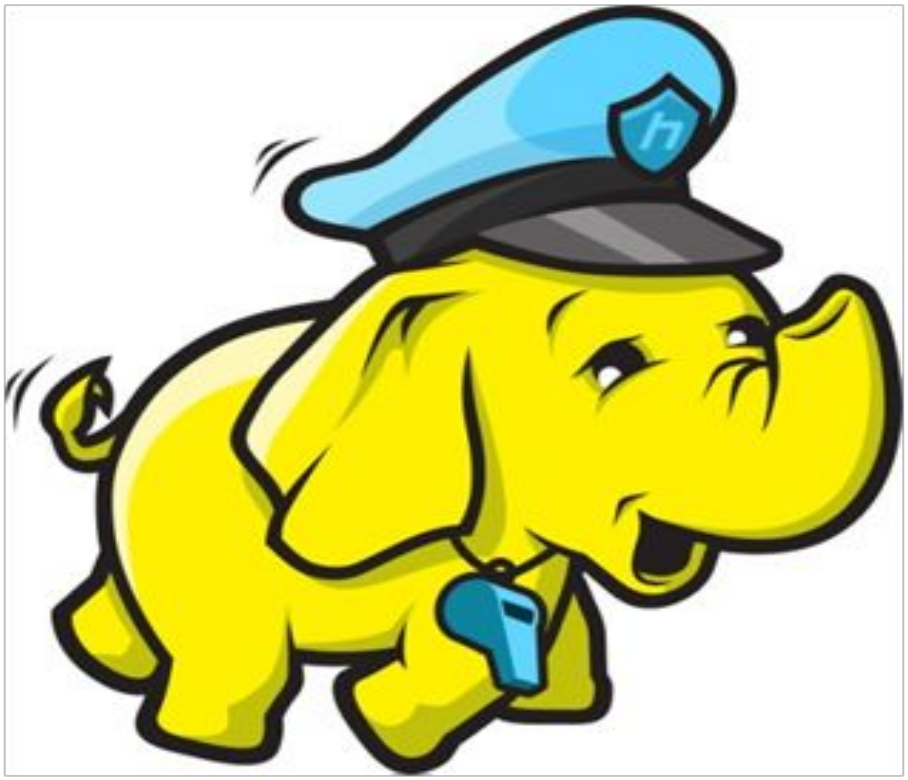
Map-Reduce for Searching & Sorting



Agenda

Searching using Map-Reduce
Mapper code for searching
Reducer code for searching
Main method for searching
Execute Searching Map-Reduce job
Sorting using Map-Reduce
Mapper code for sorting
Reducer code for sorting
Main method for sorting
Execute Sorting Map-Reduce job

Searching using Map-Reduce

A set of files, containing lines of text.

A search pattern to find (find out the word which starts with “N” and ends with “J”).

Mapper key is line number in file.

Mapper value is the line’s content.

Search pattern is taken from the user.

Map-Reduce program searches the word.

Mapper for searching

```
public static class SearchingMapper extends Mapper<Object, Text, Text, IntWritable>
{
        String search_pattern = "";
        private Text word = new Text();
        private final static IntWritable one = new IntWritable(1);

        public void map(Object key, Text value, Context context) throws
        IOException, InterruptedException
        {
                search_pattern = context.getConfiguration().get("search_word");
                StringTokenizer itr = new StringTokenizer(value.toString());
                String token;
                while (itr.hasMoreTokens())
                {
                        token = itr.nextToken();
                        if (search_pattern.equals(token))
                        {
                                word.set(token);
                                context.write(word, one);
                        }
                }
        }
}
```

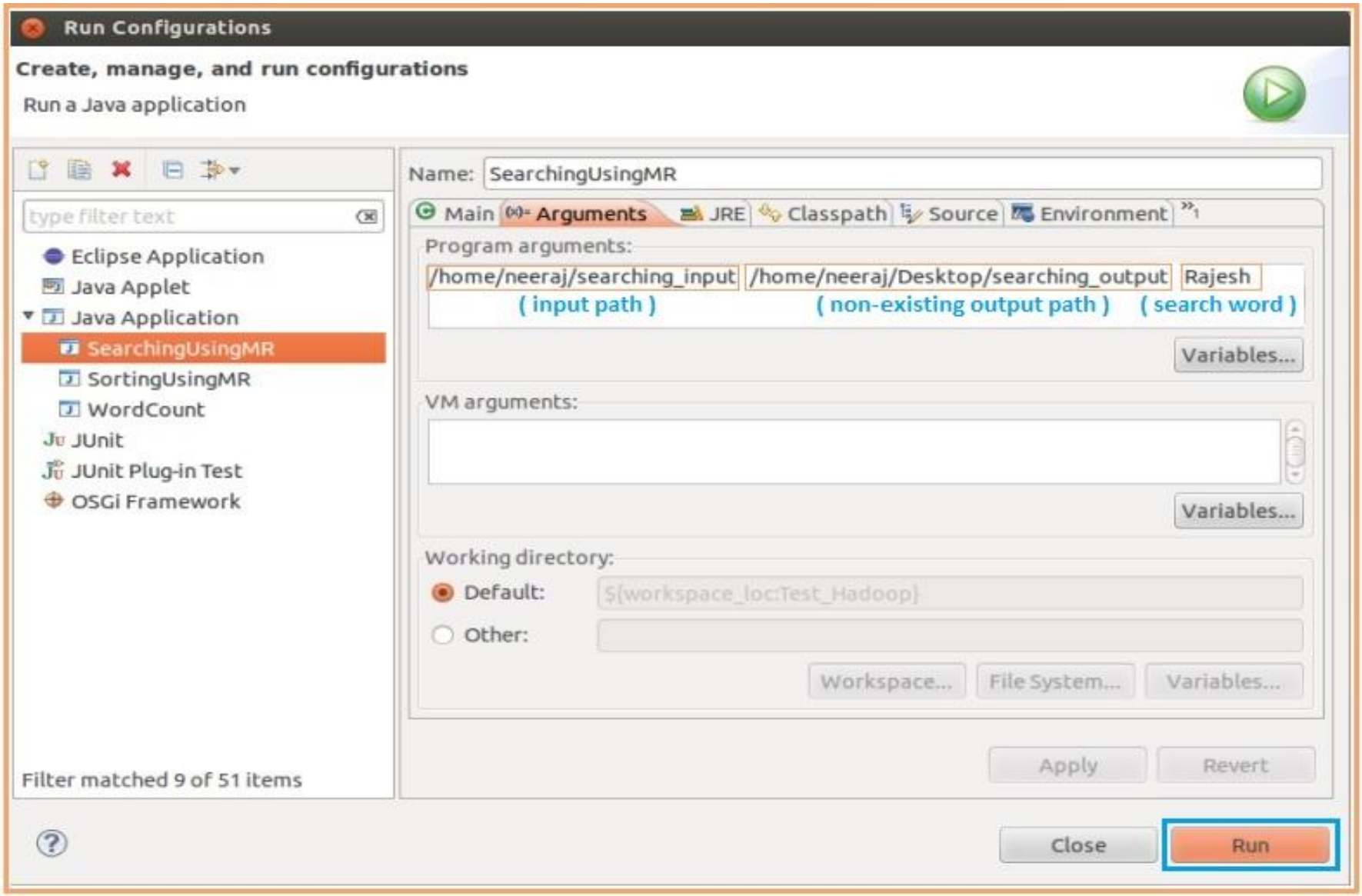
Reducer for searching

```
public static class SearchingReducer extends Reducer<Text, IntWritable, Text, IntWritable>
{

        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
        IOException, InterruptedException
        {
                int sum = 0;

                for (IntWritable val : values)
                {
                        sum = sum + val.get();
                }

                result.set(sum);
                context.write(key, result);
        }
}
```

Main method for searching

```
public static void main(String[] args) throws Exception
{
        Configuration conf = new Configuration();
        if (args.length != 3)
        {
                System.err.println("Usage: SearchingUsingMR <input path> <output path>
                                        <search word>");
                System.exit(2);
        }

        conf.set("search_word", args[2]);

        Job job = new Job(conf, "Searching Using MR");
        job.setJarByClass(SearchingUsingMR.class);
        job.setMapperClass(SearchingMapper.class);
        job.setReducerClass(SearchingReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
}
```

Execute Searching Map-Reduce job



Sorting using Map-Reduce

A set of files, containing lines of text.

Sorting millions of rows is a complex process.

Map-Reduce framework can be used for sorting.

Mapper key is line number in the file.

Mapper value is the line’s content.

Mapper's output is sorted before it's sent to reducer.

Single reducer is used to generate sorted output.

We can use **IdentityReducer** for sorting.

Mapper for sorting

```
public static class SortingMapper extends Mapper<Object, Text, Text,
NullWritable>
{

        public void map(Object key, Text value, Context context) throws
        IOException, InterruptedException
        {

                context.write(value, NullWritable.get());
        }

}
```

Reducer for sorting

If you need sorted output, but don't need any aggregation - **Use 1 identity reducer (** in-built in Hadoop **).**

Single reducer merges mapper's intermidiate output & generate sorted output.

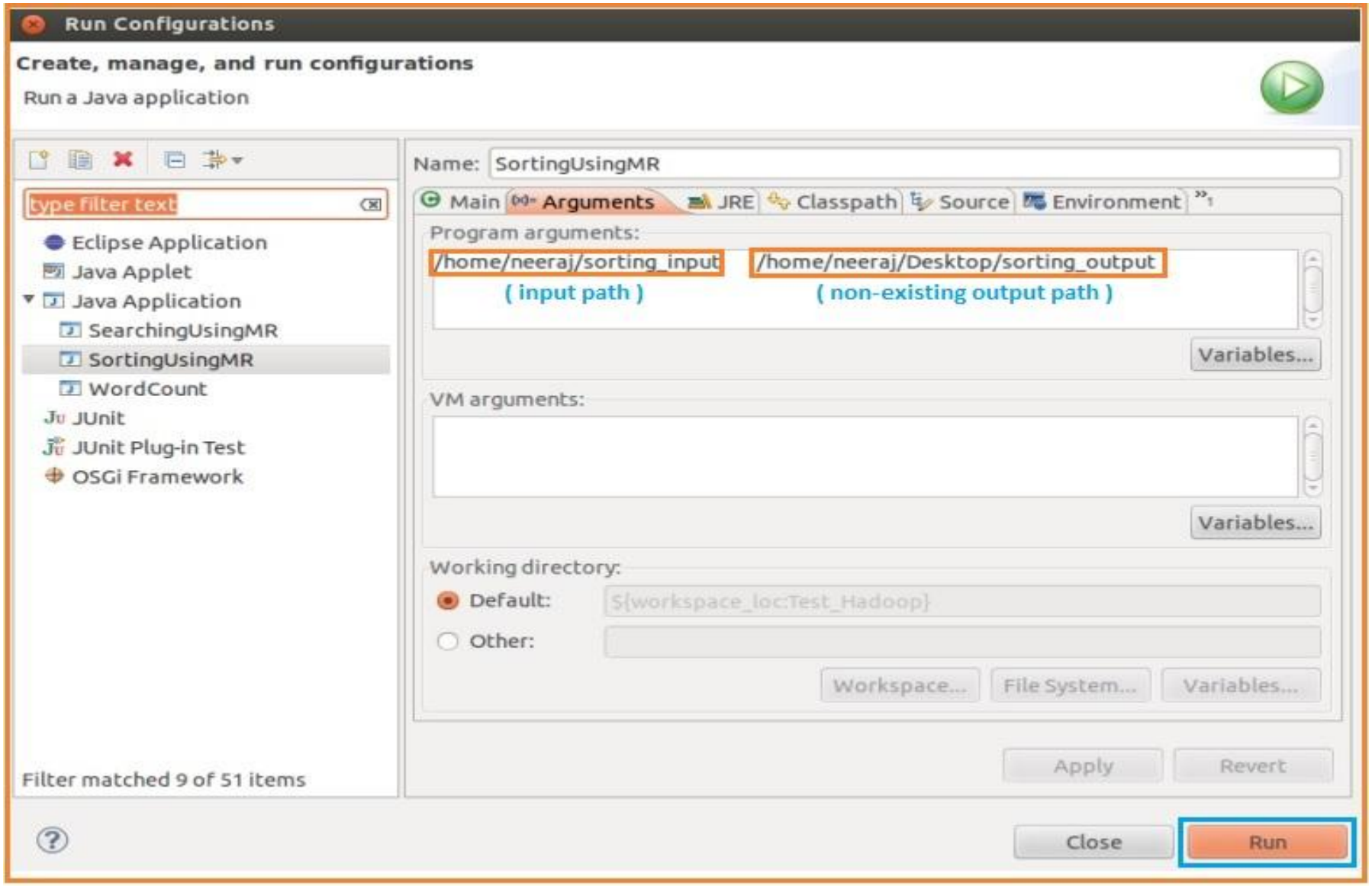
If you need sorted output, and do need aggregation - **Use user defined 1 reducer.**

If you don't need sorted output, and don't need any aggregation - **Set 0 reducer, and the job is called map only job.**

Main method for sorting

```
public static void main(String[] args) throws Exception
{
        Configuration conf = new Configuration();

        if (args.length != 2)
        {
                System.err.println("Usage: SortingUsingMR <input path> <output path> ");
                System.exit(2);
        }

        Job job = new Job(conf, "Sorting using MR");
        job.setJarByClass(SortingUsingMR.class);
        job.setMapperClass(SortingMapper.class);
        job.setReducerClass(IdentityReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
}
```

Execute Sorting Map-Reduce job



…Thanks…