Certainly! Let's dive deeper into YARN (Yet Another Resource Negotiator), focusing on its architecture, components, and a technical example.

YARN Architecture Overview

YARN is designed to provide a more scalable and flexible way of managing the computing resources in the Hadoop cluster. It separates the functionalities of resource management and job scheduling/monitoring into separate daemons: the ResourceManager (RM), the NodeManager (NM), and the ApplicationMaster (AM).

1. **ResourceManager (RM):**
    - The RM is the master daemon of YARN and is responsible for allocating resources to various running applications.
    - It has two main components:
        - **Scheduler:** Responsible for allocating resources to various running applications, based on constraints such as capacity, queues, etc.
        - **ApplicationManager:** Manages the lifecycle of applications and is responsible for accepting job submissions, negotiating the first container for executing the application-specific ApplicationMaster, and provides the service for restarting the AM container on failure.
2. **NodeManager (NM):**
    - A per-node slave daemon, it is responsible for monitoring resource usage (CPU, memory, disk, network) and reporting the same to the ResourceManager.
    - It is also responsible for managing the application's running containers, ensuring they are executed with the resource constraints (memory/CPU) specified.
3. **ApplicationMaster (AM):**
    - An instance of the ApplicationMaster is created for each application. The AM is responsible for negotiating appropriate resource containers from the Scheduler, tracking their status, and monitoring progress.

Example: Running a MapReduce Job on YARN

Prerequisites:

- A Hadoop cluster with YARN configured.
- Sample data and MapReduce job (e.g., word count).

Steps:

1. **Prepare the Environment:**
    - Place your input data in HDFS:
      `hdfs dfs -put localfile.txt /user/hadoop/input`
2. **Write a MapReduce Program:**

- For simplicity, let's assume a basic word count program in Java.

3. **Compile and Package the Application:**
   - Use Maven or another build tool to compile your Java code and package it into a JAR.
4. **Submit the Job:**
   - Use the `yarn` command to submit the job to the cluster.
     `yarn jar your-application.jar com.example.WordCount /user/hadoop/input /user/hadoop`
5. **Monitoring the Job:**
   - You can monitor the job's progress through the YARN ResourceManager UI.

Example Code (WordCount.java):

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.StringTokenizer;

public class WordCount {

  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context) throws IOException, Interrupte
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOExc
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```