Text Analysis with JAQL

Thilo Goetz, IBM R&D Germany

Hadoop users group meeting, Berlin, 9/29/2009
Text Analysis with JAQL

Thilo Goetz, IBM R&D Germany

Hadoop users group meeting, Berlin, 9/29/2009
JAQL

- JAQL: JSON Query Language
- Scripting language to manipulate JSON
- Runs on top of Hadoop
- Developed at IBM Almaden Research Center (Kevin Beyer, Vuk Ercegovac et al.)
  - I stole reuse their examples
- Open source at www.jaql.org
- JAQL syntax is under active development and should not be considered fixed yet
JSON example (sloppy)

```json
{
  text: "This is some sample text.",
  tokens: [
    {
      begin: 0,
      end: 4,
      pos: "DT",
      pos-confidence: 0.83
    },
    ...
  ]
}
```
JSON

- Javascript Object Notation
- Simple, textual format for object serialization (UTF-8)
- Semi-structured
  - Basic data structures (arrays, numbers, booleans, strings)
  - Records not typed
- Bindings available for many programming languages
JSON vs. XML

- JSON is for **data**, XML is for **documents**
  - XML has no support for arrays and primitive data types
  - JSON has no text mark-up
- JSON is simple and lightweight, XML is powerful and complex
- The core Java JSON API consists only of seven classes
- XML has a lot more tooling than JSON (such as XSLT)
Enter JAQL

- JAQL is a scripting language for manipulating JSON data
- Easily extend JAQL by writing your own Java functions
- JAQL expressions are compiled to Hadoop map/reduce jobs
JAQL pipes

```
[  
  { id: 12, name: "Joe Smith",  
    bday: date("1971-03-07"), zip: 94114 },  
  { id: 17, name: "Ann Jones",  
    bday: date("1973-02-04"), zip: 94110 },  
  { id: 19, name: "Alicia Fox",  
    bday: date("1975-04-20"), zip: 94114 }  
]

read(hdfs("users"))  
  -> filter $.zip == 94114  
  -> transform {$.id, fullname: $.name}  
  -> write(hdfs("inzip"));

[
  { id: 12, fullname: "Joe Smith" },
  { id: 19, fullname: "Alicia Fox" }
]```
Group

- Group objects by values into new objects
- \[ \text{[ "the", "man", "with", "the", "telescope"]} \]
  \>
  \text{group by $word = $}
  \text{into \{ $word, num: count($) \};}

- \[ \{ \text{word: "the", num: 2}, \text{word: "man", num: 1}\}, \ldots \]
More core language features

- **Join**: join two or more arrays on a common attribute
- **Sort**: sort arrays by values (may be complex objects)
- **Expand**: expand embedded arrays into individual values
- Also supports conditionals, loops and recursion
$books = [
    {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: '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 ...',
        discussion:
        [{user: 'ben', text: 'This is too harsh...'},
         {user: 'jill', text: 'I agree ...'}]}]]}
Expand and transform

$books
  -> expand $.reviews
  -> transform $.user;

[  "joe",
  "mary",
  "rob",
  "mike"
]
JAQL and Map/Reduce

- JAQL runs on Apache Hadoop
- JAQL queries are automatically translated into Hadoop M/R programs
- JAQL programmers are not required to know M/R details...
- ...but can get at them if they want to
JAQL M/R example

// Query 1. Return the publisher and title of each book.
read(hdfs("books"))
  -> transform {$.publisher, $.title};

// Explain Query 1: Jaql automatically rewrites the query into a map-only job
stRead(
  mapReduce(
    {input : { type: "hdfs", location: "books"},
     output : HadoopTemp(),
     map    : fn ($mapIn) [ [null,
                               { $mapIn.publisher, $mapIn.title }]]
  }));
Another M/R example

// Run a map/reduce job that counts the number of 
// objects for each 'x' value.
mapReduce(
  { input: {type: "hdfs", location: "sample.dat"},
    output: {type: "hdfs", location: "results.dat"},
    map:   fn($v) ( $v -> transform [$.x, 1] ),
    reduce: fn($x, $v)
      ( $v -> aggregate into {x: $x, num: count($)} )
  });
// define a function referenced by variable $myNewFn
$myNewFn = fn($a, $b) {
    $a + $b
};

// invoke $myNewFn
$myNewFn(1,2);

// result...
3
Java Functions

- Write java code using JAQL JSON APIs
- Create public eval() method(s)
- Add jar to JAQL classpath
- Register function with JAQL
- Call function like built-in JAQL functions
- JAQL uses reflection to find appropriate method
I/O

- Flexible I/O
  - Read/write from/to local file system, HDFS, and HBASE tables
  - Read/write new file formats with I/O adapters (Java)
Conclusion

- JAQL is a JSON query language that lets you manipulate your JSON data
- It runs on top of Hadoop, making M/R programming even easier
- It comes with flexible extensions mechanisms (functions, I/O)