Fix on HTTP Server Threading Policy based on Number of Cores

In the Eclipse Jetty Server package, the Jetty Server class (org.eclipse.jetty.server.Server.java) is involved with the class called SelectChannelConnector.java. The number of the acceptors in SelectChannelConnector.java is computed as follows:

```
public SelectChannelConnector() {
  …..
 setAcceptors(Math.max(1, (Runtime.getRuntime().availableProcessors() + 3))
  …..
}
```

Runtime.getRuntime().availableProcessors() returns the total number of the cores on the entire machine.

The above constructor is called by JettyUtils.scala class directly. The same constructor is called by RestSubmissionServer.scala via org.eclipse.jetty.server.Server.java. As a result, the total number of the server threads launched in a Spark Master or a Spark Executor is large, when such a Spark process is launched in a multicore machine with large number of the cores (for example, in a machine with 80 cores or 240 cores).

In reality, each process is bound to a specific socket of the multicore NUMA-aware machine, and thus the number of the server threads should be only proportional to the number of the cores on a socket, not to the total number of the cores on the entire machine.

The proposed code fix for both JettyUtils.scala and RestSubmissionServer.scala is to introduce a configuration parameter called: `spark.jetty.httpserver.numThreads`, to allow the Spark application to set the maximum thread number in spark-defaults.conf. By default, this number is set to 10. This is similar to what has been implemented in Netty Dispatcher class to set the netty dispatcher thread number via the configuration parameter: `spark.rpc.netty.dispatcher.numThreads`.

The two modified files from Spark 2.0: JettyUtils.scala and RestSubmissionServer.scala, are put in the directory called “threading-fixed”. The user can replace the corresponding files in Spark 2.0 with these two files before making source code compilation.