(/contact)

(https://www.3pillarglobal.com)

WHAT WE DO (HTTPS://WWW.3PILLARGLOBAL.COM/WHAT-WE-DO) HOW WE DO IT (HTTPS://WWW.3PILLARGLOBAL.COM/HOW-WE-DO-IT)

WHO WE DO IT FOR (HTTPS://WWW.3PILLARGLOBAL.COM/WHO-WE-DO-IT-FOR) INSIGHTS (HTTPS://WWW.3PILLARGLOBAL.COM/INSIGHTS)

NEWS (HTTPS://WWW.3PILLARGLOBAL.COM/NEWS) ABOUT (HTTPS://WWW.3PILLARGLOBAL.COM/ABOUT)

CONTACT (HTTPS://WWW.3PILLARGLOBAL.COM/CONTACT) CAREERS (HTTPS://WWW.3PILLARGLOBAL.COM/ABOUT/CAREERS)

Home (https://www.3pillarglobal.com/) / Insights (https://www.3pillarglobal.com/insights) / Blogs (https://www.3pillarglobal.com/insights/blog-posts) / The How To Of HBase Coprocessors

THE HOW TO OF HBASE COPROCESSORS

What is HBase™? HBase is a column oriented non-relational big data database. HBase is a distributed, scalable, reliable, and versioned storage system capable of providing random read/write access in real-time. It was modeled after Google's Bigtable (http://research.google.com/archive/bigtable.html) and just as Bigtable is built on top of Google File System, in the same manner HBase is built on HDFS (Hadoop Distributed File System). This means it is tightly integrated with Hadoop and provides base classes for writing MapReduce jobs for HBase.

HBase is capable of hosting large tables with billions of rows and millions of columns with generally millisecond SLA (Service Level Agreement). One of the key features of HBase is Coprocessor, the premise of this blog post. I am assuming readers would be familiar with HBase and also with the various terms like Column Family, HFile, Region, and RegionServer. If not, an elaboration of each is provided at the conclusion of this blog.

What is Coprocessor? Simply stated, Coprocessor is a framework that provides an easy way to run your custom code on Region Server.

When working with any data store (like RDBMS or HBase) you fetch the data (in case of RDBMS you may use query and in case of HBase you use either Get or Scan). To fetch only relevant data you filter it (for RDBMS you put conditions in 'WHERE' clause and in HBase you use Filters). After fetching the desired data, you can perform your business computation on the data.

This scenario is close to ideal for "small data," like a few thousand rows with a bunch of columns. Now imagine a scenario where there are billions of rows and millions of columns and you want to perform some computation which requires all the data, like calculating average or sum. Even if you are interested in just a few columns, you still have to fetch all the rows. There are a few drawbacks in this approach as described below:

1. In this approach the data transfer (from data store to client side) will become the bottleneck, and the time required to complete the operation is limited by the rate at which data transfer is taking place.
2. It's not always possible to hold this much data in memory and perform computation.
3. Bandwidth is one of the most precious resources in any data center. Operations like this will severely impact the performance of your cluster.

4. Your client code is becoming thick as you are maintaining the code for calculating average or summation on client side. Not a major drawback when talking of severe issues like performance/bandwidth but still worth giving consideration.

In a scenario like this it's better to move the computation to the data itself; just like stored procedure (a better analogy is MapReduce model). Coprocessor helps you achieve this but you can do more than that. To give an idea of Coprocessor's capabilities, different people give different analogies. The three most famous analogies for Coprocessor present in the industry are:

1. Triggers and Stored Procedure: This is most common analogy that you will find for Coprocessor. (The official document uses this analogy). Observer Coprocessor (discussed below) is compared to triggers because like triggers they execute your custom code when certain event occurs (like Get or Put etc.). Similarly Endpoints Coprocessor (discussed below) is compared to the stored procedures and you can perform custom computation on data directly inside the region server.
2. MapReduce: As in MapReduce you move the computation to the data in the same way. Coprocessor executes your custom

computation directly on Region Servers, i.e. where data resides. That's why some people compare Coprocessor to small MapReduce jobs.
3. AOP: Some people compare it to Aspect Oriented Programming (AOP). As in AOP, you apply advice by intercepting the request then running some custom code (probably cross-cutting) and then forwarding the request on its path as if nothing happened (or even return it back). Similarly in Coprocessor you have this facility of intercepting the request and running custom code and then forwarding it on its path (or returning it).

Actually the Coprocessor derives its roots from Google™ Bigtable (http://research.google.com/archive/bigtable-osdi06.pdf)™ but it deviates from it largely in its design. Currently there are efforts going in the HBase community to bridge this gap. For more information see JIRA ticket HBASE-4047 (https://issues.apache.org/jira/browse/HBASE-4047).

In HBase, to implement a Coprocessor certain steps must be followed as described below:

1. Either your class should extend one of the Coprocessor classes (like BaseRegionObserver) or it should implement Coprocessor interfaces (like Coprocessor, CoprocessorService).
2. Load the Coprocessor: Currently there are two ways to load the Coprocessor. One is static (i.e. loading from configuration) and the other is dynamic (i.e. loading from table descriptor either through Java code or through 'hbase shell'). Both are discussed below in detail.
3. Finally your client-side code to call the Coprocessor. This is the easiest step, as HBase handles the Coprocessor transparently and you don't have to do much to call the Coprocessor.

Coprocessors are not designed to be used by the end users but by developers. Coprocessors are executed directly on region server; therefore a faulty/malicious code can bring your region server down. Currently there is no mechanism to prevent this, but there are efforts going on for this. For more, see JIRA ticketHBASE-4047 (https://issues.apache.org/jira/browse/HBASE-4047).

Coprocessor can be broadly divided into two categories – **Observer** and **Endpoint** – and each one is discussed separately:

1. Observer Coprocessor**:** As stated above, these are just like database triggers, i.e. they execute your custom code on the occurrence of certain events. If you prefer (or if you are from Java background) you can also visualize it like advice (before and after only). Coprocessors allow you to hook your custom code in two places during the lifecycle of the event. One is just before the occurrence of the event (just like before advice in AOP). For example, it will allow your custom code to run just before the 'Put' operation. All methods providing this feature will start with the prefix 'pre.' For example, if you want to your code to be executed before the put operation then you should override following method of RegionObserver class. We will walk through a working example after this boring introduction. ☺️

```
public void prePut (final ObserverContext e, final Put put, final WALEdit edit,final Durability (
}
```

Similarly, the Observer Coprocessor also provides hooks for your code to get executed just after the occurrence of the event (similar to after advice in AOP terminology). These methods will start with the prefix 'post.' For example, if you want your code to be executed after the 'Put' operation, you should override following method:

```
verContext e, final Put put, final WALEdit edit, final Durability durability) throws IOException { }
```

In short, the following conventions are followed:

Override preXXX() if you want your code to be executed before the occurrence of the event.

Override postXXX() if you want your code to be executed after the occurrence of the event.

A few use cases of the Observer Coprocessor are:

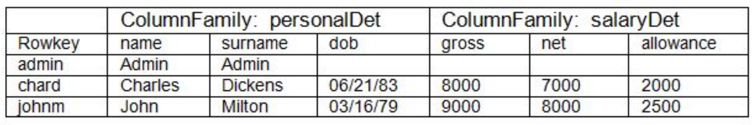
1. Security: Before performing any operation (like 'Get', 'Put') you can check for permission in the 'preXXX' methods.
2. Referential Integrity: Unlike traditional RDBMS, HBase doesn't have the concept of referential integrity (foreign key). Suppose for example you have a requirement that whenever you insert a record in 'users' table, a corresponding entry should also be created in 'user_daily_attendance' table. One way you could solve this is by using two 'Put' one for each table, this way you are throwing the responsibility (of the referential integrity) to the user. A better way is to use Coprocessor and overriding 'postPut' method in which you write the code to insert the record in 'user_daily_attendance' table. This way client code is more lean and clean.
3. Secondary Index: Coprocessor can be used to maintain secondary indexes. For more information please see SecondaryIndexing (http://wiki.apache.org/hadoop/Hbase/SecondaryIndexing).

Observer Coprocessor has following flavors:

1. **RegionObserver:** This Coprocessor provides the facility to hook your code when the events on region are triggered. Most common example include 'preGet' and 'postGet' for 'Get' operation and 'prePut' and 'postPut' for 'Put' operation.
2. **Region Server Observer:** Provides hook for the events related to the RegionServer, such as stopping the RegionServer and performing operations before or after merges, commits, or rollbacks.
3. **WAL Observer:** Provides hooks for WAL (Write-Ahead-Log) related operation. It has only two method 'preWALWrite()' and 'postWALWrite()'.
4. **Master Observer:** This observer provides hooks for DDL like operation, such as create, delete, modify table.

Example of Observer Coprocessor:

Table 1: 'users' table



	ColumnFamily: personalDet			ColumnFamily: salaryDet		
Rowkey	name	surname	dob	gross	net	allowance
admin	Admin	Admin				
chard	Charles	Dickens	06/21/83	8000	7000	2000
johnm	John	Milton	03/16/79	9000	8000	2500

Consider a hypothetical example having the 'users' table as shown above. In the above example, the client can query the information about the employee. For the purpose of demonstration of Coprocessor we assuming that 'admin' is a special person and his details shouldn't be visible to any client querying the table. To achieve this we will take the help of

Coprocessor.

Following are the steps:

1. Write a class that extends the BaseRegionObserver class.
2. Override the 'preGetOp()' method (Note that 'preGet()' method is now deprecated). You should use 'preGetOp' method here because first check if the queried rowkey is 'admin' or not. If it 'admin' then return the call without allowing the system to perform the get operation thus saving on performance.
3. Export your code in a jar file.
4. Place the jar in HDFS where HBase can locate it.
5. Load the Coprocessor.
6. Write a simple program to test it.

Let's see each step in detail:

Step 1 and Step2: Below is a class that extends one of the Coprocessor classes (BaseRegionObserver) and overrides the 'preGetOp' method.

```
public class RegionObserverExample extends BaseRegionObserver {

    private static final byte[] ADMIN = Bytes.toBytes("admin");
    private static final byte[] COLUMN_FAMILY = Bytes.toBytes("details");
    private static final byte[] COLUMN = Bytes.toBytes("Admin_det");
                private static final byte[] VALUE = Bytes.toBytes("You can't see Admin details"

    @Override
    public void preGetOp(final ObserverContext e, final Get get, final List results) throws IOE

        if (Bytes.equals(get.getRow(),ADMIN)) {
            Cell c = CellUtil.createCell(get.getRow(),COLUMN _FAMILY, COLUMN, System.currentTim
            results.add(c);
            e.bypass();
        }

        List kvs = new ArrayList(results.size());
        for (Cell c : results) {
            kvs.add(KeyValueUtil.ensureKeyValue(c));
        }
        preGet(e, get, kvs);
        results.clear();
        results.addAll(kvs);
    }
}
```

Overriding the 'preGetOp()' will only work for 'Get' operation. For 'Scan' operation it won't help you. To deal with it you have to override another method called 'preScannerOpen()' method, and add a Filter explicitly for admin as shown below:

```
@Override
public RegionScanner preScannerOpen(final ObserverContext e, final Scan scan, final RegionScanne

    Filter filter = new RowFilter(CompareOp.NOT_EQUAL, new BinaryComparator(ADMIN));
    scan.setFilter(filter);
    return s;
}
```

This method works but there is a *side effect*. If the client has used any Filter in his scan, then that Filter won't have any effect because our filter has replaced it.

Another option you can try is to deliberately remove the admin from result. This approach is shown below:

```
@Override
public boolean postScannerNext(final ObserverContext e, final InternalScanner s, final List resu
        Result result = null;
        Iterator iterator = results.iterator();
        while (iterator.hasNext()) {
            result = iterator.next();
            if (Bytes.equals(result.getRow(), ROWKEY)) {
                iterator.remove();
                break;
            }
        }
        return hasMore;
    }
```

Step 3: It's pretty convenient to export the above program in a jar file. Let's assume that we exported it in a file called 'coprocessor.jar'.

Step 4: Copy the jar to HDFS. I have used the Hadoop copy command:

hadoop fs –copyFromLocal coprocessor.jar coprocessor.jar

Step 5: See **Loading of Coprocessor**. For observer you can use any of the way you want.

Step 6: Run the following program to test. The first part is testing 'Get' and second 'Scan'.

```
Configuration conf = HBaseConfiguration.create();
HConnection connection = HConnectionManager.createConnection(conf);
HTableInterface table = connection.getTable("users");
Get get = new Get(Bytes.toBytes("admin"));
Result result = table.get(get);
for (Cell c : result.rawCells()) {
    System.out.println(Bytes.toString(CellUtil.cloneRow(c))
        + "==> " + Bytes.toString(CellUtil.cloneFamily(c))
        + "{" + Bytes.toString(CellUtil.cloneQualifier(c))
        + ":" + Bytes.toLong(CellUtil.cloneValue(c)) + "}");
}
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
for (Result res : scanner) {
    for (Cell c : res.rawCells()) {
        System.out.println(Bytes.toString(CellUtil.cloneRow(c))
        + " ==> " + Bytes.toString(CellUtil.cloneFamily(c))
        + " {" + Bytes.toString(CellUtil.cloneQualifier(c))
        + ":" + Bytes.toLong(CellUtil.cloneValue(c))
        + "}");
    }
}
```

2. Endpoint Coprocessor**:** This kind of Coprocessor can be compared to stored procedure found in RDBMS. They help in performing computation which is not possible either through observe Coprocessor or otherwise. For example, calculating average or summation over the entire table that spans across multiple regions. They do so by providing a hook for your custom code and then running it across all regions. With Endpoints Coprocessor you can create your own dynamic RPC protocol and thus can provide communication between client and region server, thus enabling you to run your custom code on region server (on each region of a table). Unlike observer Coprocessor (where your custom code is executed transparently when events like 'Get' operation occurs), in Endpoint Coprocessor you have to explicitly invoke the Coprocessor by using the 'CoprocessorService()' method of the 'HTableInterface' (or HTable). A working example is given below.

From version 0.96, implementing Endpoint Coprocessor is not straight forward. Now it is done with the help of Google's Protocol Buffer. For more details on Protocol Buffer, refer to this excellent Protocol Buffer Guide (https://developers.google.com/protocol-buffers/docs/proto). For migrating Endpoints of version 0.94 or before to 0.96 or later you have to upgrade your Endpoint Coprocessor. For more details, see JIRA ticket HBASE-5448 (https://issues.apache.org/jira/browse/HBASE-5448). For writing Endpoint Coprocessor, one should:

1. Create a '.proto' file defining your service.
2. Execute the 'protoc' command to generate the Java code from the above '.proto' file.
3. Write a class that should:
    1. Extend the above generated *service* class.
    2. It should also implement two interfaces Coprocessor and CoprocessorService.
    3. Override the service method.
4. Load the Coprocessor.
5. Write a client code to call Coprocessor.

Example of Endpoint Coprocessor: We are following the same example as described above. Just to recap:

	ColumnFamily: personalDet			ColumnFamily: salaryDet		
Rowkey	name	surname	dob	gross	net	allowance
admin	Admin	Admin				
chard	Charles	Dickens	06/21/83	8000	7000	2000
johnm	John	Milton	03/16/79	9000	8000	2500

In our hypothetical example (See Table 1), to demonstrate the Endpoint Coprocessor we see a trivial use case and will try to calculate the total (Sum) of gross salary of all employees. We will go step by step:

Step 1: Create a 'proto' file to define your service, request and response. Let's call this file "sum.proto". Below is the content of the 'sum.proto' file

```
option java_package = "org.myname.hbase.Coprocessor.autogenerated";
option java_outer_classname = "Sum";
option java_generic_services = true;
option java_generate_equals_and_hash = true;
option optimize_for = SPEED;
message SumRequest {
    required string family = 1;
    required string column = 2;
}

message SumResponse {
  required int64 sum = 1 [default = 0];
}

service SumService {
  rpc getSum(SumRequest)
    returns (SumResponse);
}
```

Step 2: Compile the proto file using proto compiler (for detailed instructions see this excellent official documentation (https://developers.google.com/protocol-buffers/docs/overview)).

```
$ protoc --java_out=src ./sum.proto
```

(Note: It is necessary for you to create the src folder).
This will generate a class call "Sum.java".

Step 3: Write your Endpoint Coprocessor: Firstly your class should extend the service just defined above (i.e. Sum.SumService). Second it should implement Coprocessor and CoprocessorService interfaces. Third, override the 'getService()', 'start()', 'stop()' and 'getSum()' methods. Below is the full code:

```java
public class SumEndPoint extends SumService implements Coprocessor, CoprocessorService {

    private RegionCoprocessorEnvironment env;

    @Override
    public Service getService() {
        return this;
    }

    @Override
    public void start(CoprocessorEnvironment env) throws IOException {
        if (env instanceof RegionCoprocessorEnvironment) {
            this.env = (RegionCoprocessorEnvironment)env;
        } else {
            throw new CoprocessorException("Must be loaded on a table region!");
        }
    }


    @Override
    public void stop(CoprocessorEnvironment env) throws IOException {
        // do mothing
    }


    @Override
    public void getSum(RpcController controller, SumRequest request, RpcCallback done) {
        Scan scan = new Scan();
        scan.addFamily(Bytes.toBytes(request.getFamily()));
        scan.addColumn(Bytes.toBytes(request.getFamily()), Bytes.toBytes(request.getColumn()));
        SumResponse response = null;
        InternalScanner scanner = null;
        try {
            scanner = env.getRegion().getScanner(scan);
            List results = new ArrayList();
            boolean hasMore = false;
                        long sum = 0L;
                do {
                        hasMore = scanner.next(results);
                        for (Cell cell : results) {
                            sum = sum + Bytes.toLong(CellUtil.cloneValue(cell));
                     }
                        results.clear();
                } while (hasMore);

                response = SumResponse.newBuilder().setSum(sum).build();

        } catch (IOException ioe) {
            ResponseConverter.setControllerException(controller, ioe);
        } finally {
            if (scanner != null) {
                try {
                    scanner.close();
                } catch (IOException ignored) {}
            }
        }
        done.run(response);
    }
```

Step 4: Load the Coprocessor. See loading of Coprocessor. I recommend using static approach for Endpoint Coprocessor.
Step 5: Now we have to write the client code to test it. To do so in your main method, write the following code as shown below:

```java
Configuration conf = HBaseConfiguration.create();
HConnection connection = HConnectionManager.createConnection(conf);
HTableInterface table = connection.getTable("users");
final SumRequest request = SumRequest.newBuilder().setFamily("salaryDet").setColumn("gross").bu
try {
Map<byte[], Long> results = table.CoprocessorService (SumService.class, null, null,
new Batch.Call<SumService, Long>() {
    @Override
        public Long call(SumService aggregate) throws IOException {
BlockingRpcCallback rpcCallback = new BlockingRpcCallback();
            aggregate.getSum(null, request, rpcCallback);
            SumResponse response = rpcCallback.get();
            return response.hasSum() ? response.getSum() : 0L;
        }
    });
    for (Long sum : results.values()) {
        System.out.println("Sum = " + sum);
    }
} catch (ServiceException e) {
e.printStackTrace();
} catch (Throwable e) {
    e.printStackTrace();
}
```

Loading of Coprocessor:

Coprocessor can be loaded broadly in two ways. One is static (loading through configuration files) and the other one is dynamic loading.

Dynamic loading: Dynamic loading means loading Coprocessor without restarting HBase. Dynamic loading can be done in three ways:

A. Using Shell: You can load the Coprocessor using the HBase shell as follows:

1. Disable the table so that you can load Coprocessor

```
hbase(main):001:0> disable ‘users’
```

2. Load the Coprocessor: (i.e. coprocessor.jar) that you copied to HDFS by using following command:

```
or’=>’hdfs://localhost/user/gbhardwaj/coprocessor.jar| org.myname.hbase.Coprocessor.RegionObserverExam
```

where “hdfs://localhost/user/gbhardwaj/coprocessor.jar” is the full path of the ‘coprocessor.jar’ in your HDFS.and “org.myname.hbase.Coprocessor.RegionObserverExample” is the full name of your class (including package name).

3. Enable the table:

```
hbase(main):003:0> enable ‘users'
```

4. Verify if Coprocessor is loaded by typing following command:

```
hbase(main):04:0> describe ‘users'
```

You must see some output like this:

DESCRIPTION ENABLED

```
r| org.myname.hbase.Coprocessor.RegionObserverExample|1073741823|’}, {NAME => ‘ personalDet’ …………………
```

B. Using setValue() method of HTableDescriptor: This is done entirely in Java as follows:

```java
String tableName = "users";
String path = "hdfs://localhost/user/gbhardwaj/coprocessor.jar";
Configuration conf = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
admin.disableTable(tableName);
HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
HColumnDescriptor columnFamily1 = new HColumnDescriptor("personalDet");
columnFamily1.setMaxVersions(3);
hTableDescriptor.addFamily(columnFamily1);
HColumnDescriptor columnFamily2 = new HColumnDescriptor("salaryDet");
columnFamily2.setMaxVersions(3);
hTableDescriptor.addFamily(columnFamily2);
hTableDescriptor.setValue("COPROCESSOR$1", path +
                  "|" + RegionObserverExample.class.getCanonicalName() +
                  "|" + Coprocessor.PRIORITY_USER);
admin.modifyTable(tableName, hTableDescriptor);
admin.enableTable(tableName);
```

C. Using addCoprocessor() method of HTableDescriptor: This method is available from 0.96 version onwards. Personally I prefer this way only:

```java
String tableName = "users";
String path = "hdfs://localhost/user/gbhardwaj/coprocessor.jar";
Configuration conf = HBaseConfiguration.create();
HBaseAdmin admin = new HBaseAdmin(conf);
admin.disableTable(tableName);
HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
HColumnDescriptor columnFamily1 = new HColumnDescriptor("personalDet");
columnFamily1.setMaxVersions(3);
hTableDescriptor.addFamily(columnFamily1);
HColumnDescriptor columnFamily2 = new HColumnDescriptor("salaryDet");
columnFamily2.setMaxVersions(3);
```

```
hTableDescriptor.addFamily(columnFamily2);
hTableDescriptor.addCoprocessor(RegionObserverExample.class.getCanonicalName(), path, Coprocess
admin.modifyTable(tableName, hTableDescriptor);
admin.enableTable(tableName);
```

2. Static Loading: Static loading means that your Coprocessor will take effect only when you restart your HBase and there is a reason for it. In this you make changes 'hbase-site.xml' and therefore have to restart HBase for your changes to take place.

Create following entry in 'hbase-site.xml' file located in 'conf' directory:

```
hbase.Coprocessor.region.classes
org.myname.hbase.Coprocessor.endpoint.SumEndPoint
```

Make your code available to the HBase. I used the following simple steps – first, export your endpoint (SumEndPoint. java), service (Sum.java) and other relevant protocol buffer classes (i.e. all classes found under '<Protocol Buffer Directory>/ java/src/main/java/com/google/protobuf' directory in jar file). Second, put this jar in the 'lib' folder of HBase and finally restart the HBase.

Note: Although the documentation (https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/HTableDescriptor.html#addCoprocessor%28java.lang.String,% clearly states:

"*Add a table Coprocessor to this table. The Coprocessor type must be RegionObserver or Endpoint. It won't check if the class can be loaded or not. Whether a Coprocessor is loadable or not will be determined when a region is opened.*"

This means that you can load both Observer and Endpoint Coprocessor statically using the following Method of HTableDescriptor (https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/HTableDescriptor.html):

```
addCoprocessor(String className, org.apache.hadoop.fs.Path jarFilePath, int priority, Map<String
```

In my case, the above method worked fine for Observer Coprocessor but didn't work for Endpoint Coprocessor, causing the table to become unavailable and finally I had to restart my HBase. The same Endpoint Coprocessor worked fine when loaded statically. Use the above method for Endpoint Coprocessor with caution.

As I wrap up, here are **few terms I have used in the above blog**:

Column Family: It is generally said that HBase is a column-oriented database, which means it has columns, and columns are grouped in a column family. While columns are dynamic (i.e. there are no fixed number of columns in a row as opposed to RDBMS where the number of columns are fixed all the time) and it is not required to be declared at the time of table creation, column families are fixed and it is required to define them at the time of table creation. They must remain more or less fixed throughout the life of the table. You can add column families after table creation but it is an expensive operation and is generally avoided.

HFile: Column Family is stored in HFile. A column family can span across multiple HFiles (until major compaction) but the converse is not true, i.e. an HFile cannot have multiple column families; it will always host a single column family.

Region: Although HBase tables host billions of rows,it is not possible to store all that together. Therefore, it is divided into chunks. A Region is responsible for holding a set of rows. Also when you keep adding more rows and its size grows beyond a threshold value, it automatically splits in two regions. A region hosts a continuous set of rows.

RegionsServer: Regions are hosted by Region Server. A Region Server can host many Regions but a Region is always hosted by one and only one Region Server.

(https://www.3pillarglobal.com/author/gbhardwaj)

GAURAV BHARDWAJ
TECHNICAL LEAD

Gaurav Bhardwaj is working as Technical Lead at 3Pillar Global. He has over 10 years of experience in the field of design and development and has provided enterprise solution using Java technology stack and various open source frameworks. He has worked in various domains like Telecom, Pharmaceutical, E-commerce, and B2B Integration. His area of interest is in Big Data technologies including Hadoop, HBase, Pig, Hive, etc. Prior to joining 3Pillar, Gaurav was working with Zone Corporate Solutions.

21 RESPONSES TO "THE HOW TO OF HBASE COPROCESSORS"

Saurabh Chhajed (http://saurzcode.in) on January 13th, 2015 - 8:12am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-34173)

I think this is the best explanation of CoProcessor , I have seen so far, which is easy to understand and explains well.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=34173#respond)

Gaurav Bhardwaj on October 26th, 2015 - 3:25am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77686)

Thanks Saurabh. Sorry for the delay in checking the comments.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77686#respond)

jeet on February 24th, 2015 - 12:09am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-40171)

Hi Gaurav This is very nice article.

I want to create a inverted index on put/delete should I use behavior coprocessor or Endpoint?

Also suppose if code inside the co processor changes what will happen. do I need disable the hbase ?

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=40171#respond)

Gaurav Bhardwaj on October 26th, 2015 - 3:29am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77687)

Thanks Jeet!, Sorry for checking your comment so late.
Whenever the coprocessor changes then you have to re-apply it, by disabling and then enabling the table with the coprocessor.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77687#respond)

Ashish Bhalgat on April 3rd, 2015 - 10:00am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-45872)

Awesome article man.
Explained complex topic in simple way. Nice/Worth reading.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=45872#respond)

Gaurav Bhardwaj on October 26th, 2015 - 3:32am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77689)

Thanks Ashish!. Sorry for checking the comments so late 🙁

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77689#respond)

somnath on June 3rd, 2015 - 5:24am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-55172)

Where will it copy the .jar file after executing this command so that i can mention it while loading coprocessor?
>hadoop fs –copyFromLocal coprocessor.jar coprocessor.jar

or Do we need to put it into /hbase/lib folder ?

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=55172#respond)

Gaurav Bhardwaj on October 26th, 2015 - 3:46am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77691)

Sorry for replying late.
First of all the command "hadoop fs –copyFromLocal coprocessor.jar coprocessor.jar" is a normal Hadoop File System command.
Secondly it copies the files from local file syatem to to Hadoop at the location http://namenode:8020/user/hadoop-user/ (http://namenode:8020/user/hadoop-user/) OR at the location you specify in the copy command.
Whether you want to copy it using the above command or want to put to 'hbase/lib' folder depends on your requirement.
Generally when you load the coprocessor statically and you want to apply it to all the HBASE table then you copy it in the 'hbase/lib' folder make an entry in 'hbase-site.xml' and restart the HBASE cluster (explained above).
On other hand if you want to apply the coprocessor to a single table and don't wish to restart the HBASE, you go with dnamic loading (explained above) and in that case it is enough to copy the jar file in hadoop using copy or put command of Hadoop.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77691#respond)

Rohit Mathews on July 8th, 2015 - 5:47pm (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-60030)

Brilliant. Loved this article. Great example and great walkthrough.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=60030#respond)

Gaurav Bhardwaj on [October 26th, 2015 - 3:47am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77692)]

Thanks Rohit and sorry for checking your comment so late 🙁

[Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77692#respond)]

Eleanor on [July 17th, 2015 - 5:04pm (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-60998)]

Hey,
just a small question,
in the example java file, do we have to import all the packages?
I'm using Maven, but I'm not sure if the import command in Maven will help.
Thanks.

[Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=60998#respond)]

Gaurav Bhardwaj on [October 26th, 2015 - 3:53am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77693)]

Eleanor, sorry for replying late.
I doubt if I understood the question correctly..
I have used the maven, and with maven it is really easy to create the jar because maven make is very easy, you can use various plugins like maven-assembly-plugin or maven-shade-plugin etc.

[Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77693#respond)]

Prasad Khode on [July 24th, 2015 - 3:44am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-61951)]

Really nice post, got clarity on Coprocessors in HBase.

One problem is, I'm trying to use Endpoint Coprocessors on CDH 5.4.3 with HBase 1.0.

I have created SumEndPoint. java, Sum.java as a jar and copied into HBase lib directory and updated hbase-site.xml with the mentioned entry. When I try to run my client I'm facing the following exception:

```
15/07/24 07:27:11 WARN ipc.CoprocessorRpcChannel: Call failed on IOException
org.apache.hadoop.hbase.exceptions.UnknownProtocolException:
org.apache.hadoop.hbase.exceptions.UnknownProtocolException: No registered coprocessor service found for
name coprocessor.SumService in region users,,1437720682419.5a073891b5fdc2c5996721fb51428cd6.
at org.apache.hadoop.hbase.regionserver.HRegion.execService(HRegion.java:7031)
at org.apache.hadoop.hbase.regionserver.RSRpcServices.execServiceOnRegion(RSRpcServices.java:1746)
at org.apache.hadoop.hbase.regionserver.RSRpcServices.execService(RSRpcServices.java:1728)
at
org.apache.hadoop.hbase.protobuf.generated.ClientProtos$ClientService$2.callBlockingMethod(ClientProtos.java:31447
at org.apache.hadoop.hbase.ipc.RpcServer.call(RpcServer.java:2035)
at org.apache.hadoop.hbase.ipc.CallRunner.run(CallRunner.java:107)
at org.apache.hadoop.hbase.ipc.RpcExecutor.consumerLoop(RpcExecutor.java:130)
at org.apache.hadoop.hbase.ipc.RpcExecutor$1.run(RpcExecutor.java:107)
at java.lang.Thread.run(Thread.java:745)

at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)
at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
at java.lang.reflect.Constructor.newInstance(Constructor.java:526)
at org.apache.hadoop.ipc.RemoteException.instantiateException(RemoteException.java:106)
at org.apache.hadoop.ipc.RemoteException.unwrapRemoteException(RemoteException.java:95)
at org.apache.hadoop.hbase.protobuf.ProtobufUtil.getRemoteException(ProtobufUtil.java:326)
at org.apache.hadoop.hbase.protobuf.ProtobufUtil.execService(ProtobufUtil.java:1622)
at org.apache.hadoop.hbase.ipc.RegionCoprocessorRpcChannel$1.call(RegionCoprocessorRpcChannel.java:92)
at org.apache.hadoop.hbase.ipc.RegionCoprocessorRpcChannel$1.call(RegionCoprocessorRpcChannel.java:89)
at org.apache.hadoop.hbase.client.RpcRetryingCaller.callWithRetries(RpcRetryingCaller.java:126)
at
org.apache.hadoop.hbase.ipc.RegionCoprocessorRpcChannel.callExecService(RegionCoprocessorRpcChannel.java:95)
at org.apache.hadoop.hbase.ipc.CoprocessorRpcChannel.callMethod(CoprocessorRpcChannel.java:56)
at com.vizzario.coprocessor.Sum$SumService$Stub.getSum(Sum.java:1250)
at com.vizzario.coprocessor.TestCoProcessor$1.call(TestCoProcessor.java:54)
at com.vizzario.coprocessor.TestCoProcessor$1.call(TestCoProcessor.java:48)
at org.apache.hadoop.hbase.client.HTable$16.call(HTable.java:1797)
at java.util.concurrent.FutureTask.run(FutureTask.java:262)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:745)
Caused by:
org.apache.hadoop.hbase.ipc.RemoteWithExtrasException(org.apache.hadoop.hbase.exceptions.UnknownProtocolExcept
org.apache.hadoop.hbase.exceptions.UnknownProtocolException: No registered coprocessor service found for
```

```
name coprocessor.SumService in region users,,1437720682419.5a073891b5fdc2c5996721fb51428cd6.
at org.apache.hadoop.hbase.regionserver.HRegion.execService(HRegion.java:7031)
at org.apache.hadoop.hbase.regionserver.RSRpcServices.execServiceOnRegion(RSRpcServices.java:1746)
at org.apache.hadoop.hbase.regionserver.RSRpcServices.execService(RSRpcServices.java:1728)
at
org.apache.hadoop.hbase.protobuf.generated.ClientProtos$ClientService$2.callBlockingMethod(ClientProtos.java:31447
at org.apache.hadoop.hbase.ipc.RpcServer.call(RpcServer.java:2035)
at org.apache.hadoop.hbase.ipc.CallRunner.run(CallRunner.java:107)
at org.apache.hadoop.hbase.ipc.RpcExecutor.consumerLoop(RpcExecutor.java:130)
at org.apache.hadoop.hbase.ipc.RpcExecutor$1.run(RpcExecutor.java:107)
at java.lang.Thread.run(Thread.java:745)

at org.apache.hadoop.hbase.ipc.RpcClientImpl.call(RpcClientImpl.java:1200)
at org.apache.hadoop.hbase.ipc.AbstractRpcClient.callBlockingMethod(AbstractRpcClient.java:216)
at
org.apache.hadoop.hbase.ipc.AbstractRpcClient$BlockingRpcChannelImplementation.callBlockingMethod(AbstractRpcCl
at
org.apache.hadoop.hbase.protobuf.generated.ClientProtos$ClientService$BlockingStub.execService(ClientProtos.java:31
at org.apache.hadoop.hbase.protobuf.ProtobufUtil.execService(ProtobufUtil.java:1618)
… 13 more
```

So instead of doing these changes only in master node, I have replicated same in even region server and when I run the client I'm still getting the same exception

Any help will be greatly appreciated….

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=61951#respond)

Gaurav Bhardwaj on October 26th, 2015 - 4:30am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77698)

Sorry for replying late.
It seems your coprocessor is not applied.
If your HBASE is clustered over bunch of machined make sure it is available to every RegionServer.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77698#respond)

Neha on September 10th, 2015 - 8:20am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-70477)

Thanks. It was awesomely explained.

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=70477#respond)

Gaurav Bhardwaj on October 26th, 2015 - 4:31am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-77699)

Thanks Neha. Sorry for replying late 🙁

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=77699#respond)

Prasanth on December 15th, 2015 - 7:53pm (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-83176)

GAURAV BHARDWAJ,
Very nice article. Got one quick question. Can we remove a row via prePut?

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=83176#respond)

Gaurav Bhardwaj on December 16th, 2015 - 2:57am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-83203)

Thanks Man!
As far as your question is concerned, I am assuming that you want to remove some row (either from the same table or some other table) while 'putting' a particular row. Certainly you can remove a row but mostly (not in every case) its better to move that operation to 'postPut', as 'postPut' ensures that your put has executed successfully handling the case that put can throw exception and then put want to revert whatever you have done earlier in 'prePut' in case you wish have 'atomic' behavior else its ok to have that in 'prePut'

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=83203#respond)

Karan (https://abc.com) on January 28th, 2016 - 6:07am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-87125)

Tried using above with with preGet and preScan using version as below

org.apache.hbase
hbase
0.94.6-cdh4.3.0

Able to call Get but not PreGet.
Please help!
Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=87125#respond)

Mohammad Adnan on March 22nd, 2016 - 3:46am (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-102309)

Nice article.
I have one query though if you may help me in this. If you have postPut region observer and the observer itself inserting another row in the same table to which co-processor is enabled. This will end up doing update in loop. is there a way to avoid this gracefully?

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=102309#respond)

Ravi Papisetti on April 5th, 2016 - 1:47pm (https://www.3pillarglobal.com/insights/hbase-coprocessors#comment-105699)

Great article, I am looking to calculate metrics such as top 5 values, min, max, avg for all columns of an hbase table. Any thought what are best practices for these kind of applications? When I loop it for each column, it takes about few minutes for each column. Appreciate your expertise..

Reply (https://www.3pillarglobal.com/insights/hbase-coprocessors?replytocom=105699#respond)

LEAVE A REPLY

Name (required)

Your Name*

Mail (required)

Your E-Mail*

Comment

Your Comment here...

进行人机身份验证

reCAPTCHA
隐私权 - 使用条款

SUBMIT

Learn More
Articles (https://www.3pillarglobal.com/news/articles)
Awards (https://www.3pillarglobal.com/news/awards)
Events (https://www.3pillarglobal.com/news/events)
Press Releases (https://www.3pillarglobal.com/news/press-release)
Blogs (https://www.3pillarglobal.com/insights/blog-posts)
Podcasts (https://www.3pillarglobal.com/insights/podcast)
Videos (https://www.3pillarglobal.com/insights/videos)

RELATED POSTS

Take 3, Scene 15: Inside the DL Labs

(https://www.3pillarglobal.com/insights/take-3-scene-15-inside-the-dl-labs) (https://www.3pillarglobal.com/insights/take-3-scene-15-inside-the-dl-labs)

(https://www.3pillarglobal.com/insights/aws-device-farm-mobile-app-testing-on-real-devices) AWS Device Farm – Mobile App Testing on Real Devices (https://www.3pillarglobal.com/insights/aws-device-farm-mobile-app-testing-on-real-devices)

(https://www.3pillarglobal.com/insights/how-do-you-plan-in-an-agile-environment) How Do You Plan In An Agile Environment? (https://www.3pillarglobal.com/insights/how-do-you-plan-in-an-agile-environment)

(https://www.3pillarglobal.com/news/adi-chikara-to-speak-at-2016-rightstech-summit) Adi Chikara to Speak at 2016 RightsTech Summit (https://www.3pillarglobal.com/news/adi-chikara-to-speak-at-2016-rightstech-summit)

(https://www.3pillarglobal.com/insights/take-3-scene-14-present-future-angular-2-part-2) Take 3, Scene 14: The Present and Future of Angular 2, Part ... (https://www.3pillarglobal.com/insights/take-3-scene-14-present-future-angular-2-part-2)

Free product development tips delivered right to your inbox

Email Address

SUBSCRIBE

LET'S WORK TOGETHER (/CONTACT)