RDD commands

plan and (key,value)

Three groups of commands

- **Creation:** RDD from files, databases, or data on driver node.
- **Transformations:** RDD to RDD
- **Actions:** RDD to data on driver node, databases, files.

Plain RDD
Transformations

```
## Initialize RDD
A=sc.parallelize(range(4))

## map
B=A.map(lambda x: x-2)
B.collect()
```

```
# output:
[-2,-1,0,1]
```

```
## Initialize RDD
A=sc.parallelize(range(4))

## map
B=A.filter(lambda x: x%2==0)
B.collect()
```

```
# output:
[0,2]
```

```
## Initialize RDD
A=sc.parallelize(range(4))

## map
B=A.map(lambda x: x-2)
B.collect()
```

```
# output:
[-2,-1,0,1]
```

Plain RDD

Actions

```
sc.parallelize(range(4)).collect()
```

```
# output:
[0,1,2,3]
```

```
## Initialize RDD
A=sc.parallelize(range(4))

## reduce
A.reduce(lambda x,y: x+y)
```

```
# output:
6
```

(key,Value) RDDs

Transformations

(key,Value) RDDs

Actions

Partitioners

Spark Partitioners

- Each RDD is divided into partitions.
  - One partition per worker (core)
  - A single partition can be operated on as a regular python list.
- After manipulations (such as **filter()**) some partitions can shrink to zero and some might be very large
  - This means that future work is not balanced across the workers.
- If RDD consists of (key,value) pairs we can use a **partitioner** to redistribute the items among the workers

Types of partitioners

- **HashPartitioner(n):** divide the keys into **n** groups at random. Divide the pairs according to their keys
- **RangePartitioner(n):** each partition corresponds to a range of key values, so that each range contains approximately the same number of items (keys).
- **Custom Partitioner:**
  define a partitioner that maps key **K** to integer **I**.
  n= number of partitions.
  pair with key **K** placed in partition **I mod n**

Custom Partition Example

In [10]:
```
data = sc.parallelize(['1', '2', '3', '4', '5']).map(lambda x: (x,x))
print data.collect()
c = data.count()
wp = data.partitionBy(c/2,lambda k: int(k))
print wp.map(lambda t: t[0]).glom().collect()
```

```
[('1', '1'), ('2', '2'), ('3', '3'), ('4', '4'), ('5', '5')]
[['2', '4'], ['1', '3', '5']]
```

Creation

- sc.parallelize(range(10000))
- Parsing text file
- Sources of files: local, S3, Web
- SparkSQL and Parquet files