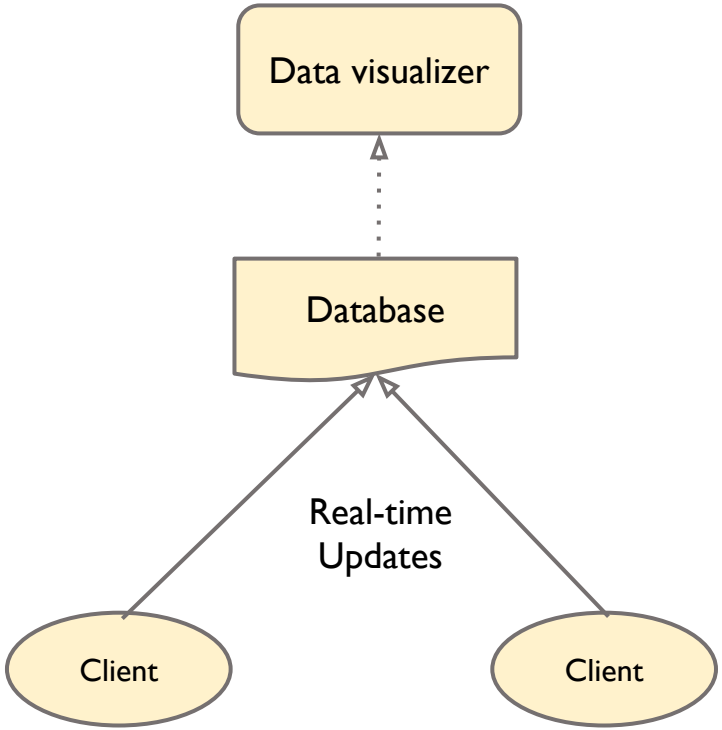
  Introduction to

  - Streaming API
  - Streaming Expressions
  - Types of Expressions
  - Shuffling
  - Workers
- Real-Life Use Cases
- Demo Application
- Performance Analysis
- Statistical Programming



# Challenges building applications on real-time data

Lucidworks




- Searching and filtering the data before performing analytics.
- Executing complex operations & co-relations on unstructured and non-preprocessed data is time consuming.
- Dependencies on multiple tools leading to higher maintenance cost.


## Parallel Computing Framework

Available in SolrCloud mode

- Streaming API
- Streaming Expressions
- Shuffling
- Worker collections
- Parallel SQL

# Streaming API

Lucidworks

- Java API for parallel computation
- Real-time Mapreduce and Parallel Relational Algebra
- Results are streams of tuples (key/value) (TupleStream)
- org.apache.solr.client.solrj.io.*

```
ParallelStream pstream =
(ParallelStream) streamFactory.constructStream("parallel(collectionName, ……....)");
pstream.open();
```

# Streaming Expressions

Use case: perform full index search and retrieve specific fields sorted

```
curl --data-urlencode 'expr=
search(gettingstarted,
zkHost="localhost:9983",
qt="/export",
q="hatchbacks",
fq="year:2014",
fl="id, model_name",
sort="id asc"))'
http://localhost:8983/solr/
gettingstarted/stream
```

- String Query Language and Serialisation format for the Streaming API
- Streaming expressions compile to TupleStream; TupleStream serialise to Streaming Expressions
- Can be used directly via HTTP to SolrJ
- Expressions can be executed against Solr API: */solr/<collection-name>/stream*

# Streaming Expressions

Use case: perform full index search and retrieve specific fields sorted

```
curl --data-urlencode 'expr=
search(gettingstarted,
zkHost="localhost:9983",
qt="/export",
q="hatchbacks",
fq="year:2014",
fl="id, model_name",
sort="id asc"))'
http://localhost:8983/solr/
gettingstarted/stream
```



```
{
 "result-set": {
  "docs": [
   {
    "model_name": "M",
    "id": "1"
   },
   {
    "model_name": "N",
    "id": "2"
   },
   {
    "model_name": "O",
    "id": "3"
   },
   {
    "EOF": true,
    "RESPONSE_TIME": 12
   }
  ]
 }
}
```

# Streaming Expressions

Lucidworks

- **Stream Sources**

The origin of a TupleStream

*search, facet, jdbc, stats, topic, timeseries, train and more..*

- **Stream Decorators**

Wrap other stream functions and perform operations on the stream, row wise

*complement, hashJoin, innerJoin, merge, intersect, top, unique and more..*

- **Stream evaluators**

evaluate (calculate) new values based on other values in a tuple, column wise

*add, eq, div, mul, sub, length, asin, acos, abs, if:then and more..*

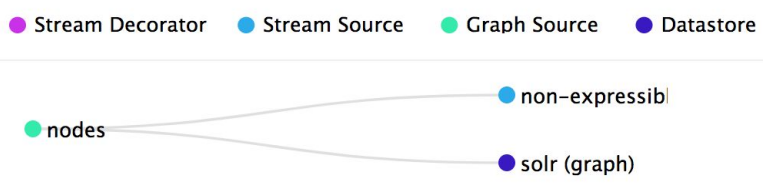
# Streaming Expressions - Use cases

Lucidworks

**Use case:** Destinations reachable with single stop from ‘New York’

(graphical traversal)

```
nodes(distances,
      nodes(distances,
            walk="New York->source_s",
            gather="destination_s"),
      walk="node->source_s",
      gather="destination_s",
      trackTraversal="true",
      scatter="branches,leaves")
```





```
{
  "result-set": {
    "docs": [
      {
        "node": "New York",
        "collection": "distances",
        "field": "node",
        "ancestors": [],
        "level": 0
      },
```

```
{
  "node": "Bengaluru",
  "field": "destination_s",
  "level": 2,
  "count(*)": 2,
  "collection": "distances",
  "ancestors": [
    "Paris",
    "New Delhi"
  ]
},
```

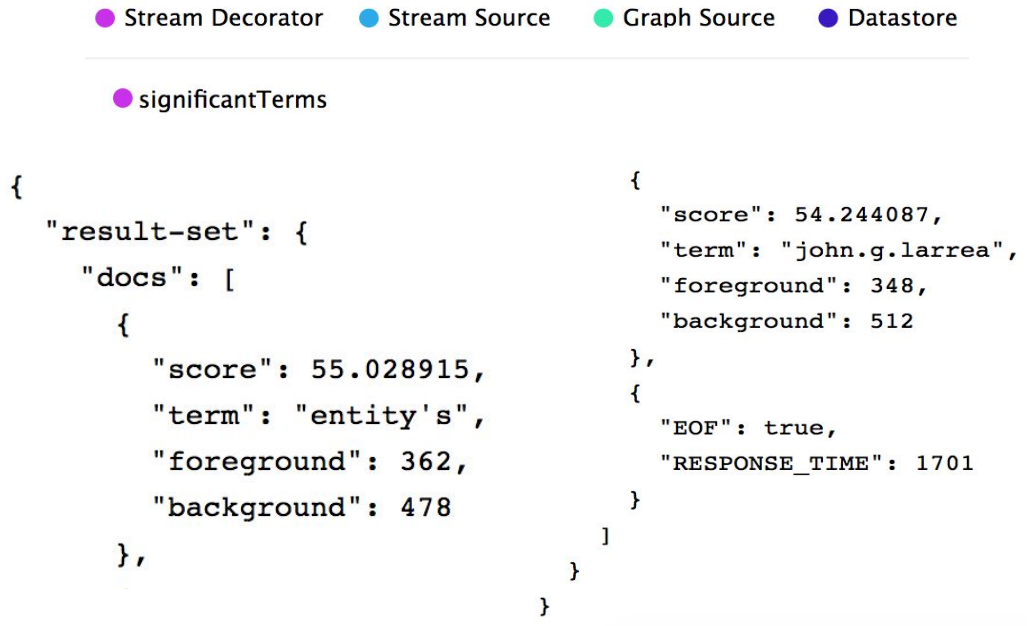

Solr indexes are stored in ‘token’ to ‘document-ids’ format, **‘nodes’** perform BFS on field tokens.

# Streaming Expressions - Use cases

Lucidworks

**Use case:** Determine most relevant terms on dynamic data set

```
significantTerms(
    enron-emails,
    q="To:Tim Belden",
    field="content",
    limit="2",
    minDocFreq="10",
    maxDocFreq=".20",
    minTermLength="5"
)
```





```
{
  "result-set": {
    "docs": [
      {
        "score": 55.028915,
        "term": "entity's",
        "foreground": 362,
        "background": 478
      },
      {
        "score": 54.244087,
        "term": "john.g.larrea",
        "foreground": 348,
        "background": 512
      },
      {
        "EOF": true,
        "RESPONSE_TIME": 1701
      }
    ]
  }
}
```


Solr indexes are stored in 'token' to 'document-ids' format, **'significantTerms'** aggregates over tokens.

# Streaming Expressions - Use cases

Lucidworks

**Use case:** Calculate useful metrics on data fetched from various sources.

- conversion ratio (conversions to clicks)
- CTR (clicks to impressions)
- cost ratio (conversions to currency cost)

| campaign_id_s | currency_cost_i |
|---|---|
| cmp-01 | 6600 |
| cmp-02 | 5840 |
| cmp-03 | 8400 |

Campaign costs stored in mysql table 'cost'

| campaign_id_s | org_id_s | conversions_i | impressions_i | clicks_i |
|---|---|---|---|---|
| cmp-01 | org-01 | 4 | 134 | 48 |
| cmp-02 | org-02 | 2 | 174 | 26 |
| cmp-03 | org-01 | 6 | 152 | 49 |
| cmp-01 | org-01 | 5 | 154 | 27 |
| cmp-02 | org-01 | 9 | 176 | 38 |
| cmp-03 | org-01 | 5 | 137 | 83 |
| cmp-01 | org-01 | 3 | 154 | 36 |
| cmp-02 | org-02 | 1 | 178 | 35 |
| cmp-03 | org-01 | 7 | 124 | 49 |
| ..... | ..... | ..... | ..... | ..... |

Events captured in solr collection 'weekly_data'

# Streaming Expressions - Use cases

Lucidworks

**Use case:** Join cost data with aggregated conversions, clicks and impressions per campaign for organisation 'org-01'

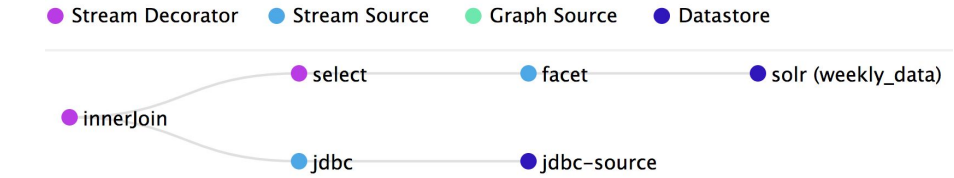
```
innerJoin(
    select(
        facet(weekly_data, q="org_id_s:org-01", buckets="campaign_id_s",
        bucketSorts="campaign_id_s asc", bucketSizeLimit=100,
        sum(conversations_i), sum(impressions_i), sum(clicks_i)),
    campaign_id_s as campaign_id_s, sum(conversations_i) as aggr_conv,
    sum(impressions_i) as aggr_impr, sum(clicks_i) as aggr_clicks),
    jdbc(connection="jdbc:mysql://localhost/cost_db?user=root&password=root",
    sql="SELECT campaign_id_s,currency_cost_i FROM cost",
    sort="campaign_id_s asc", driver="com.mysql.jdbc.Driver"),
on="campaign_id_s")
```

# Streaming Expressions - Use cases

Lucidworks

**Use case:** Join cost data with aggregated conversions, clicks and impressions per campaign for organisation 'org-01'

```
innerJoin(
    select(
        facet(weekly_data,
        q="org_id_s:org-01",
        buckets="campaign_id_s",
        bucketSorts="campaign_id_s asc",
        bucketSizeLimit=100,
        sum(conversations_i),
        sum(impressions_i),
        sum(clicks_i)),
    campaign_id_s as campaign_id_s,
    sum(conversations_i) as aggr_conv,sum(impressions_i)
    as aggr_impr, sum(clicks_i) as aggr_clicks),
    jdbc(connection="jdbc:mysql://localhost/cost_db?
    user=root&password=root",
    sql="SELECT campaign_id_s,currency_cost_i FROM cost",
    sort="campaign_id_s asc",
    driver="com.mysql.jdbc.Driver"),
on="campaign_id_s")
```





```
{
  "result-set": {
    "docs": [
      {
        "aggr_conv": 41,
        "aggr_impr": 1008,
        "currency_cost_i": 6600,
        "aggr_clicks": 259,
        "campaign_id_s": "cmp-01"
      },
      {
        "aggr_conv": 35,
        "aggr_impr": 1135,
        "currency_cost_i": 5840,
        "aggr_clicks": 297,
        "campaign_id_s": "cmp-02"
      },
      {
        "aggr_conv": 43,
        "aggr_impr": 1068,
        "currency_cost_i": 8400,
        "aggr_clicks": 394,
        "campaign_id_s": "cmp-03"
      },
      {
        "EOF": true,
        "RESPONSE_TIME": 57
      }
    ]
  }
}
```

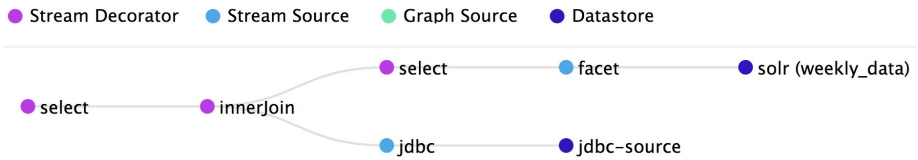
# Streaming Expressions - Use cases

Lucidworks

**Use case:** Calculate useful metrics on data fetched from various sources for ‘org-01’:

- conversion ratio (conversions to clicks)
- CTR (clicks to impressions)
- cost ratio (conversions to currency cost)

```
select(
      innerJoin(
      ……..
      on="campaign_id_s"),
div( aggr_conv, aggr_clicks )
as conversion_ratio,
div( aggr_clicks , aggr_impr )
as ctr,
div( currency_cost_i, aggr_conv)
as campaign_cost_ratio)
```





```
"result-set": {
  "docs": [
    {
      "ctr": 0.2569444444444444,
      "conversion_ratio": 0.1583011583011583,
      "campaign_cost_ratio": 1.609756097560976
    },
    {
      "ctr": 0.2616740088105727,
      "conversion_ratio": 0.1178451178451178,
      "campaign_cost_ratio": 1.668571428571429
    },
    {
      "ctr": 0.3689138576779026,
      "conversion_ratio": 0.1091370558375635,
      "campaign_cost_ratio": 1.953488372093023
    },
```

# Streaming Expressions - Use Cases

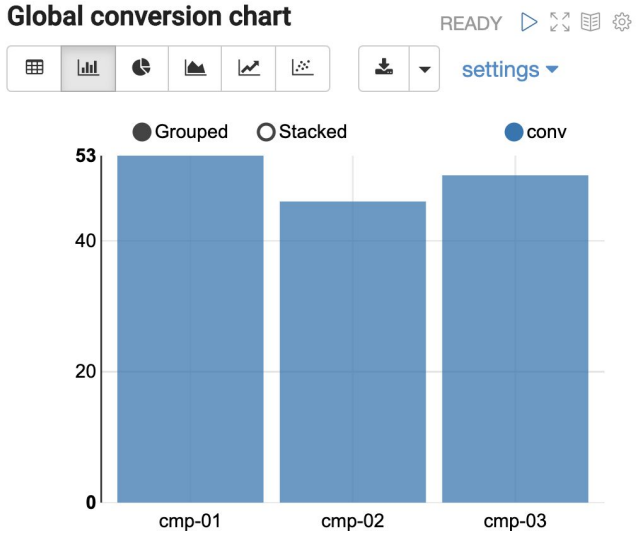
Lucidworks

**Use case:** Calculate useful metrics on data fetched from different sources for organisations and campaigns:

- Ratios
  - conversion ratio (conversions to clicks)
  - CTR (clicks to impressions)
  - cost ratio (conversions to currency cost)
- Time-series ratios
- Rankings: multi-faceted

Plot analytics dashboards on Apache Zeppelin using Solr Interpreter

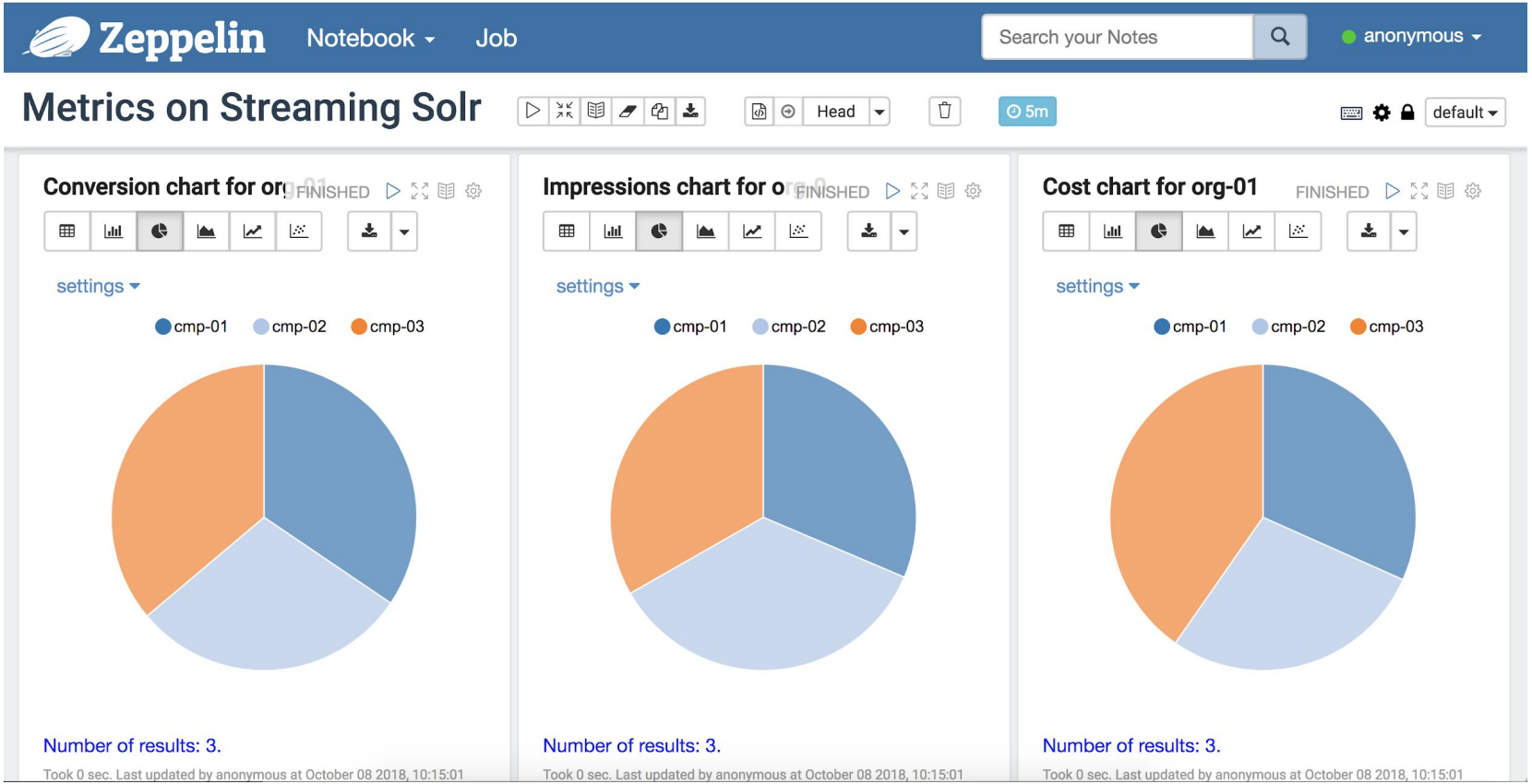
(Kiran Chitturi, Lucidworks Inc)




Apache Zeppelin Notebook

# Streaming Expressions - Demo Application

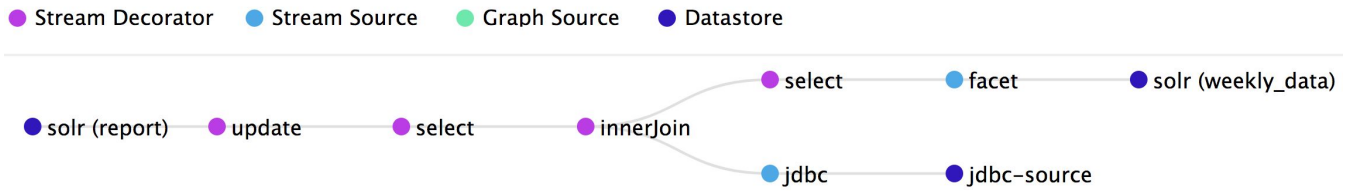
Lucidworks



# Streaming Expressions - Use cases

Lucidworks

**Use case:** Create a view from result-set of previously discussed use-case: calculate metrics (index data to new collection)




```
update(
  report, batchSize=500,
   select( ……..
  campaign_id_s as campaign)
  )
```

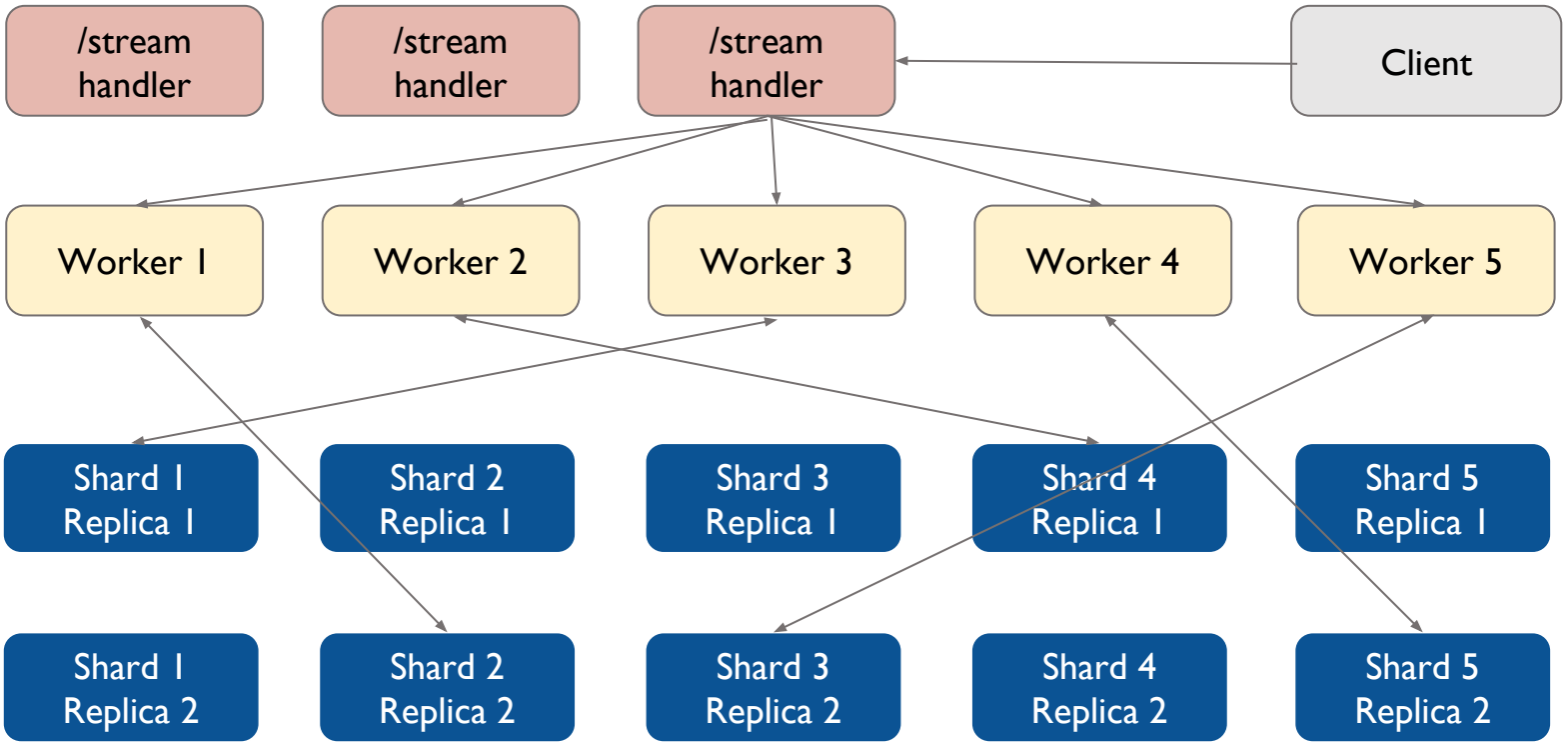
complexity - O(N)

N - total rows processed


```
"docs": [
  {
    "batchIndexed": 3,
    "totalIndexed": 3,
    "worker": "currency_cost_shard4_replica_n14",
    "batchNumber": 1
  },
```

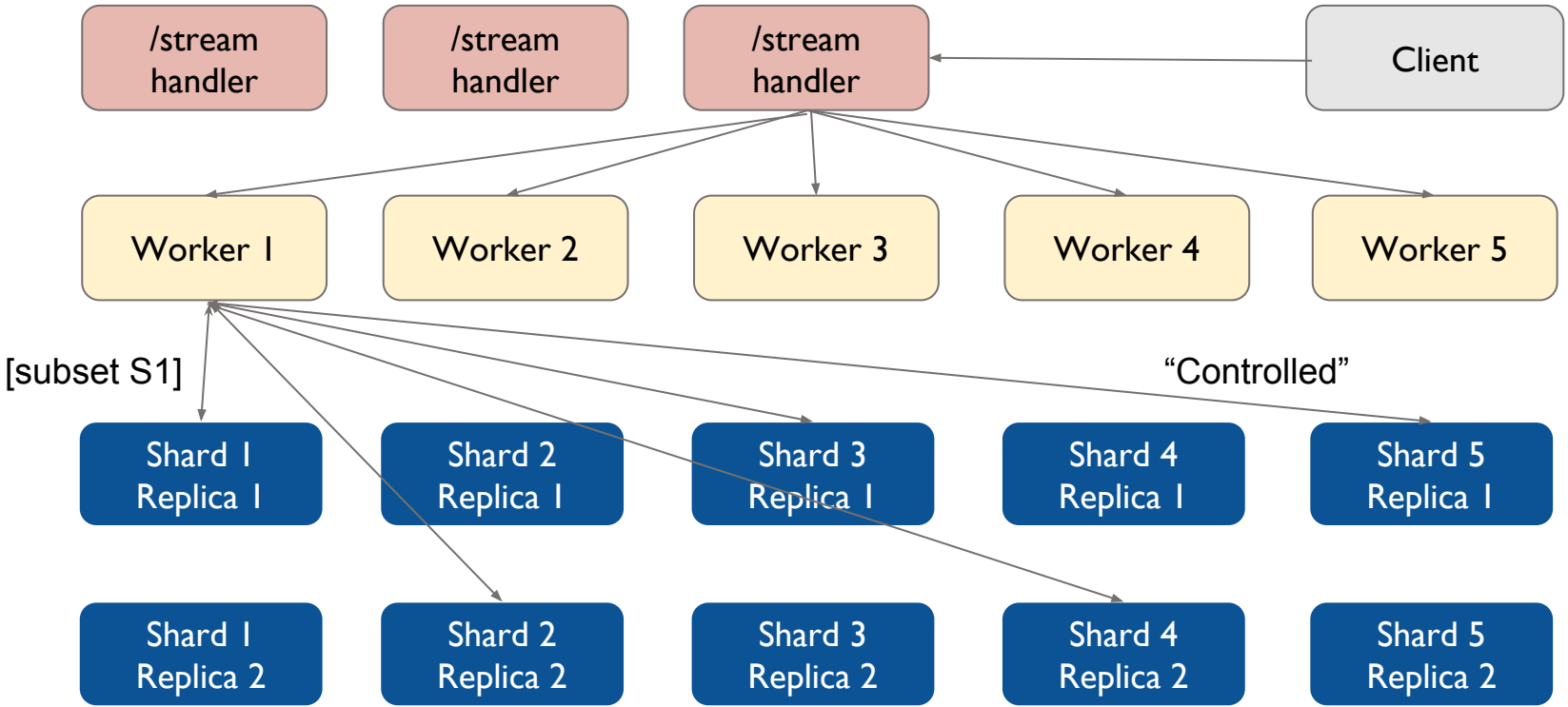
# Streaming Expressions - Shuffle

Lucidworks



# Streaming Expressions - Shuffle

Lucidworks



Streaming Expressions

# Worker Collections

- Regular SolrCloud collections
- Perform streaming aggregations using the Streaming API
- Receive shuffled streams from replicas
- May be empty or created just-in-time or have regular data
- The goal is to separate processing from data if necessary

# Streaming Expressions - Use cases

Lucidworks

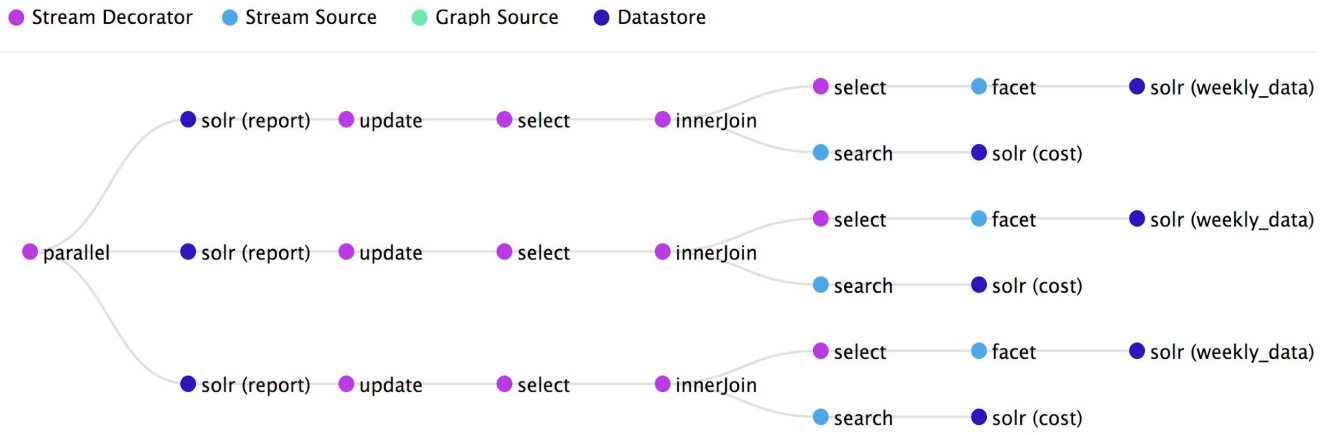
**Use case:** Indexing the result-set of discussed use-case (calculate metrics for organisation) to new collection ‘report’ parallely utilising ‘n’ workers

```
parallel(worker,
    update(report,batchSize=10,
    select(
    innerJoin(
    select(
    facet(weekly_data, q="org_id_s:org-01", buckets="campaign_id_s", bucketSorts="campaign_id_s asc", bucketSizeLimit=100,
    sum(conversations_i), sum(impressions_i), sum(clicks_i), partitionKeys="campaign_id_s"),
    campaign_id_s as campaign_id_s, sum(conversations_i) as aggr_conv, sum(impressions_i) as aggr_impr, sum(clicks_i) as aggr_clicks),
    search(cost, zkHost="localhost:9983", qt="/export",q="*:*", fl="campaign_id_s,org_id_s,currency_cost_i",
    partitionKeys="campaign_id_s", sort="campaign_id_s asc"),
    on="campaign_id_s"),
    div( aggr_conv, aggr_clicks ) as conversion_ratio, div( aggr_clicks , aggr_impr ) as ctr, div( currency_cost_i, aggr_conv)
    as campaign_cost_ratio, campaign_id_s as campaign)),
workers=3,
zkHost="localhost:9983",
sort="campaign asc")
```

# Streaming Expressions - Use cases

Lucidworks

**Use case:** Indexing the result-set of discussed use-case (calculate metrics for organisation) to new collection 'report' parallely utilising 'n' workers.





```
{
  "batchIndexed": 1,
  "totalIndexed": 1,
  "worker": "worker_shard2_replica_n4"
  "batchNumber": 1
},
{
  "batchIndexed": 1,
  "totalIndexed": 1,
  "worker": "worker_shard3_replica_n8"
  "batchNumber": 1
},
{
  "batchIndexed": 1,
  "totalIndexed": 1,
  "worker": "worker_shard4_replica_n12
  "batchNumber": 1
},
```


complexity - $Z(W) + O(N \times W)/W \sim O(N)/W$

N - total rows processed  Z - aggregation  W - number of workers utilised  N >>W

# Statistical Programming

Lucidworks

- Solr's powerful data retrieval capabilities can be combined with in-depth statistical analysis.
  - SQL, anomaly detection, time-series aggregation, Linear regressions and more..
- Syntax can be used to create arrays from the data so it can be manipulated, transformed and analyzed; can be used to train models and predict from historical data.
- Statistical function library:
  - Percentiles, Euclidean Distance, Normal Distribution, Covariances and more.
  - backed by Apache Common Maths Library

# Statistical Programming - Use cases

Lucidworks

**Use case:** Determine correlation among stocks from their historical data.

Correlation measures the extent that two variables fluctuate together. For example if rise of stock A typically coincides with rise in stock B they are positively correlated. If rise in stock A typically coincides with fall in stock B they are negatively correlated.

Data Representation:

| EventID (unique) | StockID | Date | Closing points |
|---|---|---|---|
| stockA-1 | stockA | 01-02-2013 | 30 |
| stockB-1 | stockB | 01-02-2013 | 168 |
| stockC-1 | stockC | 01-02-2013 | 356 |
| stockB-2 | stockB | 02-02-2013 | 237 |
| stockA-2 | stockA | 02-02-2013 | 43 |
| …….. | …….. | …….. | …….. |

Feb 2013 to Jan 2017

# Statistical Programming - Use cases

Lucidworks

**Use case:** Determine correlation among stocks A to B from their historical data.

```
let(
    stockA=search(historical_stocks_data,
    zkHost="localhost:9983", qt="/export",
    q="stock_s:stockA", fl="timestamp_dt, closing_pts_i",
    sort="timestamp_dt asc"),

    stockB=search(historical_stocks_data,
    zkHost="localhost:9983", qt="/export",
    q="stock_s:stockB", fl="timestamp_dt, closing_pts_i",
    sort="timestamp_dt asc"),

  pricesA = col(stockA, closing_pts_i),
  pricesB = col(stockB, closing_pts_i),

tuple(correlation=corr(pricesA, pricesB)))
```

set variables and outputs single tuple

limit the resultset to stockA, assign to variable 'stockA'

limit the resultset to stockB, assign to variable 'stockB'

'col' func creates array from a list of Tuples

corr evaluator which performs the *Pearson product-moment correlation* calculation on two columns of numbers.

# Statistical Programming - Use cases

Lucidworks

**Use case:** Determine correlation among stocks A to B and C from their historical data.


```
"result-set": {
  "docs": [
    {
      "correlation": 0.999015757799239
    },
    {
      "EOF": true,
      "RESPONSE_TIME": 76
    }
```


'A' to 'B'

Stock 'A' is ***highly positively*** correlated to stock 'B', indicating if there is a future prediction for stock 'B' to rise, it is highly likely stocks prices for stock 'A' will rise too and similar trend will follow if falling.


```
"result-set": {
  "docs": [
    {
      "correlation": -0.18167359393816224
    },
    {
      "EOF": true,
      "RESPONSE_TIME": 99
    }
```


'A' to 'C'

Stock 'A' is ***moderately negatively*** correlated to stock 'C', indicating prediction for stock 'A' cannot be relied upon stock 'C' trend.

# Statistical Programming - on Zeppelin

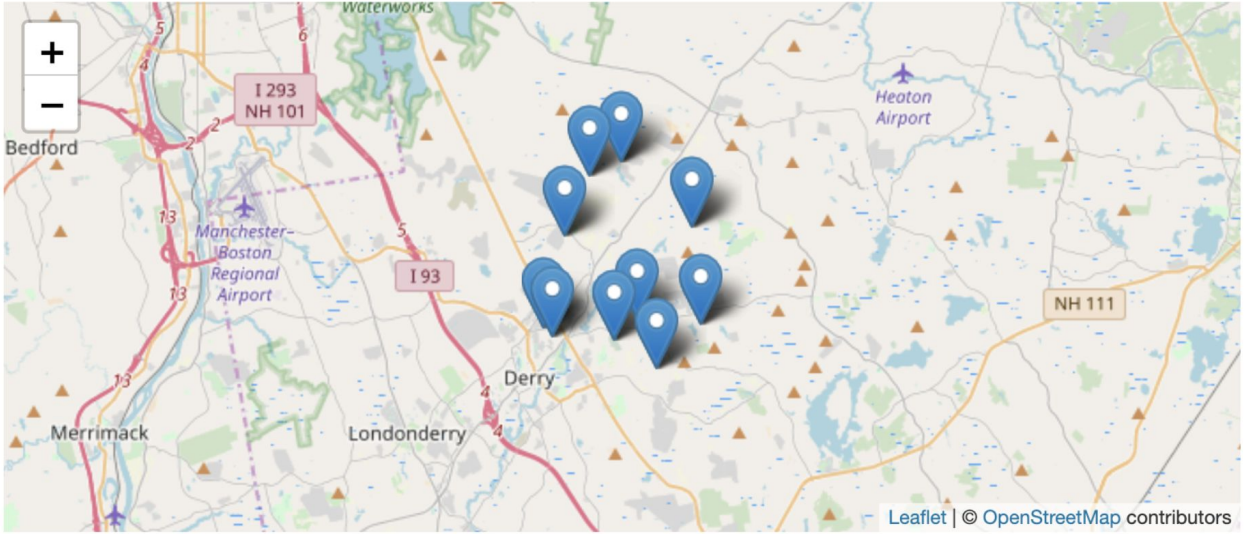
Lucidworks

## Mapping

```
let(a=search(testapp, fl="id, loc_p"),
    b=latlonVectors(a, field="loc_p"),
    lat=colAt(b, 0),
    lon=colAt(b, 1),
    i=col(a, id),
    zplot(lat=lat, lon=lon, id=i))
```

FINISHED

settings




Number of results: 10.

# Statistical Programming - on Zeppelin

Lucidworks

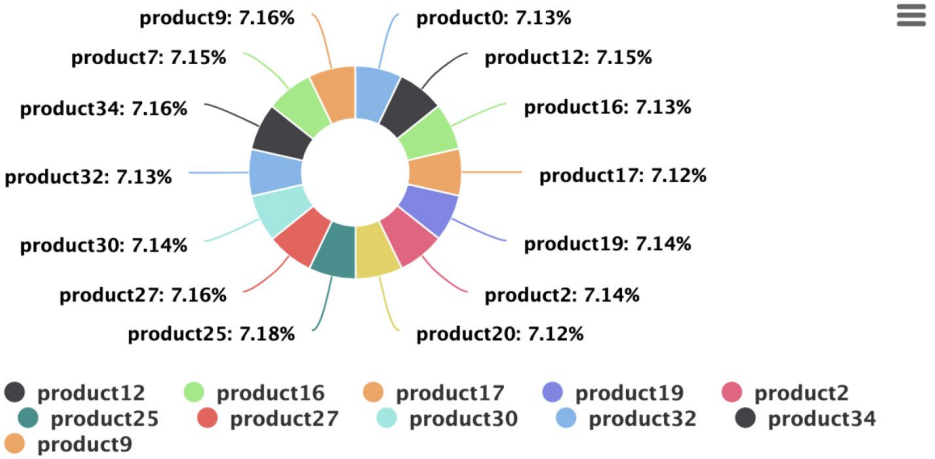
## SQL Aggregations

```
sql(stmt="
            select prod_ss, count(*)
                from testapp
            group by prod_ss
                order by count(*) desc
            limit 14

            ")
```

FINISHED

settings




Number of results: 14.

# Streaming Expressions & DeepLearning4j

Lucidworks

- Eclipse Deeplearning4j is first commercial-grade, open-source, distributed deep-learning library written for Java and Scala.
- DataSetIterator handles traversing through a dataset and preparing data for a neural network.

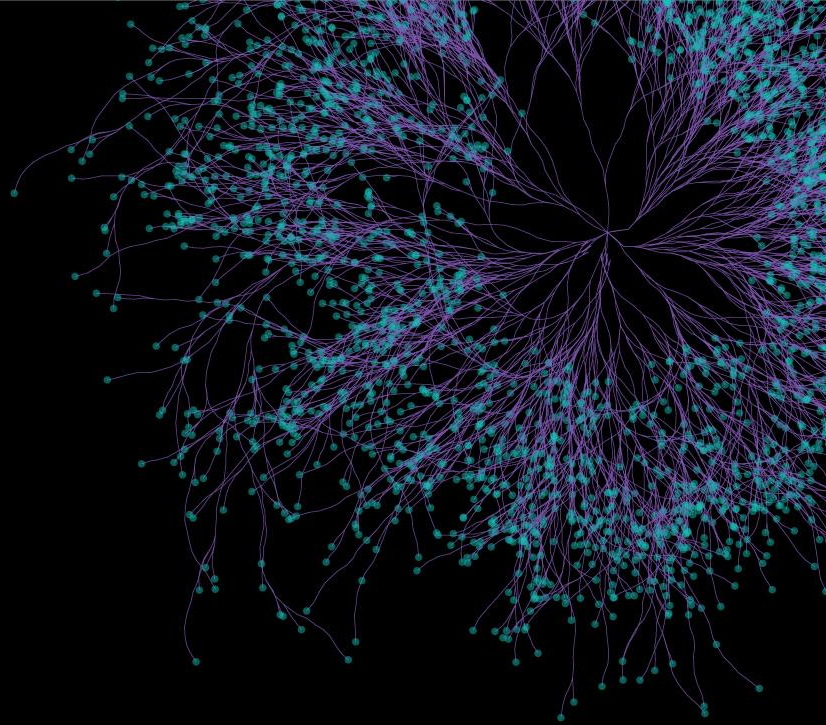
- TupleStreamDataSetIterator is introduced in 1.0.0-beta2 by Christine Poerschke, Committer PMC Apache Solr.
- Fetches data via Streaming Expressions, sources like Solr Collections, JDBC etc.

# References & Knowledge Base

Lucidworks

- Use cases and examples available on Github: /sarkaramrit2/stream-solr
- Streaming expression official documentation in Apache Solr.
- Statistical Programming official documentation in Apache Solr.
- Joel Bernstein's blog.
- Zeppelin Visualizer for Streaming Solr.
- Presentation links:
  - Applied Mathematical Modeling with Apache Solr
  - The Evolution of Streaming Expressions
  - Streaming Aggregation, New Horizons for Search
  - Analytics and Graph Traversal with Solr
  - Creating New Streaming Expressions

# Thank you!



Amrit Sarkar
Cloud Search Reliability Engineer
Lucidworks Inc

@sarkaramrit2
#BerlinBuzzwords19

Lucidworks