Lucidworks

Parallel SQL and Streaming Expressions in Apache Solr 6

Shalin Shekhar Mangar
@shalinmangar
Lucidworks Inc.

Introduction

- Shalin Shekhar Mangar
- Lucene/Solr Committer
- PMC Member
- Senior Solr Consultant with Lucidworks Inc.

Solr

The standard for enterprise search.

Solr Key Features

- Full text search (Info Retr.)
- Facets/Guided Nav galore!
- Lots of data types
- Spelling, auto-complete, highlighting
- Cursors
- More Like This
- De-duplication
- Apache Lucene
- Grouping and Joins
- Stats, expressions, transformations and more
- Lang. Detection
- Extensible
- Massive Scale/Fault tolerance

Why SQL

- Simple, well-known interface to data inside Solr
- Hides the complexity of Solr and its various features
- Possible to optimise the query plan according to best-practices automatically
- Distributed Joins done simply and well

Solr 6: Parallel SQL

- Parallel execution of SQL across SolrCloud collections
- Compiled to SolrJ Streaming API (TupleStream) which is a general purpose parallel computing framework for Solr
- Executed in parallel over SolrCloud worker nodes
- SolrCloud collections are relational ‘tables’
- JDBC thin client as a SolrJ client

Solr's SQL Interface

SQL Interface at a glance

- SQL over Map/Reduce — for high cardinality aggregations and distributed joins
- SQL over Facets — high performance, moderate cardinality aggregations
- SQL with Solr powered search queries
- Fully integrated with SolrCloud
- SQL over JDBC or HTTP — http://host:port/solr/collection1/sql

Limited vs Unlimited SELECT

- select movie, director from IMDB

  Returns the entire result set! Return fields must be DocValues

- select movie, directory from IMDB limit 100

  Returns specified number of records. It can sort by score and retrieve any stored field

- select movie, director from IMDB order by rating desc, num_voters desc

Search predicates

- select movie, director from IMDB where actor = ‘bruce’
- select movie, director from IMDB where actor = ‘(bruce tom)’
- select movie, director from IMDB where rating = ‘[8 TO *]’
- select movie, director from IMDB where (actor = ‘(bruce tom)’ AND rating = ‘[8 TO *]’)

  Search predicates are Solr queries specified inside single-quotes

  Can specify arbitrary boolean clauses

Select DISTINCT

- select distinct actor_name from IMDB
- Map/Reduce implementation — Tuples are shuffled to worker nodes and operation is performed by workers
- JSON Facet implementation — operation is ‘pushed down’ to Solr

Stats aggregations

- select count(*), sum(num_voters) from IMDB
- Computed using Solr's StatsComponent under the hood
- count, sum, avg, min, max are the supported aggregations
- Always pushed down into the search engine

GROUP BY Aggregations

- select actor_name, director, count(*), sum(num_voters) from IMDB group by actor_name, director having count(*) > 5 and sum(num_voters) > 1000 order by sum(num_voters) desc
- Has a map/reduce implementation (shuffle) and a JSON Facet implementation (push down)
- Multi-dimensional, high cardinality aggregations are possible with the map/reduce implementation

```
curl --data-urlencode 'stmt=SELECT to, count(*) FROM collection4 GROUP BY to ORDER BY count(*) desc LIMIT 10'
   http://localhost:8983/solr/collection4/sql?aggregationMode=facet
```

Below is sample result set:


```
{"result-set":{"docs":[
   {"count(*)":9158,"to":"pete.davis@enron.com"},
   {"count(*)":6244,"to":"tana.jones@enron.com"},
   {"count(*)":5874,"to":"jeff.dasovich@enron.com"},
   {"count(*)":5867,"to":"sara.shackleton@enron.com"},
   {"count(*)":5595,"to":"steven.kean@enron.com"},
   {"count(*)":4904,"to":"vkaminski@aol.com"},
   {"count(*)":4622,"to":"mark.taylor@enron.com"},
   {"count(*)":3819,"to":"kay.mann@enron.com"},
   {"count(*)":3678,"to":"richard.shapiro@enron.com"},
   {"count(*)":3653,"to":"kate.symes@enron.com"},
   {"EOF":"true","RESPONSE_TIME":10}]}
}
```


JDBC

- Part of SolrJ
- SolrCloud Aware Load Balancing
- Connection has ‘aggregationMode’ parameter that can switch between map_reduce or facet
- jdbc:solr://SOLR_ZK_CONNECTION_STRING?collection=COLLECTION_NAME&aggregationMode=facet

Inside Parallel SQL

Solr's Parallel Computing Framework

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

Streaming API

- Java API for parallel computation
- Real-time Map/Reduce and Parallel Relational Algebra
- Search results are streams of tuples (TupleStream)
- Transformed in parallel by Decorator streams
- Transformations include group by, rollup, union, intersection, complement, joins
- org.apache.solr.client.solrj.io.*

Streaming API

- Streaming Transformation

  Operations that transform the underlying streams e.g. unique, group by, rollup, union, intersection, complement, join etc

- Streaming Aggregation

  Operations that gather metrics and compute aggregates e.g. sum, count, average, min, max etc

Streaming Expressions

- String Query Language and Serialisation format for the Streaming API
- Streaming expressions compile to TupleStream
- TupleStream serialise to Streaming Expressions
- Human friendly syntax for Streaming API accessible to non-Java folks as well
- Can be used directly via HTTP to SolrJ

Streaming Expressions

```
curl --data-urlencode 'expr=search(enron_emails,
                                   q="from:1800flowers*",
                                   fl="from, to",
                                   sort="from asc",
                                   qt="/export")' http://localhost:8983/solr/enron_emails/stream
```


```
{"result-set":{"docs":[
   {"from":"1800flowers.133139412@s2u2.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers.93690065@s2u2.com","to":"jtholt@ect.enron.com"},
   {"from":"1800flowers.96749439@s2u2.com","to":"alewis@enron.com"},
   {"from":"1800flowers@1800flowers.flonetwork.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@1800flowers.flonetwork.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@1800flowers.flonetwork.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@1800flowers.flonetwork.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@1800flowers.flonetwork.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@shop2u.com","to":"ebass@enron.com"},
   {"from":"1800flowers@shop2u.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@shop2u.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@shop2u.com","to":"lcampbel@enron.com"},
   {"from":"1800flowers@shop2u.com","to":"ebass@enron.com"},
   {"from":"1800flowers@shop2u.com","to":"ebass@enron.com"},
   {"EOF":true,"RESPONSE_TIME":33}]}
}
```

Streaming Expressions

- Stream Sources

  The origin of a TupleStream

  search, jdbc, facet, stats, topic

- Stream Decorators

  Wrap other stream functions and perform operations on the stream

  complement, hashJoin, innerJoin, merge, intersect, top, unique

- Many streams can be paralleled across worker collections

Shuffling

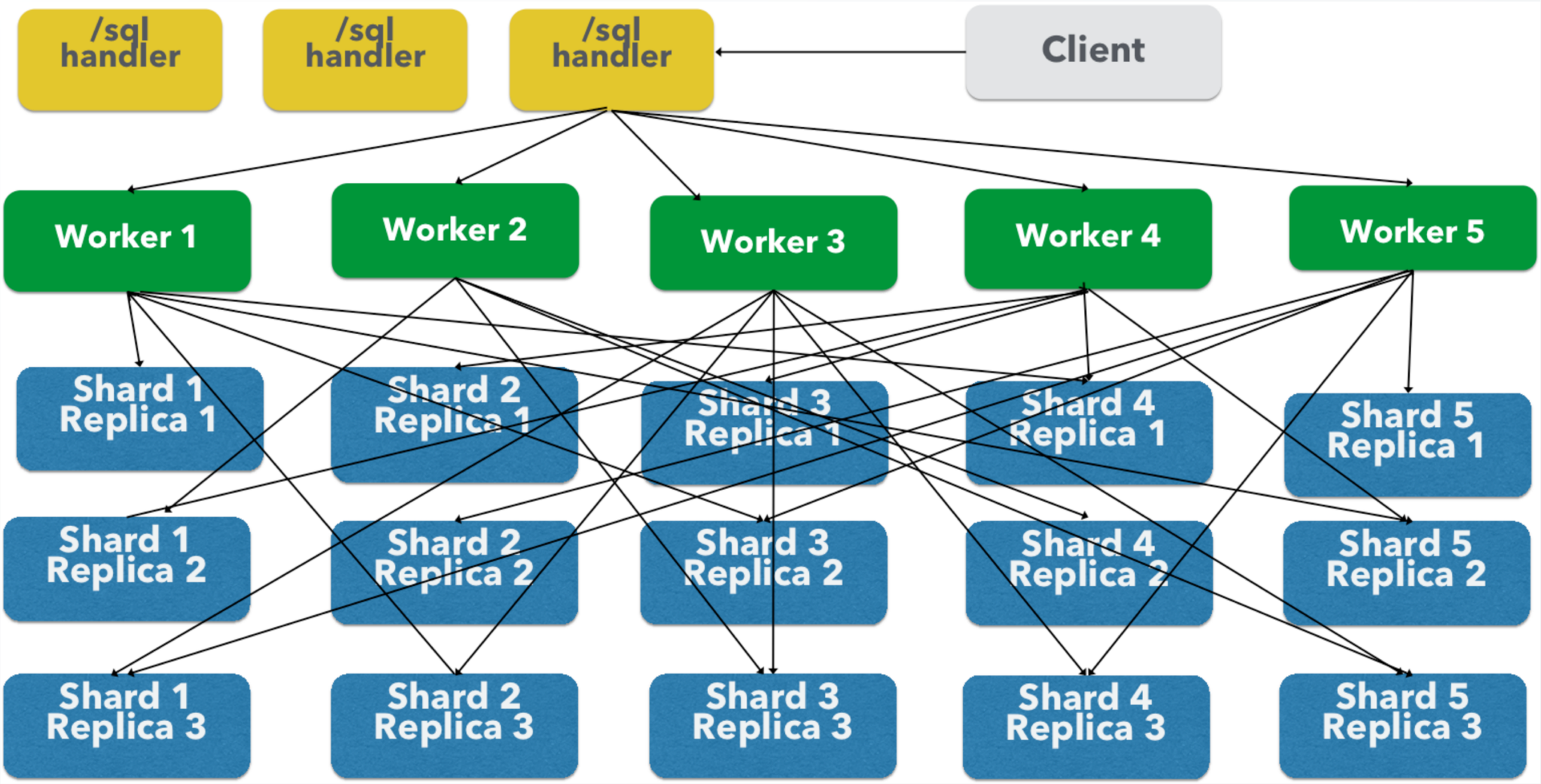
- Shuffling is pushed down to Solr
- Sorting is done by /export handler which stream-sorts entire result sets
- Partitioning is done by HashQParserPlugin which is a filter that partitions on arbitrary fields
- Tuples (search results) start streaming instantly to worker nodes never requiring a spill to the disk.
- All replicas shuffle in parallel for the same query which allows for massively parallel IO and huge throughputs.

Worker collections

- Regular SolrCloud collections
- Perform streaming aggregations using the Streaming API
- Receive shuffled streams from the replicas
- Over an HTTP endpoint: /stream
- May be empty or created just-in-time for specific analytical queries or have data as any regular SolrCloud collection
- The goal is to separate processing from data if necessary

Parallel SQL

- The Presto parser compiles SQL to a TupleStream
- TupleStream is serialised to a Streaming Expression and sent over the wire to worker nodes
- Worker nodes convert the Streaming Expression back into a TupleStream
- Worker nodes open() and read() the TupleStream in parallel


/sql handler
/sql handler
/sql handler
Client
Worker 1
Worker 2
Worker 3
Worker 4
Worker 5
Shard 1 Replica 1
Shard 2 Replica 1
Shard 3 Replica 1
Shard 4 Replica 1
Shard 5 Replica 1
Shard 1 Replica 2
Shard 2 Replica 2
Shard 3 Replica 2
Shard 4 Replica 2
Shard 5 Replica 2
Shard 1 Replica 3
Shard 2 Replica 3
Shard 3 Replica 3
Shard 4 Replica 3
Shard 5 Replica 3

What’s next

Graph traversals via streaming expressions

- Shortest path
- Node walking/gathering
- Distributed Gremlin implementation

Machine learning models

- LogisticRegressionQuery
- LogitStream
- More to come

Take actions based on AI driven alerts

- DaemonStreams
- AlertStream
- ModelStream

More, more, more!

- UpdateStream
- Publish-subscribe
- Calcite integration
- Better JDBC support

References

- Joel Bernstein's Blog — http://joelsolr.blogspot.in/
- https://cwiki.apache.org/confluence/display/solr/Streaming+Expressions
- https://cwiki.apache.org/confluence/display/solr/Parallel+SQL+Interface
- Parallel SQL by Joel Bernstein — https://www.youtube.com/watch?v=baWQfHWozXc
- Streaming Aggregations by Erick Erickson — https://www.youtube.com/watch?v=n5SYlw0vSFw

Lucidworks

Thank you

shalin@apache.org
@shalinmangar