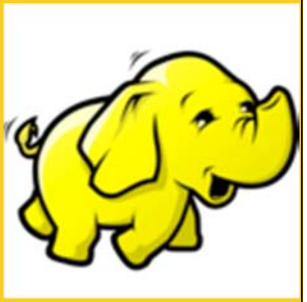
Advanced Java Client API

Advanced Topics



Agenda

- **Scan API**
- **Scan Caching**
- **Scan Batching**
- **Filters**

Scan Data Retrieval

- **Utilizes HBase's sequential storage model**
  - row ids are stored in sequence
- **Allows you to scan**
  - An entire table
  - Subset of a table by specifying start and/or stop key
  - Transfers limited amount of rows at a time from the server
    - 1 row at a time by default can be increased
- **You can stop the scan any time**
  - Evaluate at each row
  - Scans are similar to iterators

Scan Rows

1. **Construct HTable instance**
2. **Create and Initialize Scan**
3. **Retrieve ResultScanner from HTable**
4. **Scan through rows**
5. **Close ResultScanner**
6. **Close HTable**

** We are already familiar with HTable usage so let's focus on steps 2 through 5

2: Create and Initialize Scan

- **Scan class is a means to specify what you want to scan**
- **Scan is very similar to Get but allows you to scan through a range of keys**
  - Provide start and stop keys
  - Start key is inclusive while stop key is exclusive
  - If start row id is NOT provided then will scan from the beginning of the table
  - If stop row is NOT provided then will scan to the very end

2: Create and Initialize Scan

- **Construction options**
  - new Scan() - will scan through the entire table
  - new Scan(startRow) – begin scan at the provided row, scan to the end of the table
  - new Scan(startRow, stopRow) – begin scan at the provided startRow, stop scan when a row id is equal to or greater than to the provided stopRow
  - new Scan(startRow, filter) – begin scan at the provided row, scan to the end of the table, apply the provided filter

2: Create and Initialize Scan

- **Once Scan is constructed you can further narrow down (very similar to Get)**
  - scan.addFamily(family)
  - scan.addColumn(family, column)
  - scan.setTimeRange(minStamp, maxStamp)
  - scan.setMaxVersions(maxVersions)
  - scan.setFilter(filter) – to be covered later
- **For example:**

```
Scan scan = new Scan(toBytes(startRow), toBytes(stopRow));
scan.addColumn(toBytes("metrics"), toBytes("counter"));
scan.addFamily(toBytes("info"));
```

3: Retrieve ResultScanner

- **Retrieve a scanner from an existing HTable instance**

```
ResultScanner scanner = hTable.getScanner(scan);
```

4: Scan Through Rows

- **Use result scanner by calling**
  - Result next() throws IOException
    - Same Result class as in Get operation
  - Result[] next(int nbRows) throws IOException
    - Returns an array of Result object up to nbRows
    - Maybe less than nbRows
  - ResultScanner also implements an Iterable interface so we can do something like this

```
ResultScanner scanner = hTable.getScanner(scan);
for ( Result result : scanner){
  // do stuff with result
}
```

5: Close ResultScanner

- **Scanner holds to resources on the server**
  - As soon as you are done with the scanner call close()
  - Required to release all the resources
  - Always use in try/finally block

```
ResultScanner scanner = hTable.getScanner(scan);
try {
      // to stuff with scanner
} finally {
      scanner.close();
}
```

  - Most of the examples omit try/finally usage just to make them more readable

ScanExample.java

```
private static void scan(HTable hTable, String startRow,
      String stopRow) throws IOException {

  System.out.println("Scanning from " +
            "["+startRow+"] to ["+stopRow+"]");

  Scan scan = new Scan(toBytes(startRow), toBytes(stopRow));
  scan.addColumn(toBytes("metrics"), toBytes("counter"));

  ResultScanner scanner = hTable.getScanner(scan);
  for ( Result result : scanner){
        byte [] value = result.getValue(
                   toBytes("metrics"), toBytes("counter"));

        System.out.println("  " +
                   Bytes.toString(result.getRow()) + " => " +
                   Bytes.toString(value));
  }

  scanner.close();
}
```

ScanExample.java

```
public static void main(String[] args) throws IOException {
   Configuration conf = HBaseConfiguration.create();
   HTable hTable = new HTable(conf, "HBaseSamples");

   scan(hTable, "row-03", "row-05");
   scan(hTable, "row-10", "row-15");
   hTable.close();
}
```

ScanExample.java

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.ScanExample
..
..
Scanning from [row-03] to [row-05]
  row-03 => val2
  row-04 => val3
Scanning from [row-10] to [row-15]
  row-10 => val9
  row-11 => val10
  row-12 => val11
  row-13 => val12
  row-14 => val13
```

ResultScanner Lease

- **HBase protects itself from Scanners that may hang indefinitely by implementing lease-based mechanism**
- **Scanners are given a configured lease**
  - If they don't report within the lease time HBase will consider client to be dead
  - The scanner will be expired on the server side and it will not be usable
  - Default lease is 60 seconds
  - To change the lease modify hdfs-site.xml

```
<property>
 <name>hbase.regionserver.lease.period</name>
  <value>120000</value>
</property>
```

  - The same property is used for lease-based mechanism for both locks and scanners
    - Make sure the value works well for both

Scanner Caching

- **By default next() call equals to RPC (Remote Procedure Call) per row**
  - Even in case of next(int rows)

```
int numOfRPCs = 0;
for ( Result result : scanner){
      numOfRPCs++;
}
System.out.println("Remote Calls: " + numOfRPCs);
```

- **Results in a bad performance for small cells**
- **Use Scanner Caching to fetch more than a single row per RPC**

Scanner Caching

- **Three Levels of control**
  - HBase Cluster: change for ALL
  - HTable Instance: configure caching per table instance, will affect all the scans created for this table
  - ResultScanner Instance: configure caching per scan instance, will only affect the configured scan
- **Can configure at multiple levels if you require the precision**
  - Ex: Certain tables may have really big cells then lower scanning size

1: Configure Scanner Caching per HBase Cluster

- **Edit <hbase_home>/conf/hbase-site.xml**

```
<property>
  <name>hbase.client.scanner.caching</name>
  <value>20</value>
</property>
```

- **Restart the cluster to pick up the change**
- **Changes caching to 10 for ALL scans**
  - Can still override per HTable or Scan instance

2: Configure Scanner Caching per HTable Instance

- **Call hTable.setScannerCaching(10) to change caching per HTable instance**
- **Will override caching configure for the entire HBase cluster**
- **Will affect caching for every scan open from this HTable instance**
  - Can be overridden at scan level

3: Configure Scanner Caching per ResultScanner Instance

- **Set caching on Scan instance and use it to retrieve the scanner**

```
scan.setCaching(10);
ResultScanner scanner = hTable.getScanner(scan);
```

- **Will only apply to this scanner**
- **Will override cluster and table based caching configurations**

Scanner Caching Considerations

- **Balance between low number of RPC and memory usage**
  - Consider the size of the data retrieved (cell size)
  - Consider available memory on the client and Region Server
- **Setting higher caching number would usually improve performance**
- **Setting caching too high may have negative effect**
  - Takes longer for each remote call to transfer data
  - Run out of client's or Region Server's heap space and cause OutOfMemoryError

ScanCachingExample.java

```
private static void printResults(HTable hTable, Scan scan)
                             throws IOException {

  System.out.println("\nCaching table=" +
       hTable.getScannerCaching() +
       ", scanner=" + scan.getCaching());

  ResultScanner scanner = hTable.getScanner(scan);
  for ( Result result : scanner){

      byte [] value = result.getValue(
             toBytes("metrics"), toBytes("counter"));
      System.out.println("  " +
             Bytes.toString(result.getRow()) + " => " +
             Bytes.toString(value));
  }

  scanner.close();
}
```

Print caching attributes

Scan through the results

ScanCachingExample.java

```
public static void main(String[] args) throws IOException {

    Configuration conf = HBaseConfiguration.create();
    HTable hTable = new HTable(conf, "HBaseSamples");

    Scan scan = new Scan();
    scan.addColumn(toBytes("metrics"), toBytes("counter"));
    printResults(hTable, scan);

    hTable.setScannerCaching(5);
    printResults(hTable, scan);

    scan.setCaching(10);
    printResults(hTable, scan);

    hTable.close();
}
```

Caching is not set will use default

Set scanning on table level, overrides default

Set caching on Scan level Overrides default and table

ScanCachingExample.java Output

```
$yarn jar $PLAY_AREA/HadoopSamples.jar hbase.ScanCachingExample

Caching table=1, scanner=-1
  row-01 => val0
  row-02 => val1
  ...
  row-16 => val15

Caching table=5, scanner=-1
  row-01 => val0
  row-02 => val1
  ...
  row-16 => val15

Caching table=5, scanner=10
  row-01 => val0
  row-02 => val1
  ...
  row-16 => val15
```

Table defaulted to the setting of 1
Scanner caching is not set (-1)
Pulls 1 row per RPC

Updated on table level to 5
Overrides default
Pulls 5 rows per RPC

Updated on the scan level to 10
Overrides default and table level
Pulls 10 rows per RPC

Scanner Batching

- A single row with lots of columns may not fit memory
- HBase Batching allows you to page through columns on per row basis
- Limits the number of columns retrieved from each ResultScanner.next() RPC
  - Will not get multiple results
- Set the batch on Scan instance
  - No option on per table or cluster basis

```
Scan scan = new Scan();
scan.setBatch(10);
```

ScanBatchingExample.java

```
public static void main(String[] args) throws IOException {
      Configuration conf = HBaseConfiguration.create();
      HTable hTable = new HTable(conf, "HBaseSamples");

      Scan scan = new Scan();
      scan.addFamily(toBytes("columns"));
      printResults(hTable, scan);
```

Print result with default batch (loads entire row)

Print result with batch=2

```
   scan.setBatch(2);
      printResults(hTable, scan);

      hTable.close();
}
```

ScanBatchingExample.java

```
private static void printResults(HTable hTable, Scan scan)
               throws IOException {
  System.out.println("\n------------------");
  System.out.println("Batch=" + scan.getBatch());
```

Display batch size
Of this Scan instance

```
  ResultScanner scanner = hTable.getScanner(scan);
  for ( Result result : scanner){
    System.out.println("Result: ");
```

For each result print
all the cells/KeyValues

```
    for ( KeyValue keyVal : result.list()){
      System.out.println("  " +
         Bytes.toString(keyVal.getFamily()) + ":" +
         Bytes.toString(keyVal.getQualifier()) + " => " +
         Bytes.toString(keyVal.getValue()));
    }
  }
  scanner.close();
}
```

ScanBatchingExample.java Output

```
------------------
Batch=-1
Result:
  columns:col1 => colRow1Val1
  columns:col2 => colRow1Val2
  columns:col3 => colRow1Val3
  columns:col4 => colRow1Val4
Result:
  columns:col1 => colRow2Val1
  columns:col3 => colRow2Val2
  columns:col4 => colRow2Val3
------------------
Batch=2
Result:
  columns:col1 => colRow1Val1
  columns:col2 => colRow1Val2
Result:
  columns:col3 => colRow1Val3
  columns:col4 => colRow1Val4
Result:
  columns:col1 => colRow2Val1
  columns:col3 => colRow2Val2
Result:
  columns:col4 => colRow2Val3
```

Default batch load
entire row per Result
instance

Batching 2 columns
per Result instance

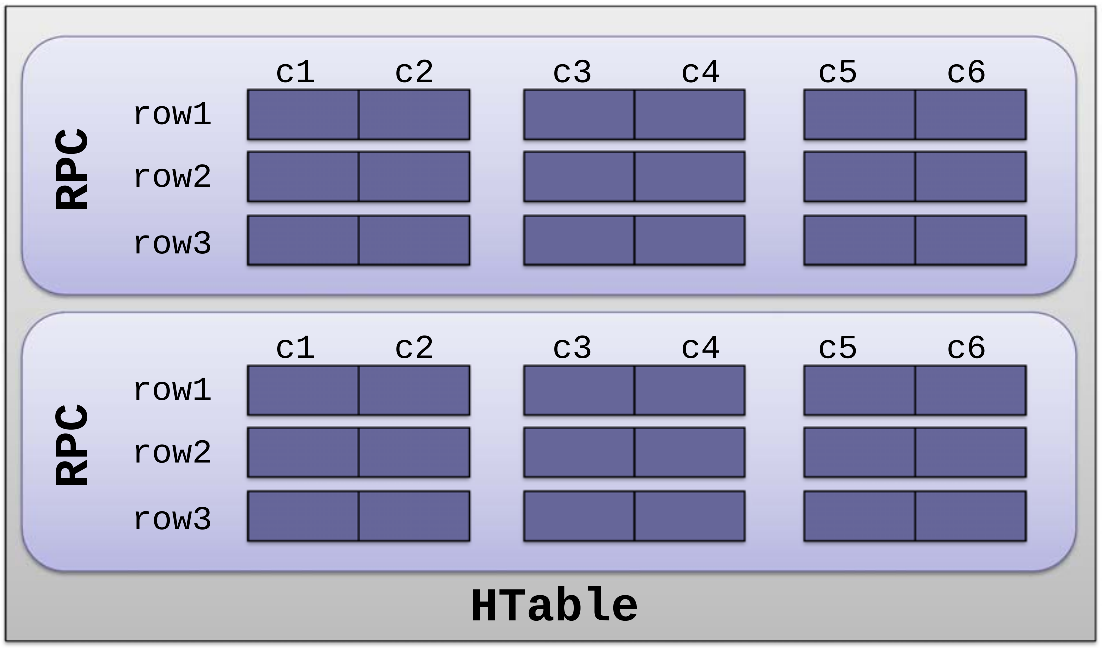
Caching and Batching

- **Caching and Batching can be combined when scanning a set of rows to balance**
  - Memory usage
  - # of RPCs
- **Batching will create multiple Result instances per row**
- **Caching specifies how many results to return per RPC**
- **To estimate Total # of RPCs**

(# of rows) * (columns per row)
/ (minimum between batch size and # of columns size)
/ (caching size)

Caching and Batching Example

Batch = 2 and Caching = 9



Source: Lars, George. HBase The Definitive Guide. O'Reilly Media. 2011

Filters

- **get() and scan() can limit the data retrieved/transferred back to the client**
  - via Column families, columns, timestamps, row ranges, etc...
- **Filters add further control to limit the data returned**
  - For example: select by key or values via regular expressions
  - Optionally added to Get and Scan parameter
- **Implemented by org.apache.hadoop.hbase.filter.Filter**
  - Use HBase’s provided concrete implementations
  - Can implement your own

Filter Usage

1. **Create/initialize an instance of a filter**
2. **Add it to Scan or Get instance**
3. **Use Scan or Get as before**

1: Create/Initialize an Instance of a Filter

- **There are a lot of filters provide by HBase**
  - ValueFilter, RowFilter, FamilyFilter, QuilifierFilter, etc...
  - 20+ today and the list is growing
- **For example ValueFilter lets you include columns that only have specific values**
  - Uses expression syntax

Comparison Operator

```
Scan scan = new Scan();
scan.setFilter(
        new ValueFilter(CompareOp.EQUAL, new SubstringComparator("3")));
```

Comparator

ValueFilterExample.java

```
public static void main(String[] args) throws IOException {
   Configuration conf = HBaseConfiguration.create();
   HTable hTable = new HTable(conf, "HBaseSamples");

   Scan scan = new Scan();
   scan.setFilter(
             new ValueFilter(CompareOp.EQUAL,
                          new SubstringComparator("3")));

   ResultScanner scanner = hTable.getScanner(scan);
   for ( Result result : scanner){
       byte [] value = result.getValue(
                    toBytes("metrics"), toBytes("counter"));
       System.out.println("  " +
                    Bytes.toString(result.getRow()) + " => " +
                    Bytes.toString(value));
   }
   scanner.close();
   hTable.close();
}
```

Only get cells whose value contains string “3”

ValueFilterExample.java Output

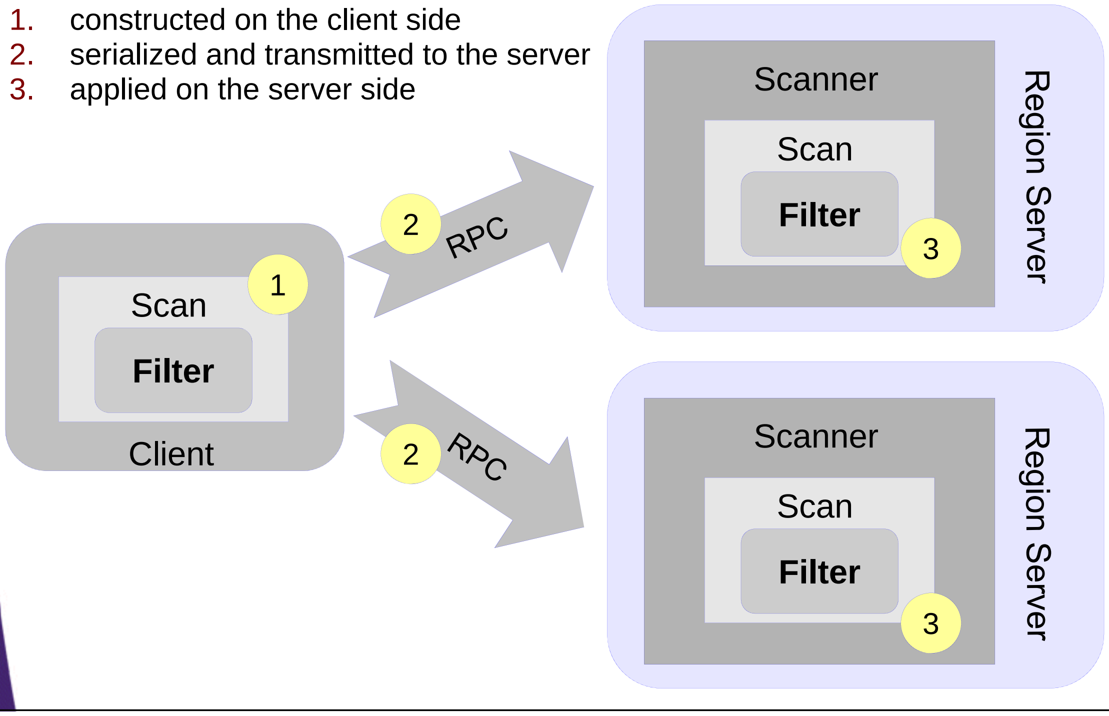
```
yarn jar $PLAY_AREA/HadoopSamples.jar hbase.ValueFilterExample
 row-04 => val3
 row-14 => val13
```

Filters

- **Filters are applied on the server side**
  - Reducing amount of data transmitted over the wire
  - Still involves scanning rows
    - For example, not as efficient using start/stop rows in the scan
- **Execution with filters**
  - constructed on the client side
  - serialized and transmitted to the server
  - executed on the server side
- **Must exist both on client's and server's CLASSPATH**

Execution of a Request with Filter(s)

1. constructed on the client side
2. serialized and transmitted to the server
3. applied on the server side



Sampling of HBase Provided Filters

Filter	Description from HBase API
ColumnPrefixFilter	This filter is used for selecting only those keys with columns that matches a particular prefix.
FilterList	Implementation of Filter that represents an ordered List of Filters
FirstKeyOnlyFilter	A filter that will only return the first KV from each row.
KeyOnlyFilter	A filter that will only return the key component of each KV
PrefixFilter	This filter is used for selecting only those keys with columns that matches a particular prefix.
QualifierFilter	This filter is used to filter based on the column qualifier.
RowFilter	This filter is used to filter based on the key
SkipFilter	A wrapper filter that filters an entire row if any of the KeyValue checks do not pass.
ValueFilter	This filter is used to filter based on column value.

To Apply Multiple Filters

1. **Create FilterList and specify operator**
   – Operator.MUST_PASS_ALL: value is only included if an only if all filters pass
   – Operator.MUST_PASS_ONE: value is returned if any of the specified filters pass
2. **Add filters to FilterList**
3. **Add it to Scan or Get instance**
4. **Use Scan or Get as before**

FilterListExample.java

```
Scan scan = new Scan();
FilterList filters = new
                    FilterList(Operator.MUST_PASS_ALL);
filters.addFilter(new KeyOnlyFilter());
filters.addFilter(new FirstKeyOnlyFilter());
scan.setFilter(filters);
```

Only load row ids by chaining KeyOnlyFilter and FirstKeyOnlyFilter

```
ResultScanner scanner = hTable.getScanner(scan);
for ( Result result : scanner){
       byte [] value = result.getValue(
              toBytes("metrics"), toBytes("counter"));
       System.out.println("  " +
              Bytes.toString(result.getRow()) + " => " +
              Bytes.toString(value));
}
scanner.close();
```

FilterListExample.java Output

```
$ yarn jar $PLAY_AREA/HadoopSamples.jar hbase.FilterListExample
  anotherRow => null
  row-01 =>
  row-02 =>
  row-03 =>
  row-04 =>
  row-05 =>
  row-06 =>
  row-07 =>
  row-08 =>
  row-09 =>
  row-10 =>
  row-11 =>
  row-12 =>
  row-13 =>
  row-14 =>
  row-15 =>
  row-16 =>
  row1 => null
```

Only row ids were retrieved because KeyOnlyFilter and FirstKeyOnlyFilter Were applied to the Scan request

Wrap-Up

Summary

- **We learned about**
  - Scan API
  - Scan Caching
  - Scan Batching
  - Filters

Questions?