JUDCon

JBoss Users & Developers Conference

Boston:2011

MapReduce on The Cloud: Infinispan Distributed Task Execution Framework

Vladimir Blagojević
Sr. Software Engineer, Red Hat

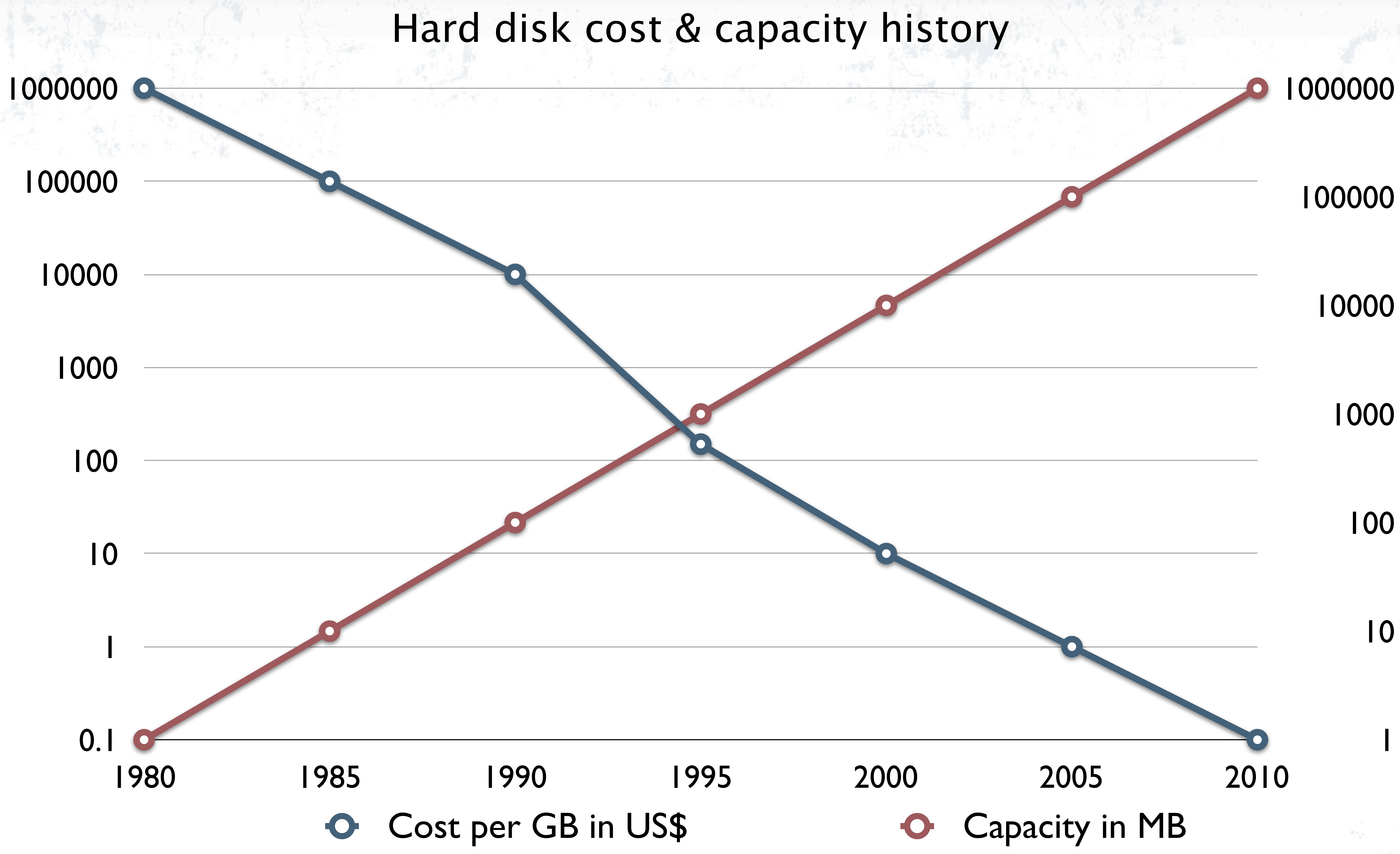
May 3rd 2011, JUDCon - Boston

Background

- Emergence of data beyond human scale
- Outgrows current platforms in scale, structure, and processing time
- Abundance of unstructured, machine generated data
- Does not fit into current software paradigms
- Not confined to Twitter, Facebook and Google only
- Need new platforms built for Big Data from ground up

Big Data

- How did Big Data happen?
- Exponential changes in storage, bandwidth and data creation
- Data exhaust
- Drowning in data and not knowing what to do with it
- Leverage, not discard Big Data
- We are not even aware of data revolution around us



Source: http://en.wikipedia.org/wiki/History_of_hard_disk_drives

”What we are seeing is the ability to have economies form around the data, and that to me is the big change at a societal and even macroeconomic level.”

“You would not just think of data as the ‘exhaust’ of providing health services, but rather they become a central asset in trying to figure out how you would improve every aspect of health care. It’s a bit of an inversion.”

Craig Mundie, Microsoft Research

Source: http://www.economist.com/node/15557443?story_id=15557443

“There was 5 exabytes of information created between the dawn of civilization through 2003....

... that much information is now created every two days”

Eric Schmidt, August 4th, 2010

Source: http://www.readwriteweb.com/archives/google_ceo_schmidt_people_arent_ready_for_the_tech.php

Big Data challenges

- Powerful platforms available today, however...
- Need to be genius to build systems on top of them
- Equivalent to programming client/server apps in assembly language
- Is there a need for a new language?
- Confusion is as big as Big Data
- Simpler yet equally powerful solutions needed

Infinispan Big Data goals

- Why waste a Ferrari?
- Humble beginnings
- Simplicity without sacrificing power
- Capitalize on existing Infinispan infrastructure
- Reuse of current programming abstractions
- Two frameworks: distributed executors and MapReduce

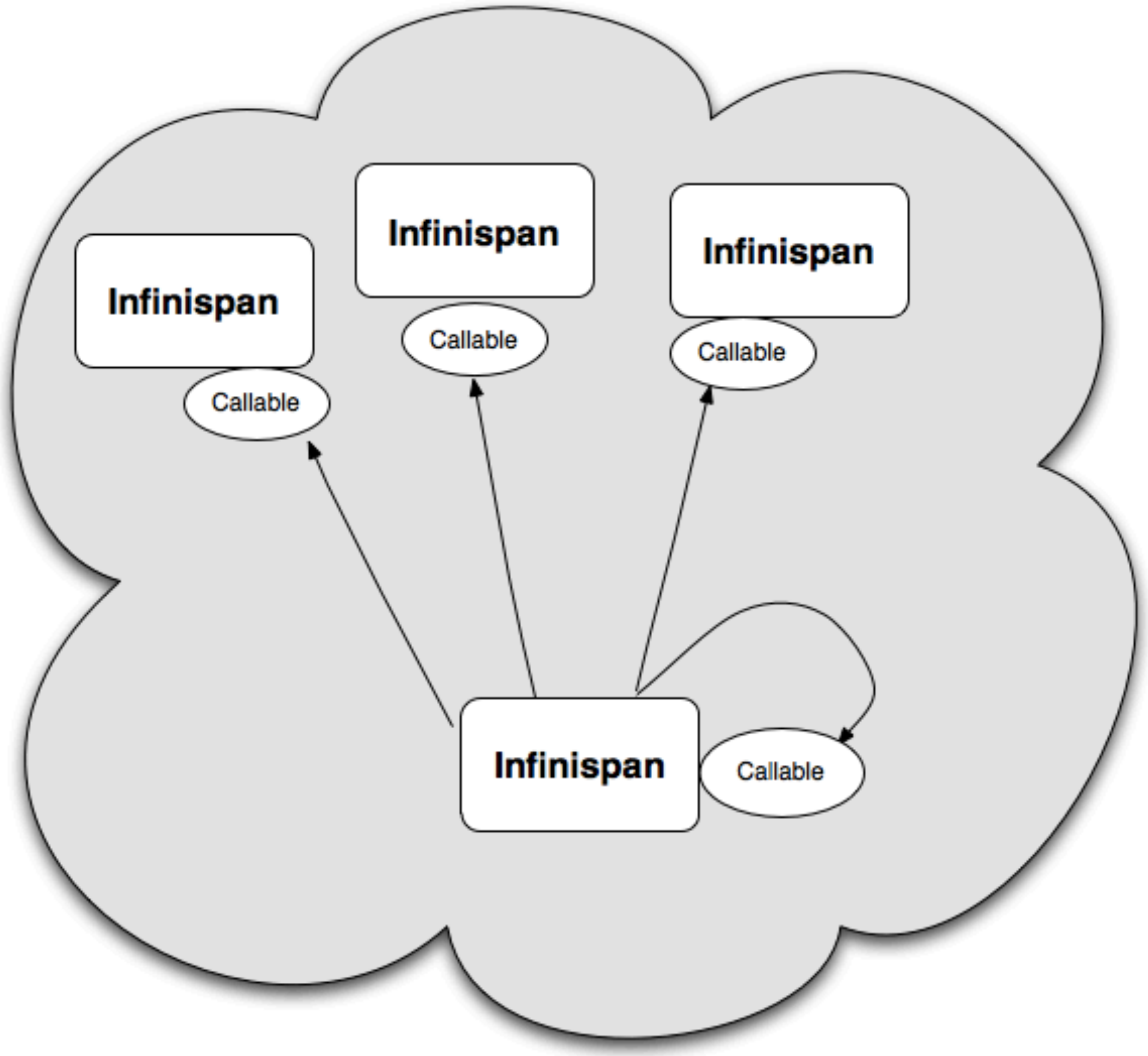
Infinispan Distributed Execution Framework

- Leverage familiar ExecutorService, Callable abstractions
- Expand it to distributed, parallel computing paradigm
- Looks like a regular ExecutorService
- Feels like a regular ExecutorService
- The magic that goes on Infinispan grid is completely transparent to users
- Nevertheless, users can experience it :-)

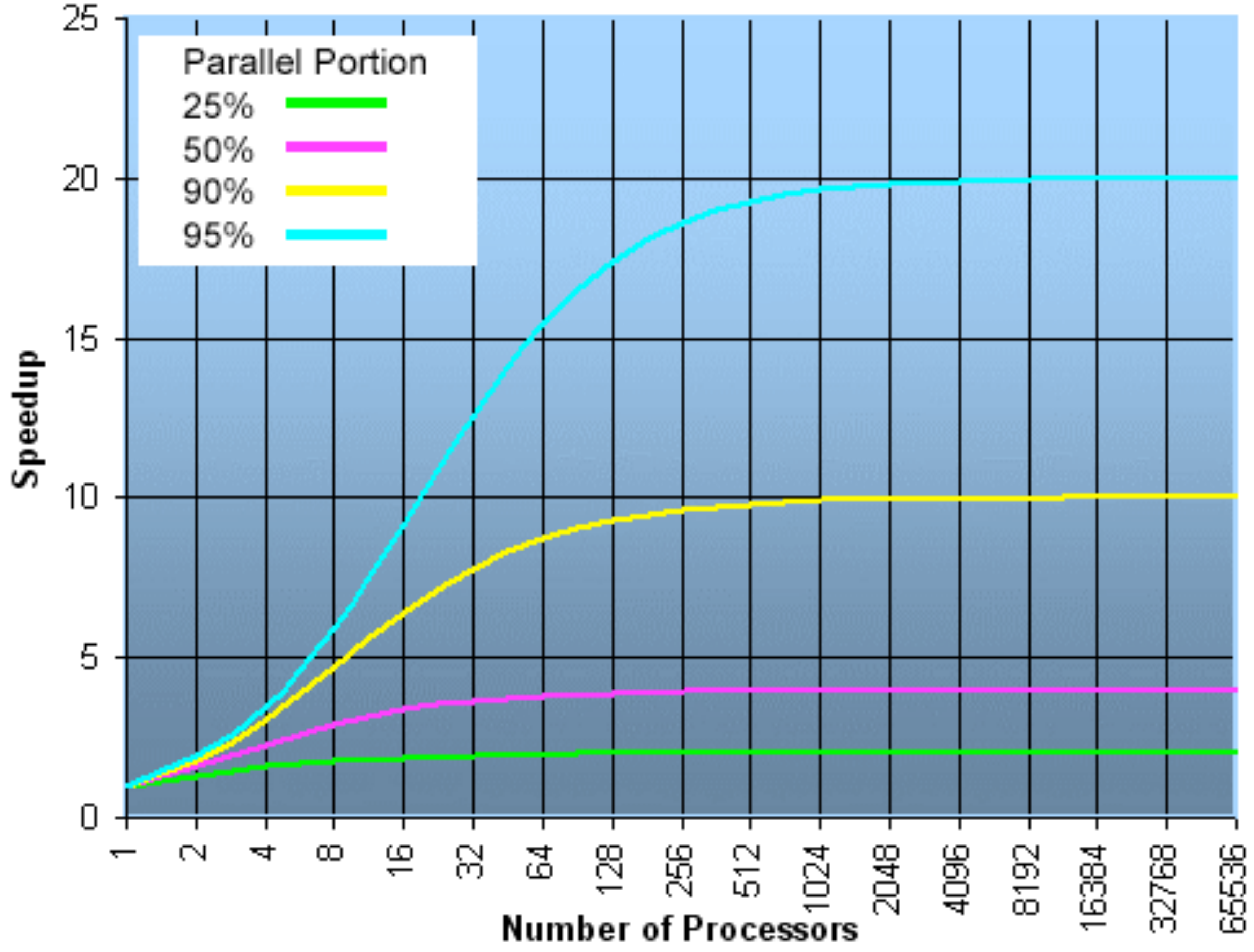
So simple it fits on one slide

```
public interface DistributedExecutorService extends ExecutorService {

   <T, K> Future<T> submit(Callable<T> task, K... input);

   <T> List<Future<T>> submitEverywhere(Callable<T> task);

   <T, K > List<Future<T>> submitEverywhere(Callable<T> task, K... input);
}


public interface DistributedCallable<K, V, T> extends Callable<T> {

  void setEnvironment(Cache<K, V> cache, Set<K> inputKeys);

}
```

However,behind the scenes..



Do not forget Gene Amdahl



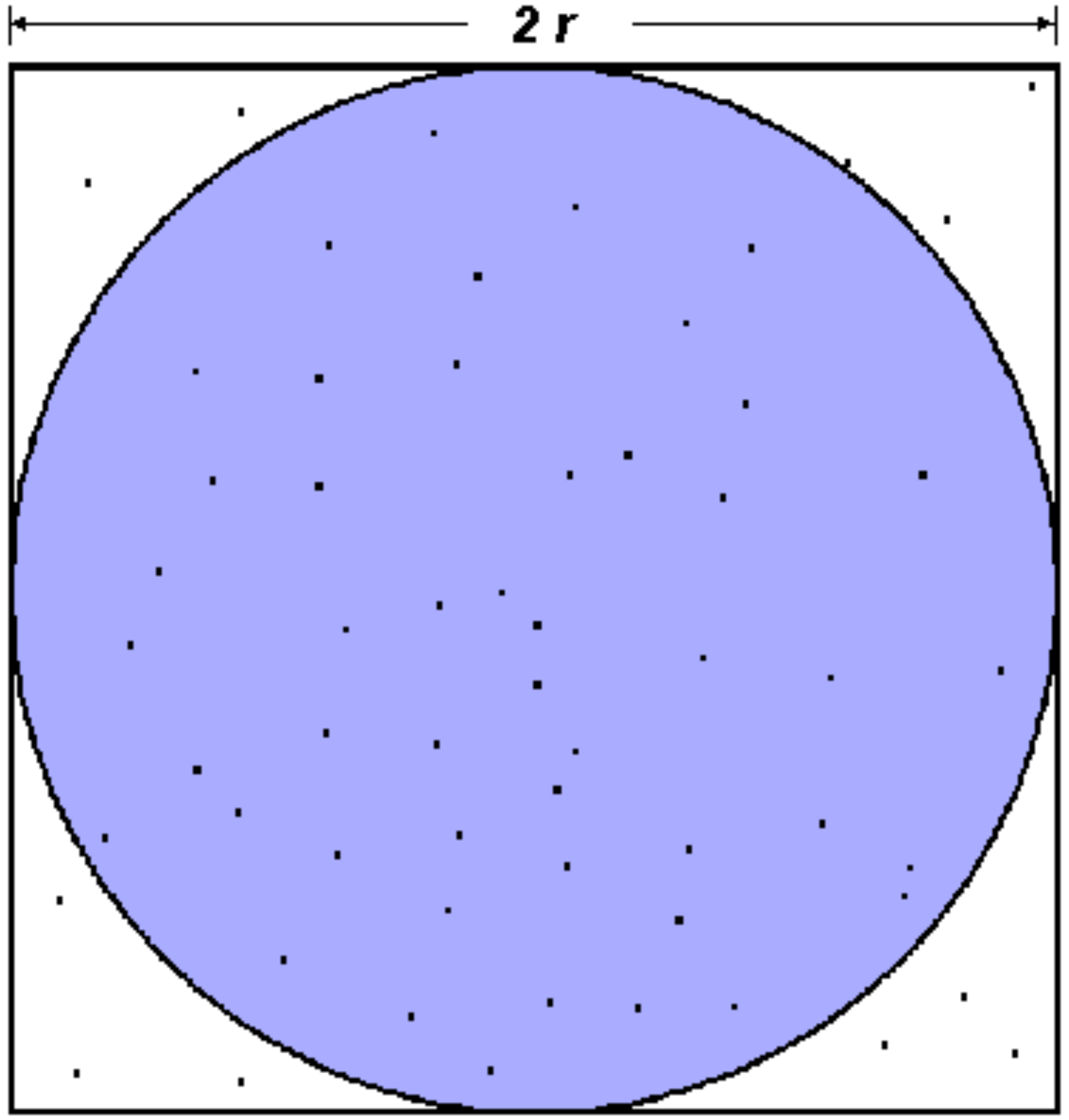
Source: https://computing.llnl.gov/tutorials/parallel_comp/

```
Speedup = 1/(p/n)+(1-p)
```

However, problems that increase the percentage of parallel time with their size are more scalable than problems with fixed percentage of parallel time

p = parallel fraction
n = number of processors

π approximation



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

π approximation

```
public class CircleTest implements Callable<Integer>, Serializable {

    private final int loopCount;

    public CircleTest(int loopCount) {
        this.loopCount = loopCount;
    }

    @Override
    public Integer call() throws Exception {
        int insideCircleCount = 0;
        for (int i = 0; i < loopCount; i++) {
            double x = Math.random();
            double y = Math.random();
            if (insideCircle(x, y))
                insideCircleCount++;
        }
        return insideCircleCount;
    }

    private boolean insideCircle(double x, double y) {
        return (Math.pow(x - 0.5, 2) + Math.pow(y - 0.5, 2)) <= Math.pow(0.5, 2);
    }
}
```

Non-distributed π approximation

```java
int numPoints = 50000000;
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<Integer> future = executor.submit(new CircleTest(numPoints));
int insideCircleCount = future.get();
double appxPi = 4.0 * insideCircleCount / numPoints;
System.out.println("PI appx is " + appxPi);
```

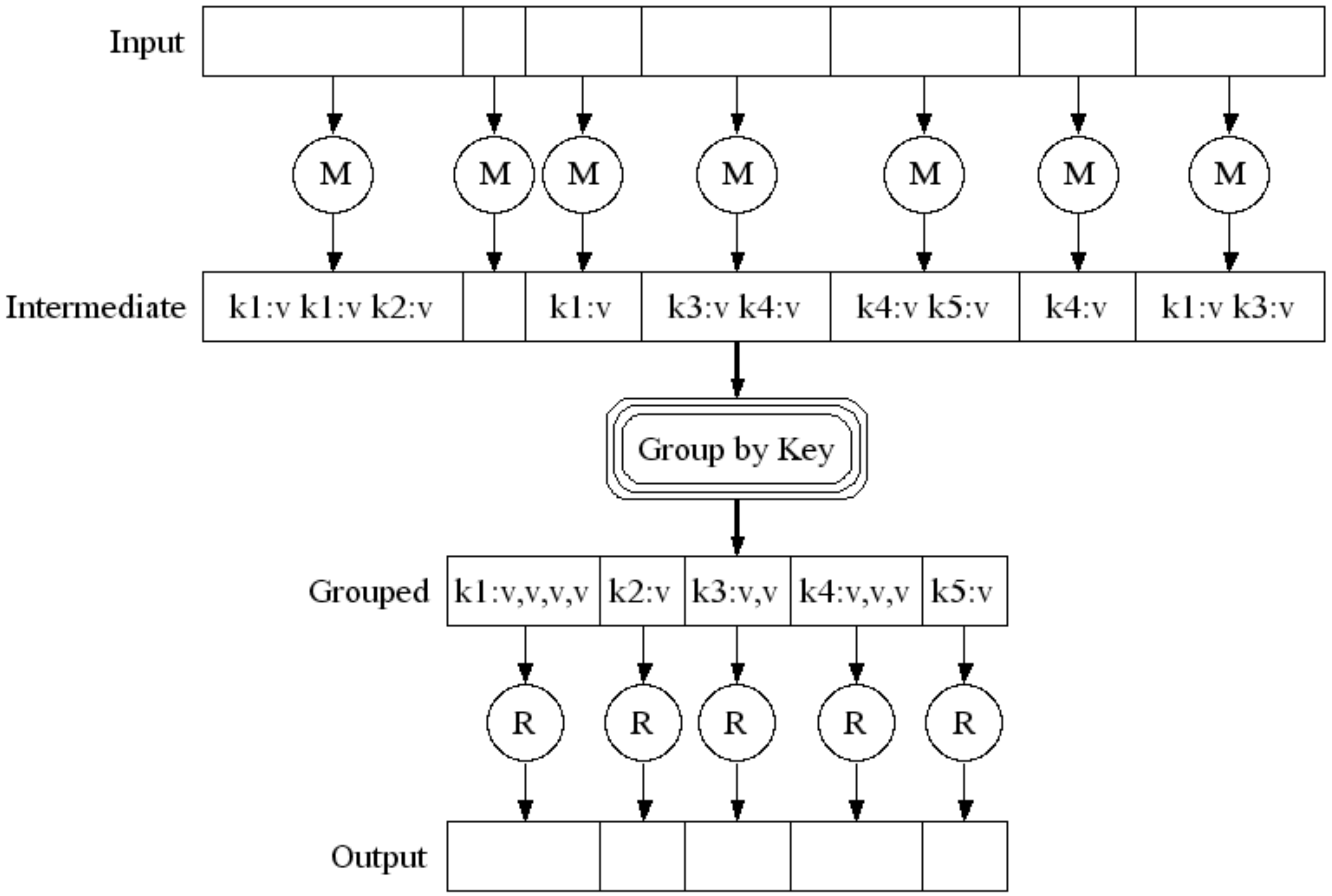
Distributed, parallel π approximation

```
EmbeddedCacheManager cacheManager = ...;
Cache<Object, Object> cache = cacheManager.getCache();
Transport transport = cache.getAdvancedCache().getRpcManager().getTransport();

int numPoints = 50000000;
int numServers = transport.getMembers().size();
int pointsPerWorker = numPoints / numServers;


DistributedExecutorService des = new DefaultExecutorService(cache);
List<Future<Integer>> results = des.submitEverywhere(new CircleTest(pointsPerWorker));
int insideCircleCount = 0;
for (Future<Integer> f : results) {
 insideCircleCount += f.get();
}
double appxPi = 4.0 * insideCircleCount / numPoints;
System.out.println("PI appx is " + appxPi);
```

Infinispan MapReduce

- We already have a data grid!
- Leverages Infinispan's DIST mode
- Cache data is input for MapReduce tasks
- Task components: Mapper, Reducer, Collator
- MapReduceTask cohering them together

MapReduce model



Source: http://labs.google.com/papers/mapreduce.html

Mapper, Reducer, Collator

```
public interface Mapper<KIn, VIn, KOut, VOut> extends Serializable {

   void map(KIn key, VIn value, Collector<KOut, VOut> collector);
}



public interface Reducer<KOut, VOut> extends Serializable {

    VOut reduce(KOut reducedKey, Iterator<VOut> iter);
}


public interface Collator<KOut, VOut, R> {

  R collate(Map<KOut, VOut> reducedResults);
}
```

Mapper, Reducer word count

```
public class WordCountMapper implements Mapper<String,String,String,Integer> {
  @Override
  public void map(String key, String value, Collector<String, Integer> c) {
     StringTokenizer tokens = new StringTokenizer(value);
     while (tokens.hasMoreElements()) {
        String s = (String) tokens.nextElement();
        c.emit(s, 1);
     }
  }
}

public class WordCountReducer implements Reducer<String, Integer> {
  @Override
  public Integer reduce(String key, Iterator<Integer> iter) {
     int sum = 0;
     while (iter.hasNext()) {
        Integer i = (Integer) iter.next();
        sum += i;
     }
     return sum;
  }
}
```

MapReduceTask word count

```
MapReduceTask<String, String, String, Integer> t =
new MapReduceTask<String, String, String, Integer>(cache);
t.mappedWith(new WordCountMapper()).reducedWith(new WordCountReducer());
String mostFrequentWord = t.execute(new Collator<String,Integer,String>(){

@Override
public String collate(Map<String, Integer> reducedResults) {
  String mostFrequent = "";
  int maxCount = 0;
   for (Entry<String, Integer> e : reducedResults.entrySet()) {
      Integer count = e.getValue();
      if(count > maxCount){
         maxCount = count;
         mostFrequent = e.getKey();
      }
   }
   return mostFrequent;
}
});
System.out.println("The most frequent word is " + mostFrequentWord);
```

Roadmap

- Improve task execution container
- Failover, execution policies
- Make sure it scales to terabytes and petabytes
- Integration with Hibernate OGM
- Do we need data analysis language?

Parting thoughts

- Data revolution is here, today!
- Profound socio-economic impact
- Do not sleep through it
- Infinispan as a platform for Big Data
- Join us in these exciting endeavours

Questions?

http://www.infinispan.org
http://blog.infinispan.org
http://www.twitter.com/infinispan
#infinispan