Jaql: Querying JSON data on Hadoop

Kevin Beyer

Research Staff Member
IBM Almaden Research Center

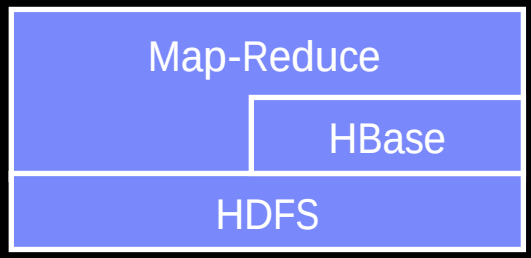
In collaboration with Vuk Ercegovac, Ning Li, Jun Rao, Eugene Shekita

Outline

- Overview of Hadoop
- JSON
- Jaql query language

The Hadoop Stack

- Components:



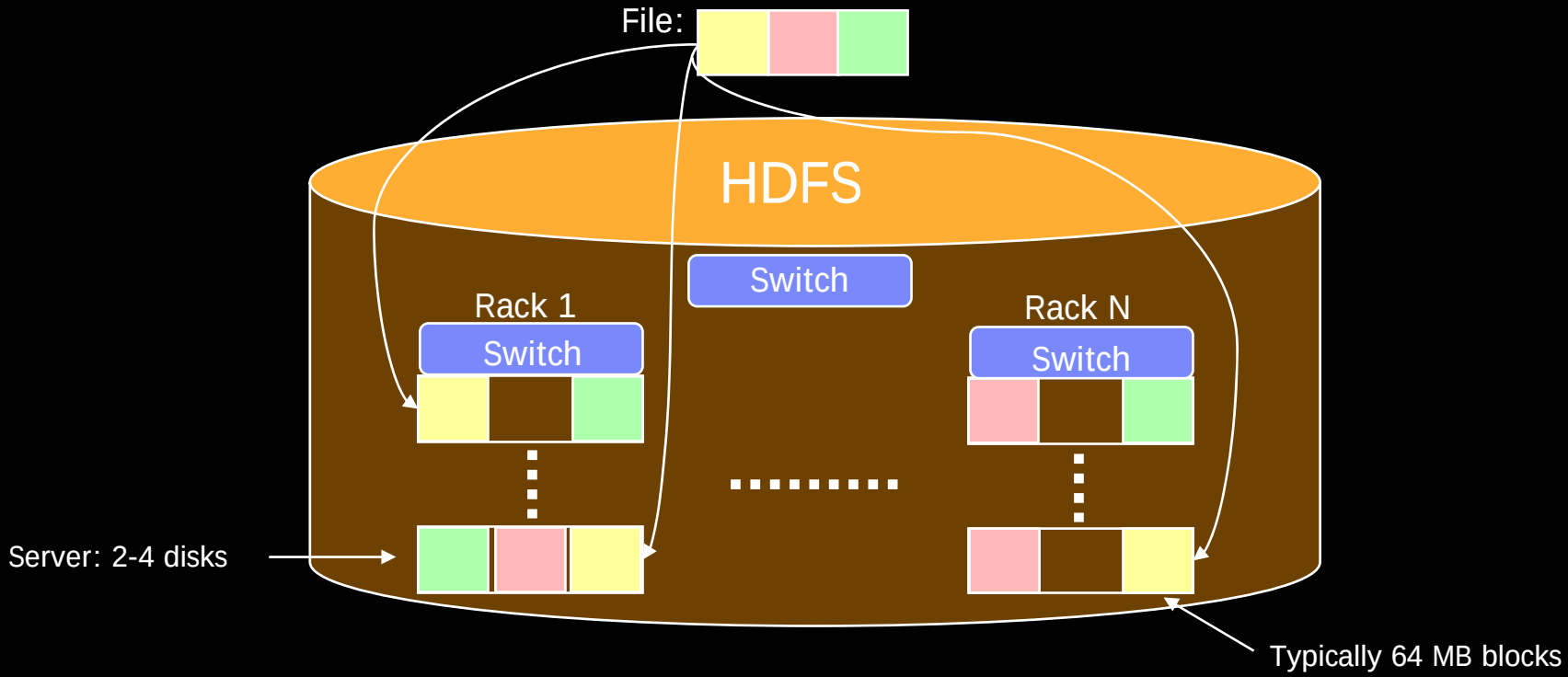
Parallel batch processing

Simple distributed database

Distributed file system

- Horizontal features:
  - Used at large scale (e.g., 10,000 cores at Yahoo)
  - Elastic (w/out data re-org)
  - Fault tolerant (getting there…)
  - Easy to administer
- Non-features:
  - No data model or types in HDFS or HBase
  - No indexing
  - No query language

HDFS Overview



- Single file-system stored on direct-attached disks of commodity servers
- Replicate file blocks for failures
- Simplified file system interface– not Posix
  - Designed for large, sequential reads

HBase Overview

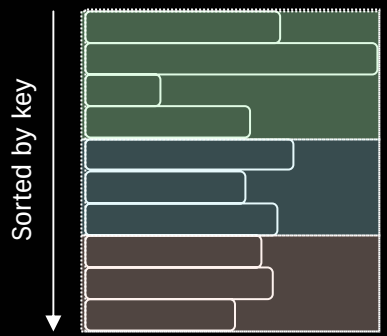
key | column name | column value

No schema, no types

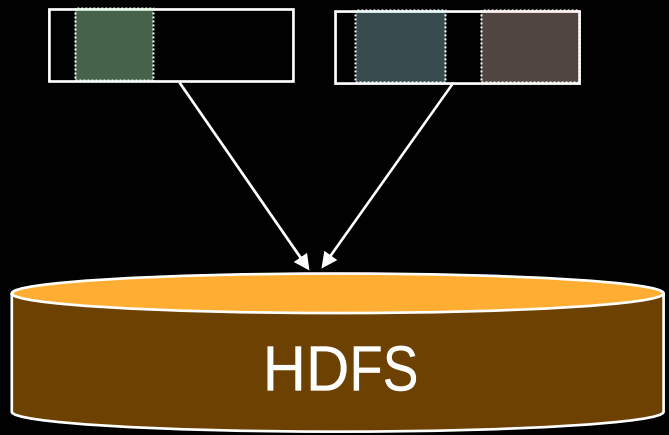
p127532	itemType: “car”	make: “VW”	doors: 2	…
p187842	itemType: “apartment”	rooms: 3	rent: 1200	location: “45E, 32N” …

…

Logical view of table



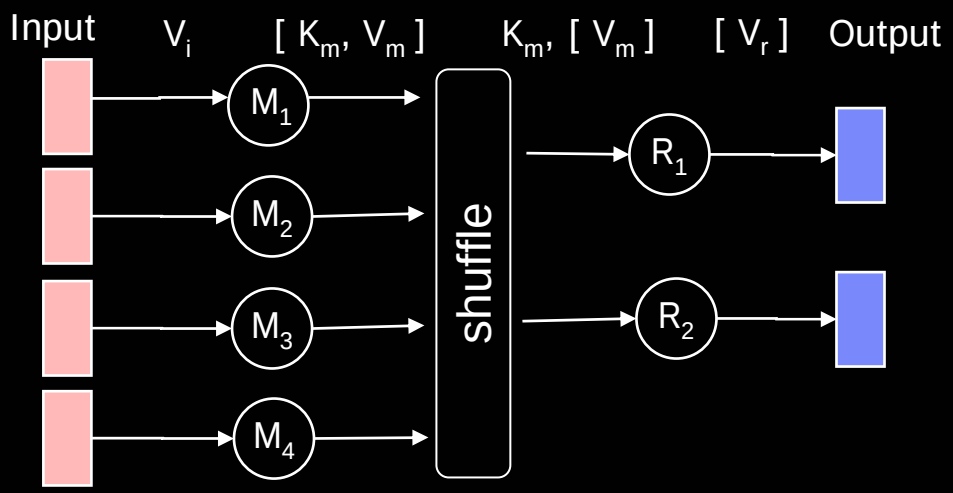
Physical view of table



Redundancy through HDFS replication

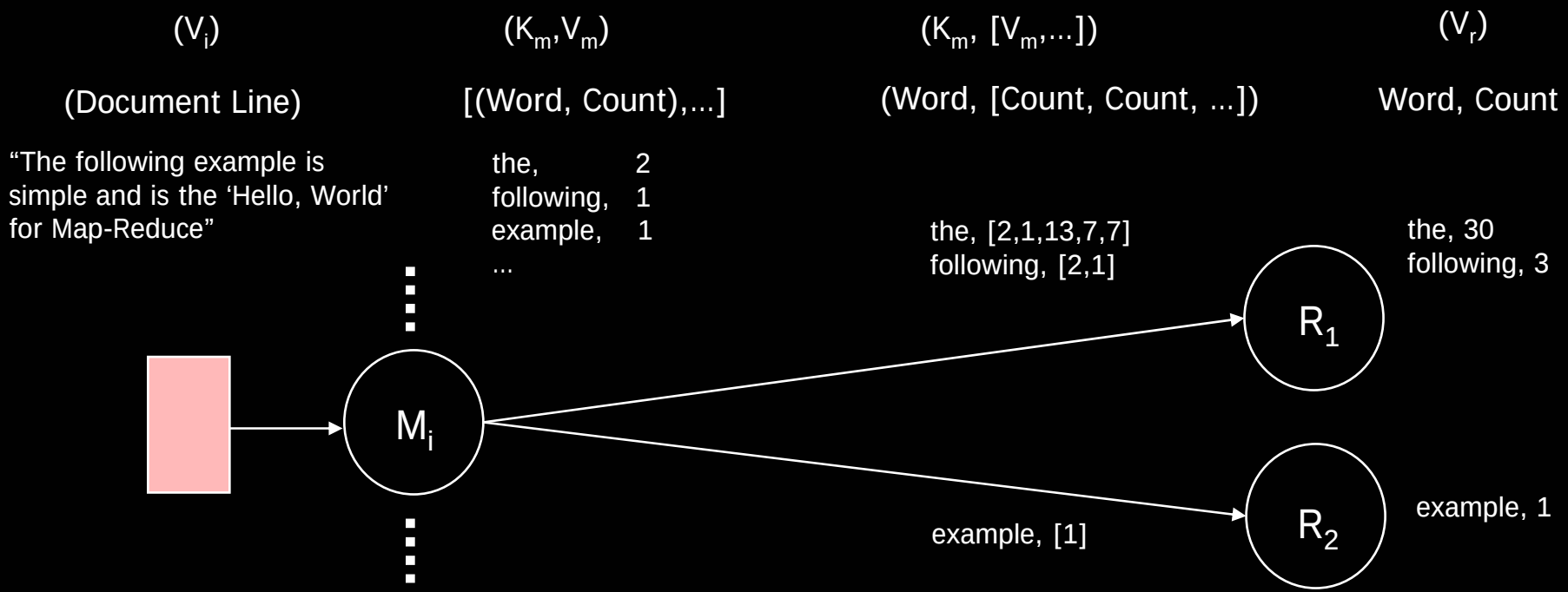
- Column values
  - Are versioned
  - Stored vertically in HDFS: <key, column, timestamp, value>

Map-Reduce Overview



- Programmer focus:
  - Map: $V_i \rightarrow [\ K_m, V_m\ ]$
  - Reduce: K_m, [V_m] $\rightarrow$ V_r
- System provides:
  - Parallelism
  - Fault tolerance
  - Key partitioning (shuffle)
  - Synchronization
  - Map task reads local block

Example: Counting Words



- Aggregate locally when possible (combine step)

Outline

- Overview of Hadoop
- JSON
- Jaql query language

What is JSON?

- JSON == Java Script Object Notation
- BNF (from www.json.org):

value ::= record | array | atom

record ::= { (string : value)* }

array ::= [(value)*]

atom ::= string | number | boolean | null

JSON Example

[] == array, {} == record or object, xxx: == field name

```
[
  { publisher: 'Scholastic', author: 'J. K. Rowling', title: 'Deathly Hallows', year: 2007 },

  { publisher: 'Scholastic', author: 'J. K. Rowling', title: 'Chamber of Secrets', year: 1999,
    reviews: [
      { rating: 10, user: 'joe', review: ‘The best ...’ },
      { rating: 6, user: 'mary', review: ‘Average ...’ }]},

  { publisher: 'Scholastic', author: 'J. K. Rowling', title: 'Sorcerers Stone', year: 1998},

  { publisher: 'Scholastic', author: 'R. L. Stine', title: 'Monster Blood IV', year: 1997,
    reviews: [
      { rating: 8, user: 'rob', review: 'High on my list...‘ },
      { rating: 2, user: 'mike', review: 'Not worth the paper ...' }]},

  { publisher: 'Grosset', author: 'Carolyn Keene', title: 'The Secret of Kane', year: 1930 }
]
```

Why JSON?

- Need nested, self-describing data
  - Data is typed, without requiring a schema
  - Support data that vary or evolve over time
- Standard
  - Wide-spread Web 2.0 adoption
  - Bindings available for many programming languages
- Not XML
  - XML data is untyped without schema validation
  - XML was designed for document mark-up, not data
- Easy integration in most programming languages
  - JSON is a subset of Javascript, Python, Ruby, Groovy, …

Outline

- Overview of Hadoop
- JSON
- Jaql query language

Jaql: A JSON Query Language

- Designed for JSON data
  - With additional atomic types: e.g., dateTime, binary
- Designed for many environments
  - Massive-scale cloud computing
    - Rewrite queries to use Map-Reduce
  - Micro-scale embedded in browser
- Designed for extensibility
  - Read / write data from any source into JSON view of data
  - Add new functions
- Functional query language
  - Few side-effects: e.g., writing to a file
  - Functions are data
- Draw on other languages
  - SQL, XQuery, PigLatin, JavaScript, Lisp, Python …

Jaql using Map

```
// Query: Find the authors and titles of books that have received a review.
$reviewed = for $b in hdfsRead('books')
            where exists( $b.reviews )
            return { $b.author, $b.title };

hdfsWrite( 'reviewedBooks', $reviewed );
```

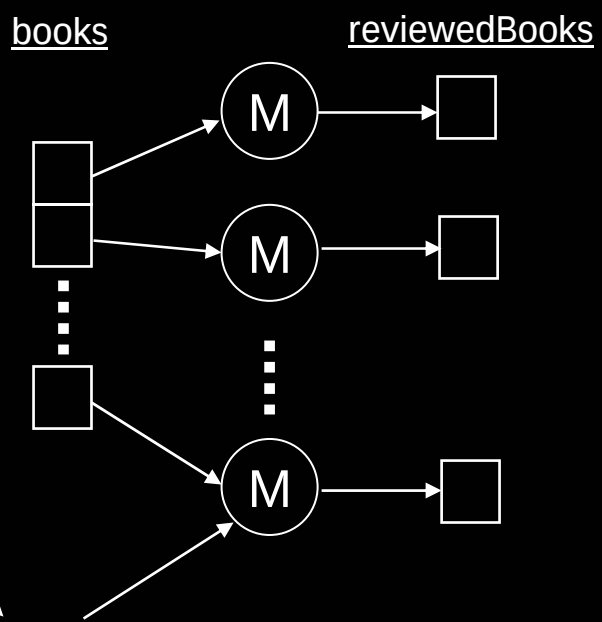
Rewrite Engine

Turn query into plan

Plan is another query!

```
// Query: equivalent map-reduce job in Jaql
mapReduce({
  input  : {type: 'hdfs', location: 'books' },
  map    : fn($b) {
     if exists($b.reviews) then
           [[ null, {$b.author, $b.title } ]]},
  output : {type: 'hdfs', location: 'reviewedBooks'}})
```

Map (book $b) -> {$b.author, $b.title}



Jaql's use of function as data -> evaluate "fn" in Map task

I/O Extensibility

- I/O layer abstracts details of data location + format
- Examples of data stores:
  - HDFS, HBase, Amazon's S3, local FS, HTTP request, JDBC call
- Examples of data formats:
  - JSON text, CSV, XML
  - Default format is JSON binary
- Simple to extend Jaql with new data stores and formats

I/O Extensibility Example

- Example: return purchase prices per book
  - Books stored in HBase
  - Purchases stored in HDFS
  - Output to a CSV file for graphing

Co-group: “outer equi-join”

Use multiple InputFormats

```
// Query: Group Books and Purchases to return book titles w/associated purchase prices
$result = group $b in hbaseRead('books')  by $bid = $b.key into $books,
          $p in hdfsRead('purchases') by $bid = $p.bid into $purchases
          return { bid: $bid, title: $books[0].title, prices: $purchases[*].price };

// Write the result to a local CSV file
hdfsWrite('bookPrices', { converter: 'CSVWriter' }, $result}
```

User defined format

```
{bid: 123, books: [{title: 'Death Hallows'}], purchases: [{bid: 123, price: 6.50,...},
                                                          {bid: 123, price: 3.43,...}, ...]}
{bid: 123, title: 'Death Hallows', prices: [6.50, 3.43] }

{bid: 789, books: [{title: 'Chamber of Secrets'}], purchases: [{bid: 789, price: 10.99,...},
                                                               {bid: 789, price: 6.75,...}, ...]}
{bid: 789, title: 'Chamber of Secrets', prices: [10.99, 6.75] }
```

Jaql I/O Extensibility using MapReduce

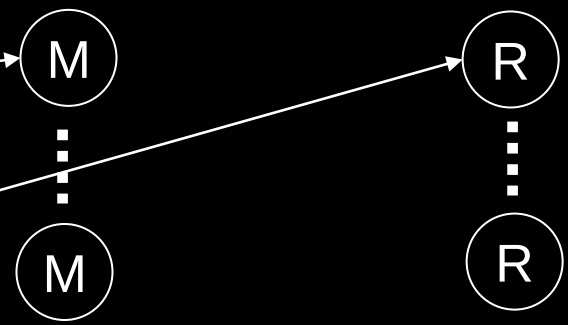
```
// Query: Group Books and Purchases to return book titles w/associated purchase prices
$result = group $b in hbaseRead('books')    by $bid = $b.key into $books,
                $p in hdfsRead('purchases') by $bid = $p.bid into $purchases
          return { id: $bid, title: $books[0].title, prices: $purchases[*].price };

// Write the result to a local CSV file
hdfsWrite( 'bookPrices', { converter: 'CSVWriter' }, $result } );
```

Rewrite Engine

```
// Query: equivalent map-reduce job in Jaql
mapReduce({
  input  : [{type: 'hbase', location: 'books' },
            {type: 'hdfs',  location: 'purchases'}],
  map    : [ fn($b) { [[ $b.key, $b ]] },
             fn($p) { [[ $p.bid,  $p ]] } ],
  reduce : fn($bid, $books, $purchases) {
             [ { id: $bid, title: $books[0].title,
                 prices: $purchases[*].price } ] },
  output : { type: 'hdfs', location: 'bookPrices',
             options: { converter: 'CSVWriter' }}})
```

Map (book $b) -> [$b.key, $b]
Map (purchases $p) -> [$p.bid, $p]
- Partition & sort by $bid

Reduce ($bid, $books, $purchases)
Extract id, title, prices



Expression Extensibility Example

- Example: segment books by their reviews' sentiment
  - Extract sentiment [0 = awful, 9 = best seller!] from each book
  - Return list of books per sentiment score

Extend Jaql with user defined expression

```
// Query: analyze book reviews
$scoredBooks = for $b in hbaseRead('books')
               return { $b.title, score: extractSentiment( $b.reviews ) };

// Query: aggregate according to sentiment score
$sentiments = group $s in $scoredBooks by $score = $s.score into $books
              return { score: $score, books: $books };

// Write the result
hdfsWrite( 'sentimentReport', $sentiment);
```

- Why user defined extension?
  - 3rd party libraries
  - Better expressed using a programming language
- Currently support Java, working on additional languages

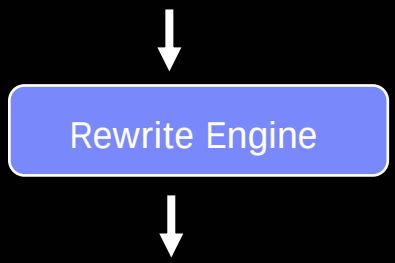
Aggregation Example

- Example: compute the stddev of sentiment per region
  - Join books and purchases for geographic region information
  - Group books by geographic region
  - Calculate standard deviation of book sentiments per region

```
// Query: analyze book reviews
$scoredBooks = for $b in hbaseRead('books')
               return { $b.id, score: extractSentiment($b.reviews) };

// Query: join scoredBooks with purchases
$bookPurchases = join $s in $scoredBooks on $s.id,
                      $p in hadoopRead('purchases') on $p.bid
                 return { $s.id, $s.score, $p.region };

// Query: aggregate by region
$regionStddev = group $bp in $bookPurchases by $r = $bp.region into $books
                return { region: $r, stddev: stddev($books[*].score) };

// Write the result
hdfsWrite('sentimentReport', $regionStddev);
```

Aggregation Example using Map-Reduce (1)

```
// Query: aggregate by region
$regionStddev = group $bp in $bookPurchases by $r = $bp.region into $books
                  return { region: $r, stddev: stddev( $books[*].score ) };

// Write the result to a local CSV file
hdfsWrite( 'sentimentReport', $regionStddev );
```




```
// Query: equivalent map-reduce job in Jaql
mapReduce({
  input  : [ ... ],
  map    : fn($bp) { [[ $bp.region, $bp ]] },
  reduce : fn($bid, $books) {
             [{ region: $r, stddev: stddev($books[*].score) }] },
  output : { type: 'hdfs', location: 'sentimentReport' } })
```

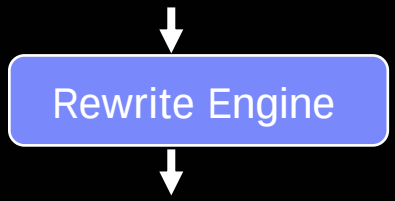
Standard deviation computed over large regions!

Distributive Aggregates

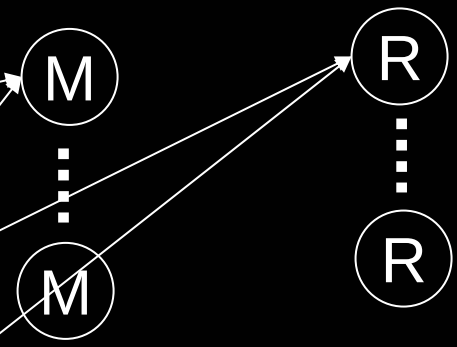
- Standard deviation is distributive
  - Final result can be computed from partial aggregates
- Map-Reduce can compute partial aggregates at Mapper
  - Map->Combine->Reduce
- Jaql's interface for distributive aggregates (for stddev):
  - Init($score):
    - { n: 1, s: $score, s2: $score*$score }
  - Combine($a, $b):
    - { n: $a.n + $b.n, s: $a.s + $b.s, s2: $a.s2 + $b.s2 }
  - Final($p):
    - sqrt($p.s2/$p.n – ($p.s/$p.n) * ($p.s/$p.n))

Aggregation Example using MapReduce (2)

```
// Query: aggregate by region
$regionStddev = group $bp in $bookPurchases by $r = $bp.region into $books
                        return {region: $r, stddev: stddev($books[*].score)};

// Write the result to a local CSV file
hdfsWrite('sentimentReport', $regionStddev);
```




```
// Query: equivalent map-reduce job in Jaql
mrAggregate({
input: { type: 'hdfs', location: 'books' },
init: fn ($bp) {
        [ $bp.region, { n: 1, s: $bp.score,
                        s2: $bp.score*$bp.score } ]},
combine: fn ($a, $b) {
           { n: $a.n + $b.n, s: $a.s + $b.s, s2: $a.s2 + $b.s2 }},
final: fn ($r, $p) {
         [{ region: $r,
            stddev: sqrt($p.s2/$p.n – ($p.s/$p.n)*($p.s/$p.n)) }]},
output: { type: 'hdfs', location: 'sentimentReport' }})
```



Related Work

- SQL, XQuery
- Sawzall (Google)
  - Wrap Map in a scripting language + library of Reducers
  - Proprietary and not a query language
- Pig (Yahoo)
  - Own data model vs. Jaql designed for JSON
  - Designed for Yahoo's data– no types, not fully composable
- Hive (Facebook)
  - Data warehouse catalog + SQL-like language
- DryadLinq (Microsoft)
  - Dryad: DAG of compute vertices and communication edges
  - Linq: embed data access in the programming language stack
- Groovy for Hadoop

Research Topics

- Usability
  - Additional Jaql features
  - Integration with programming languages
- Data model:
  - How much do we pay for dynamic typing?
  - How to take advantage of schema information?
- Optimization:
  - Indexing
  - Join strategies
    - Incorporate basic costs
  - More rewrites
  - Incremental compilation
  - Exploit HBase
    - Filters can be pushed into HBase
    - Projections have implied predicate (r.x => x exists for record r)
  - Code generation

Summary

- Scale-out infrastructure for analytics
- Hadoop: popular, open source scale-out infrastructure
- JSON provides a data model for Hadoop
  - Semi-structured and designed for data
- Jaql provides a query language for Hadoop
  - Rich analytics run in parallel
  - Extensible language and I/O layers

Questions?