Job Management System with fair scheduling
Local and Web Service
Cloud Computing, Summer 2013

Submitted to:
Prof. Wang

Submitted by:
Ryda Nofal
Diwakar Kumawat
Rohit Dhawan

Acknowledgements:

It is our pleasure to be indebted to various students, who directly or indirectly contributed in the development in this project and who influenced our thinking and acts during the course of study. We take this opportunity to express our profound gratitude and deep regards to our Professor for his exemplary guidance, monitoring and constant encouragement throughout the whole course. The blessing, help and guidance given by him time to time shall carry us a long way in the journey of life on which we are about to embark. We also would like to express our gratitude towards our classmates for their kind co-operation and encouragement who helped us in the completion of this project.

Lastly, we are highly thankful to almighty God and our friends with whom we shared our day-to-day experience and received lots of suggestions that improved our quality of work.

Table of content:

Abstract……...3
Introduction……4
What is the problem……...4
Why is this project related to this class……....4
Why other approach is not good…….....4
Statement of the problem ……4
Area or scope of investigation……...4
Theoretical basis and literature review……...4-9
Definition of the problem……...4-5
Theoretical background of the problem…….....5
Related research to solve the problem……..5
Advantage/disadvantage of those research……...5-6
Our solution to solve this problem and why it is better……...…6
Where our solution is different from others……...…..9
Hypothesis……...……9
Methodology……...…..9
How to solve the problem……....9
Algorithm design……..9
Language used……..9
How to test against hypothesis……....9
Implementation……9-38
Code……..…..9-37
Flowchart……..……38
Data analysis and discussion……..……39-40
Output generation…….....…39
Output analysis……....39
Compare output against hypothesis……..39
Abnormal case explanation…….....39-40
Discussion……..40
Conclusions……...……40
Summary and conclusions…….....40
Bibliography……...……..40

1) Abstract:

In this project, we have implemented a Job Management System with fair scheduler in a cluster. Use of fair scheduler improves turn-around and throughput by 2 - 10 times in comparison to old traditional schedulers.. The old schedulers using FIFO, time-sharing, pre-emption or sometimes shortest job first policies. This results in poor utilization of resources, throughput and turn-around for big data applications. There is need for fair scheduler to "fairly" run different kinds of jobs in terms of load, priority, utility, application etc. So, we have worked on job's priority, displaying real time cluster load/progress and displaying real time node load. It will allow a job to kill during execution. Moreover, we can raise or lower the job priority. It will allow a job to be paused as well during execution. We have implemented a Custom Task Scheduling Algorithm where there is a Priority Queue created and assigned to each Job. We have used Java because it is dynamic, flexible, and portable. Java technology has much to offer to the developers of cluster management solutions. In addition, we run local on a cluster that has N nodes, we are going to deploy our code on a cluster on the cloud then apply all the job functions to run through a web services as well as from a python program to demonstrate application to application.

2) Introduction:

- **What is the problem:** In our project, we would like to implement a JMS with fair scheduler in a cluster. It will provide features to over-ride default policy which is First In First Out. We would like to work on job's priority, displaying real time cluster load/progress and displaying real time node load. It will allow a job to kill during execution. We can raise or lower the job priority. It will allow a job to be paused.

Web Based JMS: In addition to job management system run local on a cluster that has N nodes, we are going to deploy our code on a cluster on the cloud then apply all the job functions to run through a web services as well as from a python program to demonstrate application to application.

- **Why is this project related to this class:** Our topic is related to our class because we know that monitoring is at the heart of cluster management these days. Instrumentation data is used to schedule tasks, load-balance devices and services, notify administrators of hardware and software failures, and generally monitor the health and usage of a system. The information used to perform these operations must be gathered from the cluster without impacting performance. JMS is used in various applications like spam detection in web site traffic, controlling traffic congestions, preventing DOS attacks by IP addresses etc.
- **Why other approach is not good:** We have used fair scheduler because it improves turn-around and throughput by 2-10 times in comparison to traditional schedulers. Traditional schedulers use time-sharing, FIFO, pre-emption or shortest job first policies. Such policies result in poor utilization, throughput and turn-around for Big Data applications. We believe that there is need for fair scheduler to "fairly" run different kinds of jobs (by load, priority, utility, application etc). Morever, fair scheduler works around 2 of the main bottlenecks with Hadoop/ Map-Reduce frameworks i.e., Data Locality and dependence between map and reduce jobs.
- **Statement of the problem:** Hadoop's Scheduler uses FIFO (default) Scheduling Algorithm. This approach has an inherent 'un-fair' issue - where large Jobs can starve smaller Jobs of a Cluster's resources. Facebook introduced 'Fair Scheduler' where Each Job is guaranteed a certain share of Resources such that Large Jobs can't starve Small Jobs. In fact small jobs get faster Completion times as compared to large Jobs.
- **Area or scope of investigation:** We have implemented a Java based Job Management System with "Fair Scheduling" which can be overridden by the User. Algorithm uses Priority Concurrent Queue's for each Client's Job and breaks a Job into tasks using Exponential Distribution. The Goal is to keep all resources (Nodes) as busy as possible in order to achieve best possible turn around time and throughput.

Adding the capability to run the command through web services. In order to implement the idea, we can run a web server on one of the cluster nodes. Users send their jobs to that server through a web call. We need also to have authentication system to guarantee only authorized users can have access to our web service, submit jobs and view the results.

---

3) Theoretical bases and literature review:

- **Definition of the problem:** Increased scalability and complexity of large computational systems paves the way for sophisticated management tools . Increasing grid resources

and computational load have increased the demands on a system administrator on the account of monitoring the grid and managing the running jobs.

- **Theoretical background of the problem:** Tools for monitoring Linux clusters have traditionally provided a limited amount of data with delivery frequencies measured in seconds. High performance cluster monitoring is defined as the ability to efficiently gather data from nodes at intrasecond frequencies. When dealing with large clusters, inefficiencies in the monitoring software become increasingly problematic. This is because running applications must coordinate with each other or with shared global resources. Interference on one node can affect jobs running on other nodes. An example of this is an MPI job that needs to synchronize all participating nodes. One solution to this problem is to gather less data and transmit it less frequently. However if the objective is high-performance monitoring, this solution is unacceptable. A cluster that is heavily utilized should be monitored constantly and frequently. Local job schedulers must be able to make decisions quickly based on resource usage. Administrators often want to be notified of critical events and view historic trend data, which is unavailable unless the cluster is monitored constantly and consistently. Therefore utilizing more efficient algorithms, increasing transmission parallelism, increasing the efficiency of the transmission protocol and data format, and reducing redundancy are needed. Cluster monitoring primarily consumes two important resources: CPU cycles and network bandwidth. However, the resource consumption problem is fundamentally different for these two resources. The CPU usage problem is completely localized to the node, and is addressed by creating efficient gathering and consolidating algorithms. Network bandwidth however is a shared resource and is a problem of scale. The network bandwidth usage problem is addressed by finding ways to minimize the amount of data transmitted over the network.

We will be using Java because it is dynamic, flexible, and portable. Java technology has much to offer to the developers of cluster management solutions. These unique features make it an ideal foundation for building cluster management solutions over heterogeneous networks and platforms. Java has an extensive library of routines for coping easily with IP protocols such as TCP and UDP, and for network programming on multi-homed hosts. This makes creating network connections much easier than in C or C++.

Sharing a MapReduce cluster between users is attractive because it enables statistical multiplexing (lowering costs) and allows users to share a common large data set. However, traditional scheduling algorithms can perform very poorly in MapReduce due to two aspects of the MapReduce setting: the need for data locality (running computation where the data is) and the dependence between map and reduce tasks. We will illustrate these problems through our experience designing a fair scheduler.

- **Related research to solve the problem:** We did the research based on few papers from IEEE, ACM and other technical societies. We followed high-performance Linux cluster monitoring using Java by Curtis Smith, Job-Oriented Monitoring of Clusters and cloud control with Distributed Rate Limiting, Barath Raghavan etc.
- **Advantage/disadvantage of those research:**

Advantages:

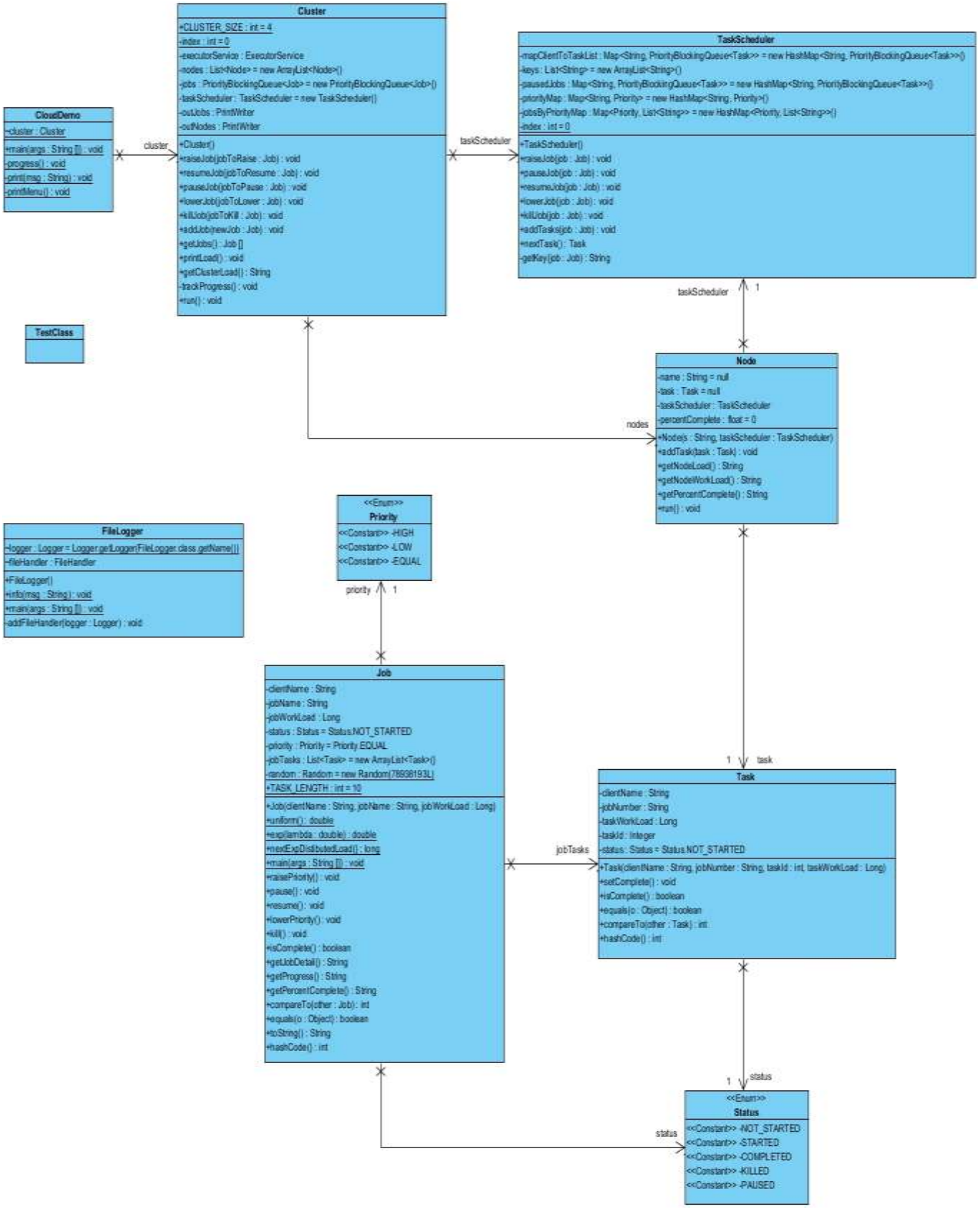
The system monitors the jobs being executed and the resources on each node like RAM, disk space, CPU which leads to better resource utilization and Idle and increased cluster performance. Time critical jobs are monitored and tracked easily. Job redistribution takes place as soon as a Job failure is detected. Load balancing is achieved according to the efficiency analysis of the computing nodes. The system is made fault-tolerant**.**

Disadvantages:

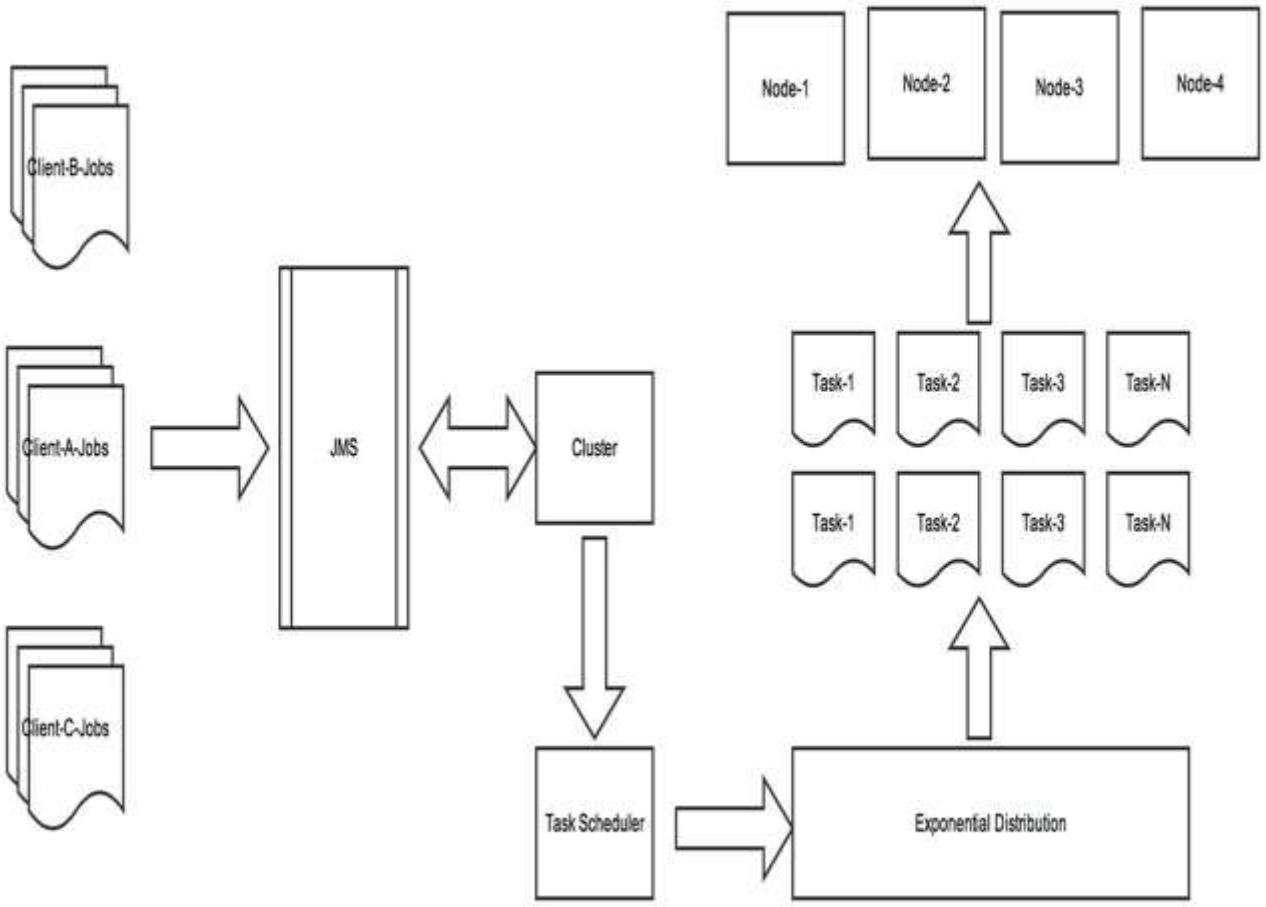
Still Proposed architectures in those papers needs further enhancement to be capable of monitoring multiple clusters simultaneously. External job schedulers should be used for better scheduling policies.

- **Our solution to solve this problem and why it is better:**

We implemented a Custom Task Scheduling Algorithm where there is a Priority Queue created and assigned to each Job. A Job is divided into 1 or more tasks in an Exponential Distribution. Each Node when Free to Compute, asks TaskScheduler for next Task. TaskScheduler uses Priority of each Job, Status (Paused, Killed, Active) and round robins among eligible Tasks and submits a Task to Node. Node computes the work and again asks Scheduler for next Task.
Algorithm ensures a "Fairness" based on Priority and State of a Job. Algorithm also ensures that all Nodes are busy and small Jobs get completed quickly at the same time ensuring Large Jobs get enough Resources. Our implementation is better then FIFO as it provides for democratic Resource sharing irrespective of Job Size. Resources are fairly divided among all Clients and their Jobs. User can control Job Execution by raising/lowering Priorities, Pausing/Resuming Jobs or even Killing a long running/non-responding Job. Cluster and Node Level Statistics are available in Real Time.


Cluster
+CLUSTER_SIZE : int = 4
-index : int = 0
-executorService : ExecutorService
-nodes : List<Node> = new ArrayList<Node>()
-jobs : PriorityBlockingQueue<Job> = new PriorityBlockingQueue<Job>()
-taskScheduler : TaskScheduler = new TaskScheduler()
-outJobs : PrintWriter
-outNodes : PrintWriter
+Cluster()
+raiseJob(jobToRaise : Job) : void
+resumeJob(jobToResume : Job) : void
+pauseJob(jobToPause : Job) : void
+lowerJob(jobToLower : Job) : void
+killJob(jobToKill : Job) : void
+addJob(newJob : Job) : void
+getJobs() : Job []
+printLoad() : void
+getClusterLoad() : String
-trackProgress() : void
+run() : void
CloudDemo
-cluster : Cluster
+main(args : String []) : void
-progress() : void
-print(msg : String) : void
-printMenu() : void
cluster
taskScheduler
TaskScheduler
-mapClientToTaskList : Map<String, PriorityBlockingQueue<Task>> = new HashMap<String, PriorityBlockingQueue<Task>>()
-keys : List<String> = new ArrayList<String>()
-pausedJobs : Map<String, PriorityBlockingQueue<Task>> = new HashMap<String, PriorityBlockingQueue<Task>>()
-priorityMap : Map<String, Priority> = new HashMap<String, Priority>()
-jobsByPriorityMap : Map<Priority, List<String>> = new HashMap<Priority, List<String>>()
-index : int = 0
+TaskScheduler()
+raiseJob(job : Job) : void
+pauseJob(job : Job) : void
+resumeJob(job : Job) : void
+lowerJob(job : Job) : void
+killJob(job : Job) : void
+addTasks(job : Job) : void
+nextTask() : Task
-getKey(job : Job) : String
taskScheduler
1
TestClass
Node
-name : String = null
-task : Task = null
-taskScheduler : TaskScheduler
-percentComplete : float = 0
+Node(s : String, taskScheduler : TaskScheduler)
+addTask(task : Task) : void
+getNodeLoad() : String
+getNodeWorkLoad() : String
+getPercentComplete() : String
+run() : void
nodes
<<Enum>>
Priority
<<Constant>> -HIGH
<<Constant>> -LOW
<<Constant>> -EQUAL
priority
1
FileLogger
-logger : Logger = Logger.getLogger(FileLogger.class.getName())
-fileHandler : FileHandler
+FileLogger()
+info(msg : String) : void
+main(args : String []) : void
-addFileHandler(logger : Logger) : void
Job
-clientName : String
-jobName : String
-jobWorkLoad : Long
-status : Status = Status.NOT_STARTED
-priority : Priority = Priority.EQUAL
-jobTasks : List<Task> = new ArrayList<Task>()
-random : Random = new Random(78938193L)
+TASK_LENGTH : int = 10
+Job(clientName : String, jobName : String, jobWorkLoad : Long)
+uniform() : double
+exp(lambda : double) : double
+nextExpDistibutedLoad() : long
+main(args : String []) : void
+raisePriority() : void
+pause() : void
+resume() : void
+lowerPriority() : void
+kill() : void
+isComplete() : boolean
+getJobDetail() : String
+getProgress() : String
+getPercentComplete() : String
+compareTo(other : Job) : int
+equals(o : Object) : boolean
+toString() : String
+hashCode() : int
1
task
Task
-clientName : String
-jobNumber : String
-taskWorkLoad : Long
-taskId : Integer
-status : Status = Status.NOT_STARTED
+Task(clientName : String, jobNumber : String, taskId : int, taskWorkLoad : Long)
+setComplete() : void
+isComplete() : boolean
+equals(o : Object) : boolean
+compareTo(other : Task) : int
+hashCode() : int
jobTasks
1
status
<<Enum>>
Status
<<Constant>> -NOT_STARTED
<<Constant>> -STARTED
<<Constant>> -COMPLETED
<<Constant>> -KILLED
<<Constant>> -PAUSED
status

UML for JMS



Cloud JMS

Architecture: REST to implement the web service functions:
REST is an architectural style which is based on web-standards and the HTTP protocol.
Java defines REST support via the Java Specification Request 311 (JSR). This specification is called JAX-RS (The Java API for RESTful Web Services). JAX-RS uses annotations to define the REST relevance of Java classes.
Jersey is the reference implementation for this specification. Jersey contains basically a REST server and a REST client.

Deployment:
In a REST based architecture a server is need REST server.
Google offers a free of charge solution based on Java (Java development on the Google App Engine) .
For JMS projects a private server was built and used http://rnofal.dyndns-home.com:8080
Web service sever is tomcat6 (Apache http server)

- **Where our solution is different from others:**

In JMS-Clusters we provided a JMS solution that performs cluster monitoring. Our system monitors a job execution at each step of its life cycle down to the task level, fine granularity monitoring. Furthermore, the solution we designed uses the monitoring information to perform fair job scheduling performed by a scheduler. This project proposes a 3 key architecture, consisting of a monitor and a dispatcher- parallel to our monitoring and scheduling functions, in addition to a job logs component.

---

4) Hypothesis:

Our hypothesis states that our fair scheduler is better than hadoop scheduler. Our goal is to implement in Java a custom implementation of "Fair Scheduler" which improves Throughput and Response Time as compared to FIFO based Default Hadoop Scheduler. Also provide User level controls to Manage Priority, Job Lifecycle and Real Time Statistics.

---

5) Methodology:

- **How to solve the problem:** The only way a job can be higher propriety is if you the user explicitly mark it by menu. In this case we assume user wants to first finish this job before lower priority jobs. We have fair scheduling when there are no priority changes - then big jobs and all jobs all get fair resources.
- **Algorithm design:** The current system doesn't have a dispatcher that sends the next job to a node. We emulate this mechanism by creating a pool of threads with the size of the cluster (number of nodes). We break incoming jobs into 10 sec long tasks and pass them to those thread. Actually, in this emulation those tasks sleep most of the time. We need to create a dispatcher that sends each task to a node through ssh command or socket connection. This though, is way beyond the boundary of this project.
- **Language used:** Java technology has much to offer to the developers of cluster management solutions. Java is dynamic, flexible, and portable. These unique features make it an ideal foundation for building cluster management solutions over heterogeneous networks and platforms. Java has an extensive library of routines for coping easily with IP protocols such as TCP and UDP, and for network programming on multi-homed hosts. This makes creating network connections much easier than in C or C++. Through the Java Native Interface (JNI), Java code that runs within a Java Virtual Machine (JVM) can interoperate with applications and libraries written in other languages, such as C, C++, and assembly.
- **How to test against hypothesis:** Measurement of Performance, Cluster Throughput, Node Throughput all could be compared with other Distributions like Bernoulli, Binomial, Pareto etc and compare the results.

---

6) Implementation:

- **Code:**

  Main driver Class. Takes Input and sends Input Commands to Cluster

```
import java.io.BufferedReader;
```

```
import java.io.InputStreamReader;
import java.util.List;
import java.util.StringTokenizer;


/**
* Created with IntelliJ IDEA.
* User: dkumawat
* Date: 8/3/13
* Time: 12:32 PM
* To change this template use File | Settings | File Templates.
*/
public class CloudDemo {


  static Cluster cluster;


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


  String eoi = null;// end of input
  BufferedReader br = (new BufferedReader(new InputStreamReader(System.in)));
  String input = null;
  boolean continued = true;
  FileLogger fileLogger = new FileLogger();
  cluster = new Cluster();
  do {
     printMenu();
     //eoi = c.readLine("Enter input (type end to End Program): ");
     System.out.print("Enter Menu Number (type 0 to End Program): ");
     eoi = br.readLine();
     int command = -1;
     try {
        command = Integer.parseInt(eoi);
     }catch(NumberFormatException nfx) {
        print("Invalid Command. Try Again.");
        continue;
     }


     if(command < 0 || command > 9) {
        print("Invalid Command. Try Again.");
        continue;
     }
```

```
if(command == 9) {
   //print(cluster.getClusterLoad());
   cluster.printLoad();
}


if(command == 5) {
   print("Enter Job to Pause {ClientName JobName}");
   input = br.readLine();
   StringTokenizer stk = new StringTokenizer(input);
   cluster.pauseJob(new Job(stk.nextToken(), stk.nextToken(), null));
}


if(command == 6) {
   print("Enter Job to Resume {ClientName JobName}");
   input = br.readLine();
   StringTokenizer stk = new StringTokenizer(input);
   cluster.resumeJob(new Job(stk.nextToken(), stk.nextToken(), null));
}


if(command == 7) {
   print("Enter Job And GuaranteedRate(%) {ClientName JobName
25/50/75/100}");
   input = br.readLine();
   StringTokenizer stk = new StringTokenizer(input);
   cluster.rateAssignJob(new Job(stk.nextToken(), stk.nextToken(), null),
stk.nextToken());
}



if(command == 8) {
   progress();
}


if(command == 2) {
   print("Enter Job To Kill {ClientName JobName");
   input = br.readLine();
   StringTokenizer stk = new StringTokenizer(input);
   cluster.killJob(new Job(stk.nextToken(), stk.nextToken(), null));
}
```

```
      if(command == 3) {
         print("Enter Job to Lower Priority {ClientName JobName}");
         input = br.readLine();
         StringTokenizer stk = new StringTokenizer(input);
         cluster.lowerJob(new Job(stk.nextToken(), stk.nextToken(), null));
      }


      if(command == 4) {
         print("Enter Job to Raise Priority {ClientName JobName}");
         input = br.readLine();
         StringTokenizer stk = new StringTokenizer(input);
         cluster.raiseJob(new Job(stk.nextToken(), stk.nextToken(), null));
      }


      if(command == 0) {
         print("Shutting Off JMS. GoodBye.");
         System.exit(0);
         break;
      }


      if(command == 1) {
         print("Enter a New Job {ClientName JobName WorkLoad(seconds)} ...");
         input = br.readLine();
         StringTokenizer stk = new StringTokenizer(input);
         cluster.addJob(new Job(stk.nextToken(), stk.nextToken(),
Long.parseLong(stk.nextToken())));
      }


   }while(continued);


   }


   private static void progress() {
      Job[] jobs = cluster.getJobs();
      System.out.println();
      System.out.printf("%-40s %-100s %20s %n", "Client.Job.WorkLoad.Status.Priority",
"Progress", "Percent Complete");
```

```
            for(Job job: jobs) {
                System.out.printf("%-40s %100s %20s %n", job.getJobDetail(), job.getProgress(),
        job.getPercentComplete());
            }
        }


        private static void print(String msg) {
            System.out.println(msg);
        }


        private  static void printMenu() {
            print("");
            print("---------- Cloud JMS System Menu ----------");
            print("0. Exit");
            print("1. Enter a New Job");
            print("2. Kill an existing Job");
            print("3. Lower Job Priority");
            print("4. Raise Job Priority");
            print("5. Pause a Job");
            print("6. Resume a Job");
            print("7. Rate Guarantee A Job");
            print("8. Print Progress");
            print("9. Print Cluster Load");
            print("----------------------------------------------------");
        }
    }
```

Cluster Class which runs in its own Thread (to simulate a Node)

```
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.PriorityBlockingQueue;


/**
* Cluster - runs in its own Thread.
*/
```

```
public class Cluster extends Thread {


   // Number of Nodes in Cluster
   public static int CLUSTER_SIZE = 4;


   private static int index = 0;


   private ExecutorService executorService;
   private List<Node> nodes = new ArrayList<Node>();
   private PriorityBlockingQueue<Job> jobs = new PriorityBlockingQueue<Job>();
   private TaskScheduler taskScheduler = new TaskScheduler();
   private PrintWriter outJobs;
   private PrintWriter outNodes;


   public Cluster() {
      executorService = Executors.newFixedThreadPool(CLUSTER_SIZE);
      setDaemon(true);
      start();
      for(int i=0;i<CLUSTER_SIZE;i++) {
         nodes.add(new Node("Node-" + i, taskScheduler));
         executorService.execute(nodes.get(i));
      }


      try {
         outJobs = new PrintWriter(new FileWriter("cloud_jobs.txt", true));
         outNodes = new PrintWriter(new FileWriter("cloud_nodes.txt", true));
      }catch(Exception x) {
         x.printStackTrace();
      }
      System.out.println("Cluster Initialized with Size - " + CLUSTER_SIZE);
   }


   public synchronized  void raiseJob(Job jobToRaise) {
      if(jobs.contains(jobToRaise)) {
         taskScheduler.raiseJob(jobToRaise);
         Job[] allJobs = getJobs();
         for(Job job: allJobs) {
            if(job.equals(jobToRaise))
               job.raisePriority();
         }
```

```
        } else {
            System.out.println("Job does not exist - " + jobToRaise);
        }
    }


    public synchronized void resumeJob(Job jobToResume) {
        if(jobs.contains(jobToResume)) {
//            taskScheduler.resumeJob(jobToResume);
            Job[] allJobs = getJobs();
            for(Job job: allJobs) {
                if(job.equals(jobToResume))
//                    job.resume();
                    taskScheduler.resumeJob(job);
            }
        }
    }


    public synchronized void rateAssignJob(Job rateJob, String percentRate) {
        Integer rate = Integer.parseInt(percentRate);


        if(rate <= 25) {
            // Assign 1 node
            nodes.get(0).addExclusiveJob(rateJob);
        } else if(rate <= 50) {
            // Assign 2 nodes
            nodes.get(0).addExclusiveJob(rateJob);
            nodes.get(1).addExclusiveJob(rateJob);
        } else if(rate <= 75) {
            // Assign 3 nodes
            nodes.get(0).addExclusiveJob(rateJob);
            nodes.get(1).addExclusiveJob(rateJob);
            nodes.get(2).addExclusiveJob(rateJob);


        } else if(rate <= 100) {
            // Assign 4 nodes.
            nodes.get(0).addExclusiveJob(rateJob);
            nodes.get(1).addExclusiveJob(rateJob);
            nodes.get(2).addExclusiveJob(rateJob);
            nodes.get(3).addExclusiveJob(rateJob);


        }
```

```
}


public synchronized void pauseJob(Job jobToPause) {
        if(jobs.contains(jobToPause)) {
        //taskScheduler.pauseJob(jobToPause);
        Job[] allJobs = getJobs();
        for(Job job: allJobs) {
            if(job.equals(jobToPause))
                taskScheduler.pauseJob(job);
                //job.pause();
        }
    } else {
        System.out.println("Job does not exist - " + jobToPause);
    }
}


public synchronized  void lowerJob(Job jobToLower) {
    if(jobs.contains(jobToLower)) {
        taskScheduler.lowerJob(jobToLower);
        Job[] allJobs = getJobs();
        for(Job job: allJobs) {
            if(job.equals(jobToLower))
                job.lowerPriority();
        }
    } else {
        System.out.println("Job does not exist - " + jobToLower);
    }
}


public synchronized void killJob(Job jobToKill) {
    if(jobs.contains(jobToKill)) {
        taskScheduler.killJob(jobToKill);
        Job[] allJobs = getJobs();
        for(Job job: allJobs) {
            if(job.equals(jobToKill))
                job.kill();
        }
    } else {
        System.out.println("Job does not exist - " + jobToKill);
    }
}
```

```
  public synchronized  void addJob(Job newJob) {
     jobs.add(newJob);
     taskScheduler.addTasks(newJob);
  }


  public Job[] getJobs() {
     return jobs.toArray(new Job[0]);
  }


  public void printLoad() {
     System.out.println();
     System.out.printf("%-50s %70s %20s %n", "Node-Id.Client.JobName.TaskId", "Node Work
Queue", "Percent Complete");


     for(Node node: nodes) {
        System.out.printf("%-50s %70s %20s %n", node.getNodeLoad(),
node.getNodeWorkLoad(), node.getPercentComplete());
     }
  }


  public String getClusterLoad() {
     StringBuffer sb = new StringBuffer();
     for(Node n: nodes) {
        sb.append(n.getNodeLoad());
        sb.append(" ").append(n.getPercentComplete());
        sb.append("\r\n");
     }
     return sb.toString();
  }


  private void trackProgress() {
     if(!jobs.isEmpty()) {
        for(Job job: jobs)
           FileLogger.logger.info(job.getProgress());
     }
  }


  @Override
  public void run() {
     do {
```

```
        try {
           Thread.sleep(250);
           outNodes.printf("%-40s %60s %20s %n", "Node-Id.Client.JobName.TaskId", "Node
Work Queue", "Percent Complete");
           for(Node node: nodes) {
              outNodes.printf("%-40s %60s %20s %n", node.getNodeLoad(),
node.getNodeWorkLoad(), node.getPercentComplete());
           }


           Job[] jobs = getJobs();
           outJobs.printf("%-40s %100s %20s %n", "Client.Job.WorkLoad.Status.Priority",
"Progress", "Percent Complete");


           for(Job job: jobs) {
              outJobs.printf("%-40s %100s %20s %n", job.getJobDetail(), job.getProgress(),
job.getPercentComplete());
           }


        } catch(InterruptedException ix) {
           ix.printStackTrace();
        }
     }while(true);
  }
}
```

Node Class which runs in its own Thread to simulate a Node.

```
import java.util.concurrent.PriorityBlockingQueue;


/**
* Node implemented as a Thread.
*/
public class Node implements Runnable {
  private String name = null;


  // Job which is guaranteed this Node exclusively
  private Job rateGuaranteedJob = null;


  private Task task = null;
```

```
private TaskScheduler taskScheduler;
public Node(String s, TaskScheduler taskScheduler) {
   this.name = s;
   this.taskScheduler = taskScheduler;
}



public void addExclusiveJob(Job job) {
   rateGuaranteedJob = job;
}


public String getNodeLoad() {
   StringBuffer sb = new StringBuffer("{" + name + "}: ");
   if(task != null)
      sb.append(task.getClientName() + "." + task.getJobNumber() + "." + task.getTaskId());
   else
      sb.append(" No Task.");


   return sb.toString();
}


public String getNodeWorkLoad() {
   StringBuffer sb = new StringBuffer();
   if(task != null) {
      long load = Math.round(task.getTaskWorkLoad() * (1-percentComplete/100));
      for(int i=0;i< load;i++)
         sb.append(".");
   }


   return sb.toString();
}


public String getPercentComplete() {
   return String.valueOf(percentComplete) + "%";
}


private float percentComplete = 0;
```

```
  @Override
  public void run() {
     do {
     // Keep working on tasks
        if(null != rateGuaranteedJob)
           task = taskScheduler.nextTask(rateGuaranteedJob);
        else
           task = taskScheduler.nextTask();


        if(null == task) {
           addExclusiveJob(null);
        }


        if(task != null && !task.isComplete()) {
           percentComplete = 0;
           long size = task.getTaskWorkLoad();


           try {
              for(int i=1;i<=size;i++) {
                 Thread.sleep(1000);
                 percentComplete = 100 * i/(size);
              }
              task.setComplete();
           }catch(InterruptedException ix) {
              System.out.println(ix);
           }
        }
     try {
        Thread.sleep(200);
     }catch(InterruptedException ix) {
        ix.printStackTrace();
     }
     }while(true);
  }
}
```

Job Class - this is the Unit of work submitted by Client.

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

```
/**
* Created with IntelliJ IDEA.
* User: dkumawat
* Date: 8/3/13
* Time: 10:51 PM
* To change this template use File | Settings | File Templates.
*/
public class Job implements Comparable<Job> {
  private String clientName;
  private String jobName;
  private Long jobWorkLoad;
  private Status status = Status.NOT_STARTED;
  private Priority priority = Priority.EQUAL;
  private List<Task> jobTasks = new ArrayList<Task>();
  private static Random random = new Random(78938193L);


  public static int TASK_LENGTH = 10; // 10 second tasks


  public Job(String clientName, String jobName, Long jobWorkLoad) {
    this.clientName = clientName;
    this.jobName = jobName;
    if(jobWorkLoad == null) // for non-work jobs, it can be missing
      jobWorkLoad = 0l;
    this.jobWorkLoad = jobWorkLoad;
    // Create Tasks
    long assignedWorkLoad = 0;
    int i = 1;
    for(;assignedWorkLoad<jobWorkLoad;) {
      long nextExpLoad = nextExpDistibutedLoad();
      if(nextExpLoad > (jobWorkLoad - assignedWorkLoad))
        nextExpLoad = jobWorkLoad - assignedWorkLoad;
      Task task = new Task(clientName, jobName, i++, nextExpLoad);
      assignedWorkLoad += task.getTaskWorkLoad();
      jobTasks.add(task);
    }
  }


  /**
   * Return real number uniformly in [0, 1).
   */
  public static double uniform() {
```

```
    return random.nextDouble();
}


/**
 * Return a real number from an exponential distribution with rate lambda.
 */
public static double exp(double lambda) {
    return -Math.log(1 - uniform()) / lambda;
}


public static long nextExpDistibutedLoad() {
    long random = Math.round(10 * exp(1));
    return random > 0 ? random : 1;
}


public static void main(String [] args) throws Exception {
    for(int i=100;i<1000;i++) {
        System.out.println(nextExpDistibutedLoad());
    }
}


public Priority getPriority() {
    return priority;
}


public String getClientName() {
    return clientName;
}


public String getJobName() {
    return jobName;
}


public List<Task> getJobTasks() {
    return jobTasks;
}


public void raisePriority() {
```

```
    if(priority.equals(Priority.LOW))
        priority = Priority.EQUAL;
    else if(priority.equals(Priority.EQUAL))
        priority = Priority.HIGH;
}


public void pause() {
    status = Status.PAUSED;
}


public void resume() {
    status = Status.STARTED;
}


public void lowerPriority() {
    if(priority.equals(Priority.HIGH))
        priority = Priority.EQUAL;
    else if(priority.equals(Priority.EQUAL))
        priority = Priority.LOW;
}


public void kill() {
    status = Status.KILLED;
}


public boolean isComplete() {


    if(status.equals(Status.PAUSED))
        return false;


    if(status.equals(Status.KILLED))
        return true;


    for(Task task: jobTasks) {
        if(!task.isComplete()) {
            status = Status.STARTED;
            return false;
        }
```

```
        }


        status = Status.COMPLETED;
        return true;
    }


    public String getJobDetail() {
        StringBuffer sb = new StringBuffer("[" + clientName + "." + jobName + "." + jobWorkLoad
+
                "." + status.name() + "." + priority.name() +"] ");


        return sb.toString();
    }


    public String getProgress() {
        isComplete();
        int completed = 0;


        int size = jobTasks.size();
        for(Task task: jobTasks)
            if(task.isComplete())
                completed++;


        float percentComplete = 100 * completed/size;



        StringBuffer sb = new StringBuffer();


        int i = 0;
        for(;i<percentComplete;i++)
            sb.append("|");


        for(;i<100;i++)
            sb.append(".");


        return sb.toString();
```

```
  }

  public String getPercentComplete() {
    int size = jobTasks.size();
    int completed = 0;
    for(Task task: jobTasks)
      if(task.isComplete())
        completed++;

    float percentComplete = 100 * completed/size;

    StringBuffer sb = new StringBuffer();
    if(status.equals(Status.KILLED) || status.equals(Status.COMPLETED) ||
status.equals(Status.PAUSED))
      sb.append(status.name());
    else
      sb.append(" " + percentComplete + "%");

    return sb.toString();
  }

  @Override
  public int compareTo(Job other) {

    if(this.clientName.equals(other.clientName) && this.jobName.equals(other.jobName))
      return 0;

    return 1;
  }

  @Override
  public boolean equals(Object o) {
    Job other = (Job) o;

    return this.clientName.equals(other.clientName) && this.jobName.equals(other.jobName);
  }
```

```
  @Override
  public String toString() {
     return "Job{" +
           "clientName='" + clientName + '\'' +
           ", jobName='" + jobName + '\'' +
           '}';
  }


  public int hashCode() {
     return clientName.hashCode() + jobName.hashCode();
  }
}
```

A Task Class. A Single Job is divided into 1 or more Tasks using Exponential Distribution. Each Job is scheduled with a Node for completion.

```
/**
* Created with IntelliJ IDEA.
* User: dkumawat
* Date: 8/3/13
* Time: 2:09 PM
* To change this template use File | Settings | File Templates.
*/
public class Task implements Comparable<Task> {
  private String clientName;
  private String jobNumber;
  private Long taskWorkLoad;
  private Integer taskId;
  private Status status = Status.NOT_STARTED;


  public Task(String clientName, String jobNumber, int taskId, Long taskWorkLoad) {
     this.clientName = clientName;
     this.jobNumber = jobNumber;
     this.taskId = taskId;
     this.taskWorkLoad = taskWorkLoad;
  }


  public Status getStatus() {
```

```
    return status;
  }

  public void setComplete() {
    status = Status.COMPLETED;
  }

  public boolean isComplete() {
    return status.equals(Status.COMPLETED);
  }

  public String getClientName() {
    return clientName;
  }

  public String getJobNumber() {
    return jobNumber;
  }

  public Integer getTaskId() {
    return taskId;
  }

  public Long getTaskWorkLoad() {
    return taskWorkLoad;
  }

  @Override
  public boolean equals(Object o) {
    Task other = (Task) o;

    return this.clientName.equals(other.clientName) &&
this.jobNumber.equals(other.jobNumber)
        && this.taskId == other.taskId;
  }
```

```
  @Override
  public int compareTo(Task other) {


     if(this.clientName.equals(other.clientName) && this.jobNumber.equals(other.jobNumber)
           && this.taskId == other.taskId)
        return 0;


     return 1;
  }


  public int hashCode() {
     return clientName.hashCode() + jobNumber.hashCode() + taskId.hashCode();
  }


}
```

Core Controller Logic for the JMS Scheduler. Scheduler maintains all Data Structures for active jobs, tasks, paused, running and all States. Each Node asks TaskScheduler for next Task to execute.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.PriorityBlockingQueue;


/**
* Main Task Scheduler
* Maintains a Queue for Each Client. Ensures Fair Task assignment for Task Scheduling.
*/
public class TaskScheduler {
  private Map<String, PriorityBlockingQueue<Task>> mapClientToTaskList = new
HashMap<String, PriorityBlockingQueue<Task>>();
  private List<String> keys = new ArrayList<String>();
  private Map<String, PriorityBlockingQueue<Task>> pausedJobs = new HashMap<String,
PriorityBlockingQueue<Task>>();
  private Map<String, Priority> priorityMap =  new HashMap<String, Priority>();
  private Map<Priority, List<String>> jobsByPriorityMap = new HashMap<Priority,
List<String>>();
```

```
private static int index = 0;


public TaskScheduler(){
   jobsByPriorityMap.put(Priority.HIGH, new ArrayList<String>());
   jobsByPriorityMap.put(Priority.EQUAL, new ArrayList<String>());
   jobsByPriorityMap.put(Priority.LOW, new ArrayList<String>());
}


/**
 * Bump up all Tasks to front of the work Q
 * @param job
 */
public synchronized  void raiseJob(Job job) {
   // can't change priority for a paused job
   if(pausedJobs.containsKey(getKey(job)))
      return;


   Priority priority = priorityMap.get(getKey(job));
   if(priority.equals(Priority.HIGH)) {
      // can't go higher then that
   } else if(priority.equals(Priority.EQUAL)) {
      priorityMap.remove(getKey(job));
      priorityMap.put(getKey(job), Priority.HIGH);
      jobsByPriorityMap.get(Priority.EQUAL).remove(getKey(job));
      jobsByPriorityMap.get(Priority.HIGH).add(getKey(job));
   } else if(priority.equals(Priority.LOW)) {
      priorityMap.remove(getKey(job));
      priorityMap.put(getKey(job), Priority.EQUAL);
      jobsByPriorityMap.get(Priority.LOW).remove(getKey(job));
      jobsByPriorityMap.get(Priority.EQUAL).add(getKey(job));
   }
}


public synchronized void pauseJob(Job job) {
   pausedJobs.put(getKey(job), mapClientToTaskList.get(getKey(job)));
   mapClientToTaskList.remove(getKey(job));
   priorityMap.remove(getKey(job));
   jobsByPriorityMap.get(job.getPriority()).remove(getKey(job));
   job.pause();
}
```

```
public synchronized void resumeJob(Job job) {
   mapClientToTaskList.put(getKey(job), pausedJobs.get(getKey(job)));
   pausedJobs.remove(getKey(job));
   priorityMap.put(getKey(job), job.getPriority());
   jobsByPriorityMap.get(job.getPriority()).add(getKey(job));
   job.resume();
}


/**
 * Take all the tasks for job to back of the q
 * @param job
 */
public synchronized void lowerJob(Job job) {
   // can't change priority for a paused job
   if(pausedJobs.containsKey(getKey(job)))
      return;


   Priority priority = priorityMap.get(getKey(job));
   if(priority.equals(Priority.HIGH)) {
      priorityMap.remove(getKey(job));
      priorityMap.put(getKey(job), Priority.EQUAL);
      jobsByPriorityMap.get(Priority.HIGH).remove(getKey(job));
      jobsByPriorityMap.get(Priority.EQUAL).add(getKey(job));
   } else if(priority.equals(Priority.EQUAL)) {
      priorityMap.remove(getKey(job));
      priorityMap.put(getKey(job), Priority.LOW);
      jobsByPriorityMap.get(Priority.EQUAL).remove(getKey(job));
      jobsByPriorityMap.get(Priority.LOW).add(getKey(job));
   } else if(priority.equals(Priority.LOW)) {
      // can't go lower
   }
}


public synchronized  void killJob(Job job) {
   mapClientToTaskList.remove(getKey(job));
   keys.remove(getKey(job));
   index--;
}


public synchronized  void addTasks(Job job) {
```

```
    if(mapClientToTaskList.containsKey(getKey(job))) {
        // Add to the same Entry
        mapClientToTaskList.get(getKey(job)).addAll(job.getJobTasks());
    } else {
        PriorityBlockingQueue<Task> newTaskList = new PriorityBlockingQueue<Task>();
        newTaskList.addAll(job.getJobTasks());
        mapClientToTaskList.put(getKey(job), newTaskList);
        keys.add(getKey(job));
        priorityMap.put(getKey(job), job.getPriority());
        jobsByPriorityMap.get(job.getPriority()).add(getKey(job));
    }
}


public synchronized Task nextTask(Job job) {


    if(null == job)
        return null;


    String key = getKey(job);


    // Get task list
    PriorityBlockingQueue<Task> taskList = mapClientToTaskList.get(key);


    if(taskList == null || taskList.isEmpty()) {
        // Completed.
        keys.remove(key);
        index--;
        mapClientToTaskList.remove(key);
        // remove from priority maps
        Priority priority = priorityMap.get(key);
        if(priority != null)
            jobsByPriorityMap.get(priority).remove(key);
        priorityMap.remove(key);
        return null;
    }


    // Remove the task
    return taskList.remove();
}
```

```
/**
 * Return the new Task in a fair Manner.
 * @return next Task
 */
public synchronized Task nextTask() {
   if(mapClientToTaskList.isEmpty())
      return null;


   if(index < 0)
      index = 0;


   String key;
   if(jobsByPriorityMap.get(Priority.HIGH).size() > 0) {
      int serialEnd = jobsByPriorityMap.get(Priority.HIGH).size();
      if(index >= serialEnd)
         index = 0;


      key = jobsByPriorityMap.get(Priority.HIGH).get(index++);
   } else if(jobsByPriorityMap.get(Priority.EQUAL).size() > 0) {
      int serialEnd = jobsByPriorityMap.get(Priority.EQUAL).size();
      if(index >= serialEnd)
         index = 0;


      key = jobsByPriorityMap.get(Priority.EQUAL).get(index++);
   } else if(jobsByPriorityMap.get(Priority.LOW).size() > 0) {
      int serialEnd = jobsByPriorityMap.get(Priority.LOW).size();
      if(index >= serialEnd)
         index = 0;


      key = jobsByPriorityMap.get(Priority.LOW).get(index++);
   } else {
      return null;
   }


   // Get task list
   PriorityBlockingQueue<Task> taskList = mapClientToTaskList.get(key);
```

```
      if(taskList == null || taskList.isEmpty()) {
         // Completed.
         keys.remove(key);
         index--;
         mapClientToTaskList.remove(key);
         // remove from priority maps
         Priority priority = priorityMap.get(key);
         if(priority != null)
            jobsByPriorityMap.get(priority).remove(key);
         priorityMap.remove(key);
         return null;
      }


      // Remove the task
      return taskList.remove();
   }



   private String getKey(Job job) {
      return                                                                                    new
StringBuffer().append(job.getClientName()).append(".").append(job.getJobName()).toString();
   }
}
```

Code for JMS web service:

```
package web.jms;
/**
 * Created by Eclipse
 * User: Ryda Nofal
 * Date: 8/20/13
 * Time: 11:40 PM
 * This class handles submit jobs.
 */

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.QueryParam;


@Path("/submit")
public class WebJMSHandler {
```

```
        Cluster cluster = WebMain.instance.getCluster() ;

         public boolean isNumeric(String str)
         //Checks if a string is an integer number
         {
                  return str.matches("\\d+");  //match a number
         }
         @GET
         //@Produces(MediaType.TEXT_PLAIN)
         @Produces(MediaType.TEXT_HTML)
         public String sumbitNewJob(
                  @DefaultValue("") @QueryParam("client") String clientName,
                  @DefaultValue("") @QueryParam("job") String jobName,
                  @DefaultValue("") @QueryParam("load") String workLoad)
                  {
                   String str = genJobSubInfo(clientName, jobName, workLoad) ;
                   //System.out.println(str) ;
                   //cluster.addJob(new Job(clientName, jobName, Long.parseLong(workLoad)));
                   if (clientName.length()==0) {
                            str += "<h1>Unable to submit a job Missing Client Name<h1>" ;
                            return str ;
                   } else {
                            if (jobName.length()==0) {
                                     str += "<h1>Unable to submit a job Missing Job Name<h1>" ;
                                     return str ;
                            } else {
                                     if (workLoad.length()==0) {
                                              str += "<h1>Unable to submit a job Missing
workLoad<h1>" ;
                                              return str ;
                                     } else {
                                              if (! isNumeric(workLoad)) {
                                                       str += "<h1>Unable to submit a job work Load is
not an integer value<h1>" ;
                                                       return str ;
                                              }
                                     }
                            }
                   }
                   System.out.println("clientName="+ clientName + " JobName=" + jobName +"
workLoad=" + workLoad) ;
                  cluster.addJob(new Job(clientName, jobName, Long.parseLong(workLoad)));
                  str += "<h1>The above job has been submitted successfully  <h1>" ;
                  return str ;
         }
```

```
        @GET
        @Path("progress")
        @Produces(MediaType.TEXT_HTML)
        public String progress() {
            Job[] jobs = WebMain.instance.getCluster().getJobs();
            String str = new String() ;
            System.out.println();
            System.out.printf("%-40s %-100s %20s %n",
"Client.Job.WorkLoad.Status.Priority", "Progress", "Percent Complete");
            str = "Client.Job.WorkLoad.Status.Priority, Progress Percent Complete +\n" ;
            for(Job job: jobs) {
               str += job.getJobDetail() + job.getProgress()+ job.getPercentComplete() +"\n";
               System.out.printf("%-40s %100s %20s %n", job.getJobDetail(),
job.getProgress(), job.getPercentComplete());
            }
            return str ;
          }
        public String genJobSubInfo(String client, String job, String load){
              String str = "<html><head><title>Submit a new Job</title></head><body>" +
                                  "<h1>New Job Submission Info</h1>" +
                "<ul><li> " +
                      "<span style=\"font-size:24px;\">Client Name: <span style=\"background-
color:#afeeee;\">" +
                      client + "</span></span></li>" +
                "<li><span style=\"font-size:24px;\">Job   Name:  <span style=\"background-
color:#dda0dd;\">" +
                      job + "</span></span></li>" +
                 "<li><span style=\"font-size:24px;\">Work  Load: <span style=\"background-
color:#ff8c00;\">" +
                      load + "</span></span></li>" +
                "</ul><p> </p></body></html>" ;
              return str ;

        }
}

package web.jms;
/**
 * Created by Eclipse
 * User: Ryda Nofal
 * Date: 8/23/13
 * Time: 11:40 PM
 * This is a Singleton to handle web requests.
 */
public enum WebMain {
        instance ;
```

```
        private Cluster cluster = new Cluster();
        private WebMain() {
                //Cluster cluster = new Cluster() ;
        }
        public Cluster getCluster(){
                return cluster ;
        }
}
```

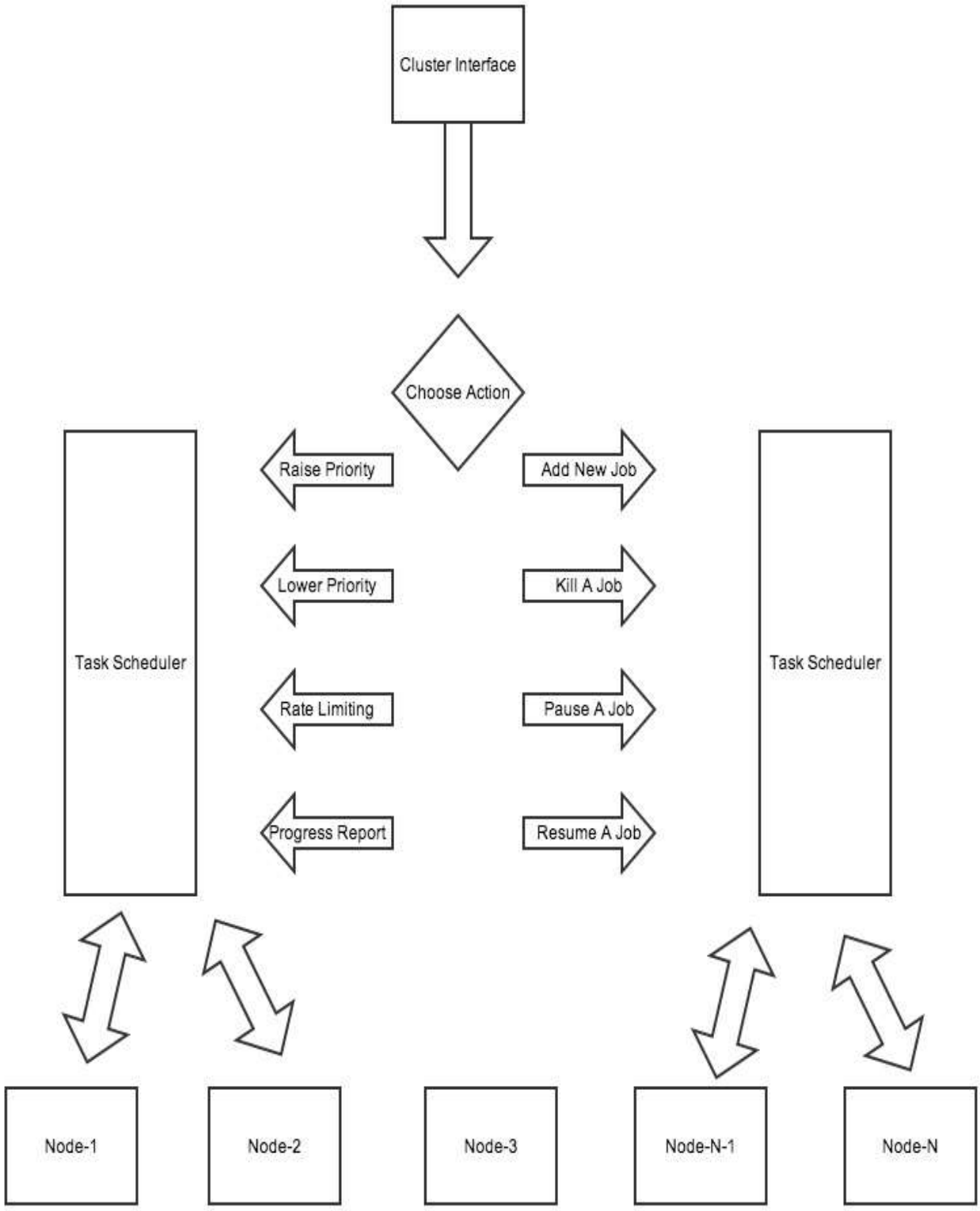
Application to application:

Python script:

```
import urllib.request

import time

hostName = "http://rnofal.dyndns-home.com:8080/web.jms/rest/command/"

urlAddress= hostName

startCommand  = "start"

submitCommand1 = "submit?client=%s&job=%s&load=%s"%("Ryda","PythonAAA",400)

submitCommand2 = "submit?client=%s&job=%s&load=%s"%("Ryda","PythonBBB",400)

submitCommand3 = "submit?client=%s&job=%s&load=%s"%("Ryda","PythonCCC",400)

killCommand1 = "kill?client=%s&job=%s"%("Ryda","PythonCCC")

killCommand2 = "kill?client=%s&job=%s"%("Ryda","PythonBBB")

#Sending Start command

data = urllib.request.urlopen(urlAddress+startCommand)

#Sending 3 Submit commands

data1 = urllib.request.urlopen(urlAddress+submitCommand1)

data2 = urllib.request.urlopen(urlAddress+submitCommand2)

data3 = urllib.request.urlopen(urlAddress+submitCommand3)

#Let's sleep for 40 second

time.sleep(40)

#Then kill job #3

data4 = urllib.request.urlopen(urlAddress+killCommand1)

#Let's sleep for another 20 second

time.sleep(20)
```

```
#Then kill Job #2
data5 = urllib.request.urlopen(urlAddress+killCommand2)
print ("test is Done :)")
```

Flowchart:


Cluster Interface
Choose Action
Raise Priority
Add New Job
Lower Priority
Kill A Job
Rate Limiting
Pause A Job
Progress Report
Resume A Job
Task Scheduler
Task Scheduler
Node-1
Node-2
Node-3
Node-N-1
Node-N

---

7) Data analysis and discussion:

- **Output generation:**
  - Output is generated in 2 Formats.
    - Cluster (Jobs) Output.

```
Client.Job.WorkLoad.Status.Priority                                    Progress    Pe
rcent Complete
[A.aaa.1200.STARTED.EQUAL]          ||||||||.....................................................
                                    9.0%
[B.bbb.1000.STARTED.EQUAL]          ||||.....................................................
                                     4.0%
[C.ccc.400.STARTED.EQUAL]           |||.....................................................
                                    3.0%
```

    - Nodes Output

Node-Id.Client.JobName.TaskId	Node Work Queue	Percent Complete
{Node-0}: B.bbb.93	.....	26.0%
{Node-1}: B.bbb.92	.	66.0%
{Node-2}: A.aaa.88	.....	6.0%
{Node-3}: C.ccc.22	.....	43.0%

- **Output analysis:**
  - Jobs output shows each submitted Job with following details:
    - Client Job: Client Name and Job Name
    - Status: Job Status (Started, Paused, Killed, Completed)
    - Priority: Job Priority (Low, High, Equal)
    - Progress: Moving Bars and Dots show a Jobs Progress.
    - Percent Complete: Percent Completion for the Job.
  - Nodes Output shows what Each Node is doing with following details:
    - Node-Id: Each Node in the Cluster has an Id.
    - Client Job Name: Client and Job Name a Node is handling currently
    - TaskId: Exponentially Distributed Task Id.
    - Work Queue: Dot represented work Queue of Node.
    - Percent Complete: Completion of the task Node is currently executing.

- **Compare output against hypothesis:**
  - Default Hadoop Scheduler uses FIFO Policy to schedule Client Jobs. This policy is not "fair" and can starve smaller jobs submitted after a Large Job. JMS Scheduler solves this issue by using "Fair" scheduling using Exponential Distribution of a Job and scheduling each Task based on its priority and using even distribution for equal priority Tasks.
- **Abnormal case explanation (the most important task):**
  - Exponential Distribution of Job into Tasks is not always optimal.

- A Job Workload is specified in Seconds at Job entry time - this may not always be the case in reality where workload cannot be accurately measured or specified.

- **Discussion:**
  - It will be interesting to compare throughput and response time using other Distribution.

---

8) Conclusions:

- **Summary and conclusions**
  - Default Hadoop Scheduler is very limiting in its approach and real usage.
  - If enhanced by using Fair Scheduling mechanism using Priority, Hadoop can be used for real time production systems supporting many concurrent Users and Jobs.
  - Relatively fewer Nodes in the Cluster can provide Service to a large number of Clients.
  - Fault Tolerance and High Availability as provided by Hadoop are not affected by overriding Scheduling Policies and Implementations.
  - JMS Scheduler can be extended to provide more detailed Analysis and Monitoring of a Job's progress, current Status and Completion Time.

---

9) Bibliography:

1) Cloud Control with Distributed Rate Limiting, Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, and Alex C. Snoeren, Department of Computer Science and Engineering, University of California, San Diego.

2) Adaptive Cluster Throttling: Improving High-Load Performance in Bufferless On-Chip Networks, Rachata Ausavarungnirun, Kevin Kai-Wei Chang, Chris Fallin, Onur Mutlu, Computer Architecture Lab (CALCM), Carnegie Mellon University.

3) Job-Oriented Monitoring of Clusters, Vijayalaxmi Cigala, Dhirajkumar Mahale, Monil Shah, Sukhada Bhingarkar, Computer Engineering Department, MIT College of Engineering, Pune.

4) Multi-scale Real-time Grid Monitoring with Job Stream Mining, Xiangliang Zhang, Michele Sebag, Cecile Germain-Renaud, Universite Paris-Sud 11, F-91405 Orsay, France.

5) In Search for Contention-Descriptive Metrics in HPC Cluster Environment, Sergey Blagodurov, Alexandra Fedorova, Systems Research Lab, Simon Fraser University.

6) High-performance Linux cluster monitoring using Java, Curtis Smith, David Henry, Linux NetworX, Inc.
