# 524 - Lecture 4

Map / Reduce

# MPI retro

- Ran out of memory in Vagrant
- Debugging was hard
  - Seg-fault
  - should connect to wedged process after the fact by catching signal and spinning
- efficient algorithm for BFS?
- easier than GPUs
- MPI gather and scatter were hugely helpful
- Not everything has a C++ binding
- Broadcast can use a tree
- Buffer challenging was challenging
- Don’t send too much at once :)

# The problem being solved

- Big cluster
    - Distributed
    - hardware is not fault tolerant
        - nodes die
    - big data
        - more data than fits on a single node
- Want to be used by a wide variety of programmers
    - Think people who are just graduating from Excel macros

# Map/Reduce

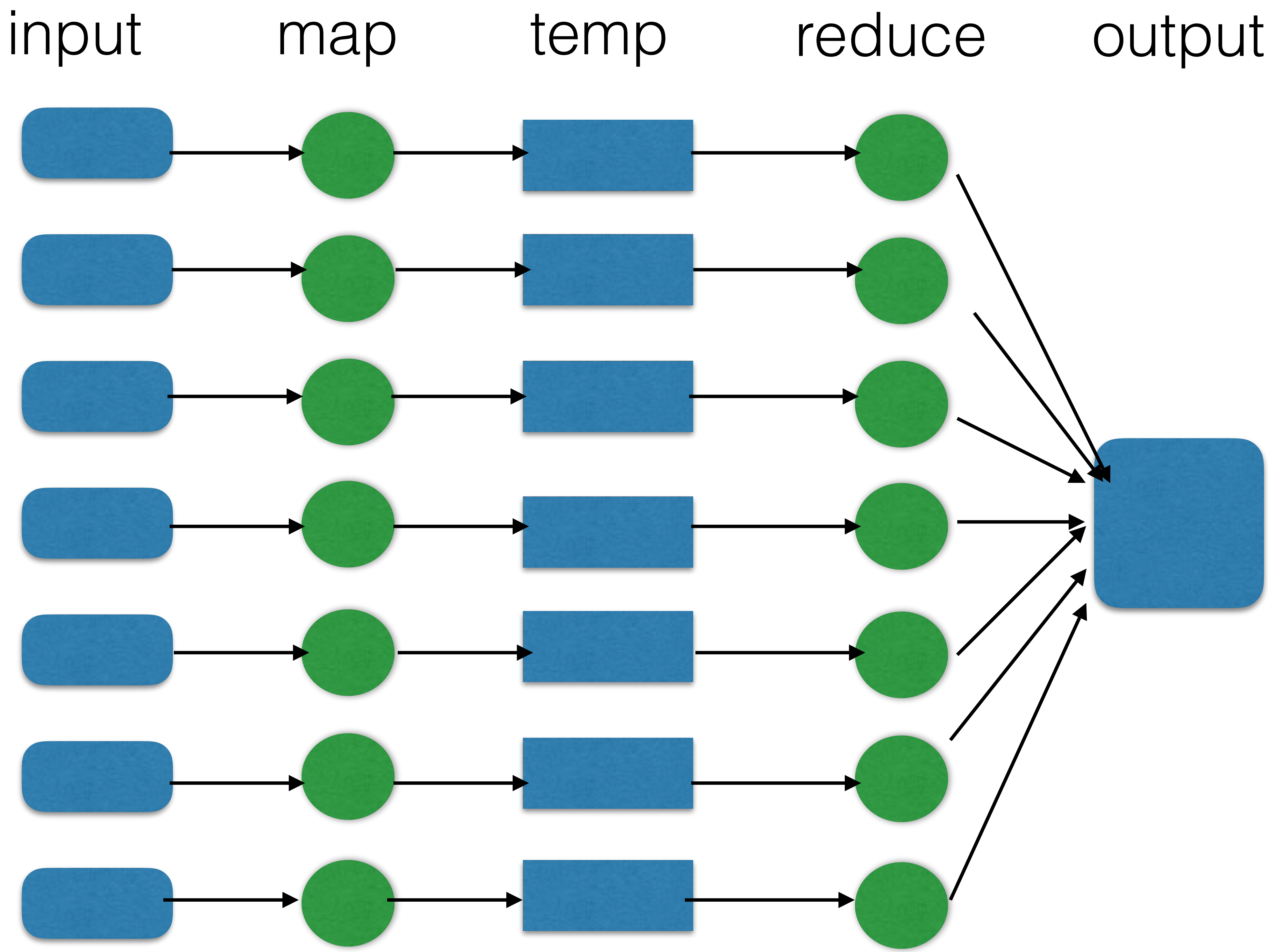
- Simple concept
  - map: apply a function to each element of data
  - reduce: summarize the result of a map operation
- With a twist:
  - map should be side-effect free (purely functional)
  - good reduce operators should be associative so that a reduction tree can be formulated

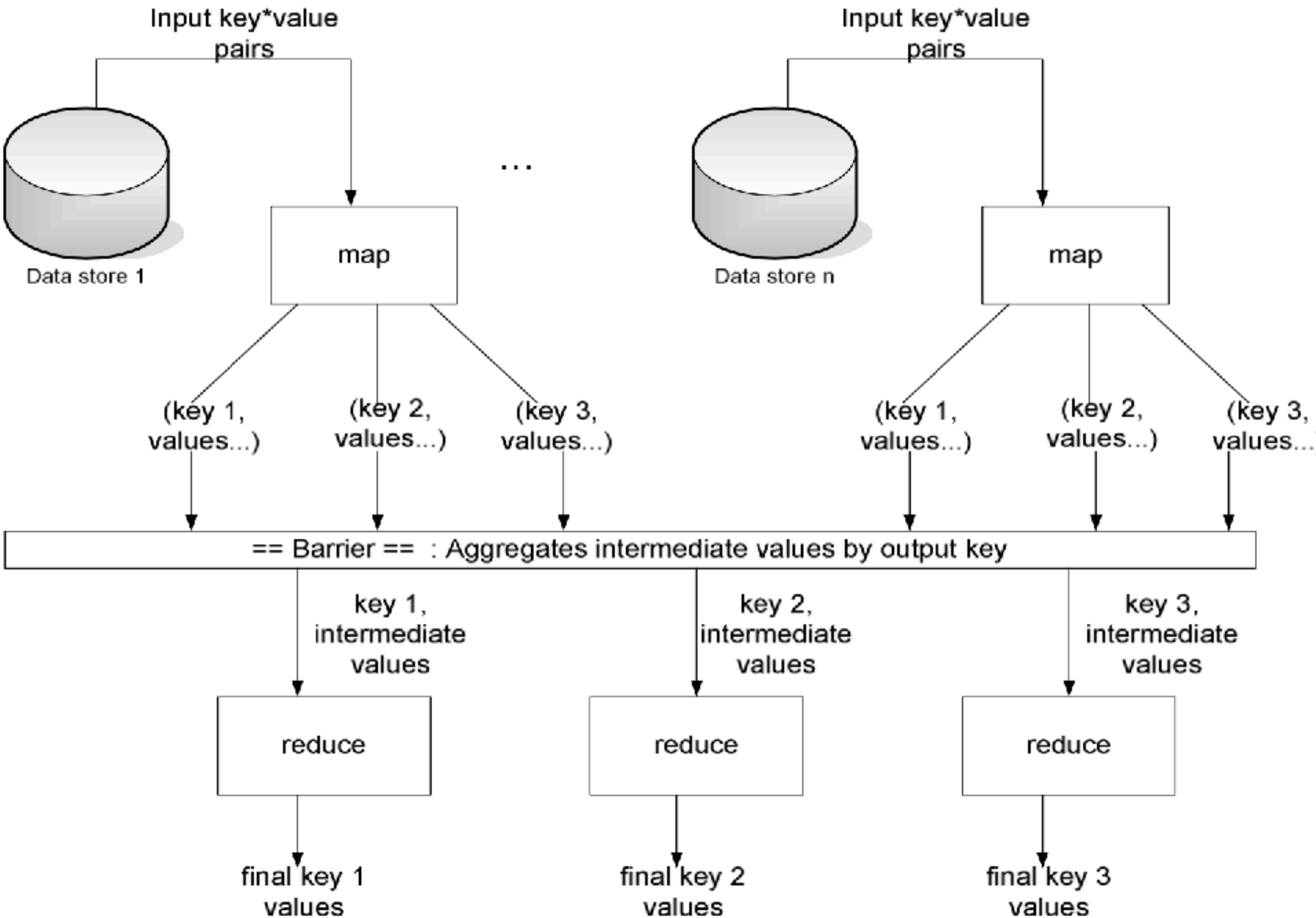
# In more CS speak

- Map/reduce is functional programming meets distributed systems
- Functional programming brings the side-effect freeness
- The framework brings attributes of distributed systems programming that are desirable:
  - fault tolerance
    - map operation died halfway through?  No problem, just re-srtart the node (the computation is side effect free!)
  - Scalable
    - map operations being side-effect free are easy to parallelize
    - associative reduction operators can be distributed and made hierarchical
  - You get these benefits *for free* if you buy into the marp/reduce framework

# In even more CS geek speak

- map(in_key, in_value) -> (out_key, intermediate_value) list
- reduce(out_key, intermediate_value list) -> out_value list
- For example:
  - Records of database (lets say SS# and name) are fed into a map function as (key, value) pairs
  - map produces one or more intermediate_value(s) along with an output key. For example, { (first, “John”), (last, “Smith”) }
  - Conceptually all resulting values from the map operation are squashed into a single list
  - reduce then processes this list. For example, counting up name frequency. { (first:John, 1), (last:Smith, 1) }
  - Note how this reduction operator is associative.


input
map
temp
reduce
output



Input key*value pairs
Input key*value pairs
Data store 1
map
...
Data store n
map
(key 1, values...)
(key 2, values...)
(key 3, values...)
(key 1, values...)
(key 2, values...)
(key 3, values...)
== Barrier == : Aggregates intermediate values by output key
key 1, intermediate values
key 2, intermediate values
key 3, intermediate values
reduce
reduce
reduce
final key 1 values
final key 2 values
final key 3 values

# What’s the catch?

- **Map/Reduce is a round hole and some times your problem is a square peg.**
  - You’ll see this when you code BFS…
- **In big distributed systems data distribution *is* work distribution.**
  - If your data isn’t distributed well or is distributed incorrectly for the problem you need to solve, performance suffers.
- These systems are trying to solve a lot at once: distributed systems, fault tolerance, distributed data storage, “ease of use”.
  - You have to buy into framework you choose.  It’s rarely something you can do on the side.
  - Choose wisely.

# Which framework to use?

- Worthy read for the newbie: http://www.metistream.com/comparing-hadoop-mapreduce-spark-flink-storm/
- Hadoop: your basic map/reduce framework. Most useful for HDFS (distributed file store) and YARN (the job dispatcher)
- Spark: a module within the Hadoop ecosystem. Provides in-memory computation capabilities (speed)
    - Also provides a lot of pre-built useful modules for ML, Graphs, SQL, etc
- Flink & Storm: provides streaming continuous processing, where as Hadoop/Spark are batch orientated.
- For this class, we'll use *either* Spark or "disco".
    - http://discoproject.org
    - http://spark.apache.org
    - Both work in vagrant. disco is easier to install. Spark requires you to upgrade java to Oracle's java. And download Spark 2 from the website directly.

# Example (disco)

```
from disco.core import Job, result_iterator

def map(line, params):
    for word in line.split():
        yield word, 1

def reduce(iter, params):
    from disco.util import kvgroup
    for word, counts in kvgroup(sorted(iter)):
        yield word, sum(counts)

if __name__ == '__main__':
    job = Job().run(input=["http://discoproject.org/
media/text/chekhov.txt"],
                    map=map,
                    reduce=reduce)
    for word, count in
result_iterator(job.wait(show=True)):
        print(word, count)
```

# Example (Spark)

```
from __future__ import print_function

import sys
from random import random
from operator import add

from pyspark.sql import SparkSession


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    spark = SparkSession\
        .builder\
        .appName("PythonPi")\
        .getOrCreate()

    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 < 1 else 0

    count = spark.sparkContext.parallelize(range(1, n + 1), partitions).map(f).reduce(add)
    print("Pi is roughly %f" % (4.0 * count / n))

    spark.stop()
```

# Example (in C)

```
package org.apache.spark.examples;

import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.Function;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.sql.SparkSession;

import java.util.ArrayList;
import java.util.List;

public final class JavaSparkPi {

  public static void main(String[] args) throws Exception {
    SparkSession spark = SparkSession
      .builder()
      .appName("JavaSparkPi")
      .getOrCreate();

    JavaSparkContext jsc = new JavaSparkContext(spark.sparkContext());

    int slices = (args.length == 1) ? Integer.parseInt(args[0]) : 2;
    int n = 100000 * slices;
    List<Integer> l = new ArrayList<>(n);
    for (int i = 0; i < n; i++) {
      l.add(i);
    }

    JavaRDD<Integer> dataSet = jsc.parallelize(l, slices);

    int count = dataSet.map(new Function<Integer, Integer>() {
      @Override
      public Integer call(Integer integer) {
        double x = Math.random() * 2 - 1;
        double y = Math.random() * 2 - 1;
        return (x * x + y * y < 1) ? 1 : 0;
      }
    }).reduce(new Function2<Integer, Integer, Integer>() {
      @Override
      public Integer call(Integer integer, Integer integer2) {
        return integer + integer2;
      }
    });

    System.out.println("Pi is roughly " + 4.0 * count / n);

    spark.stop();
  }
}
```

# Installation

## Spark

```
sudo apt-get install python-software-properties
sudo apt-add-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java8-installer
sudo apt-get install oracle-java8-set-default
wget http://d3kbcqa49mib13.cloudfront.net/spark-2.0.2-bin-hadoop2.7.tgz
tar xfz spark-2.0.2-bin-hadoop2.7.tgz
```

```
cd spark-2.0.2-bin-hadoop2.7/
./bin/run-example SparkPi 10
```

## Disco

```
apt-get install erlang
apt-get install git
git clone git://github.com/discoproject/disco.git disco
cd disco
make
export DISCO_HOME=/home/vagrant/disco
disco/bin/disco start
```

n $\leftarrow$ size of rows

$D^{(0)} \leftarrow$ input distance matrix

$\Pi^{(0)} \leftarrow$ calculate precedence matrix

tN $\leftarrow$ number of threads

for k $\leftarrow$ 1 to n

  parallel start

    tid $\leftarrow$ id of thread

    for i $\leftarrow \dfrac{\text{tid}^* n}{\text{tN}}$ to $\dfrac{(\text{tid} + 1)^* n}{\text{tN}} - 1$

      for j $\leftarrow$ 1 to n

$$d_{ij}^{(k)} \leftarrow min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

  parallel end

return $D^{(n)}$ $\Pi^{(n)}$

**Algorithm 2:** OpenMP pseudocode for the all-pairs-shortest-path problem.

input: $\left[\mathrm{i}\ \ j\ \ d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\ \right]$

$\mathrm{Map}\left(\mathrm{Object\ key} = (\mathrm{i\ \ j}),\ \mathrm{Value\ val} = \left(d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\right)\right)$

  if i == k or j == k then

    for m ← 1 to n

      if j == k then write(j m), $\left(\mathrm{i}\ \ j\ \ d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\right)$

      if i == k then write(m i), $\left(i\ \ j\ \ d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\right)$

    else then write(i j), $\left(i\ \ j\ \ d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\right)$

$\mathrm{Reduce}\left(\mathrm{Object\ key} = (\mathrm{i\ \ j}),\ \mathrm{Value\ val} = \left(i\ \ j\ \ d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\right)\right)$

  $d_{ij}^{(k)} \leftarrow min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

  $$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

  write(i j), $\left(d_{ij}^{(k)}\ \ \pi_{ij}^{(k)}\right)$

Driver( )

  n ← size of rows

  for $k \leftarrow 1$ to $n$

    $\mathrm{Map}\left((i\ \ j), \left(d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\right)\right)$

    $\mathrm{Reduce}\left((i\ \ j), \left(i\ \ j\ \ d_{ij}^{(k-1)}\ \ \pi_{ij}^{(k-1)}\right)\right)$

**Algorithm 4:** MapReduce pseudocode for the all-pairs-shortest-path problem.

**Table 1:** Execution times for the all-pairs-shortest-path problem.

| Node size | Framework | | | |
|---|---|---|---|---|
| | MapReduce | MPI | | OpenMP |
| | | Cluster | Single machine | |
| 10 | 2 m 26 s | 0.32 s | 0.34 s | 0.1 s |
| 100 | 16 m 52 s | 0.44 s | 0.41 s | 0.25 s |
| 1000 | 4 h 4 m 39 s | 4 m 48 s | 24.14 s | 8.03 s |

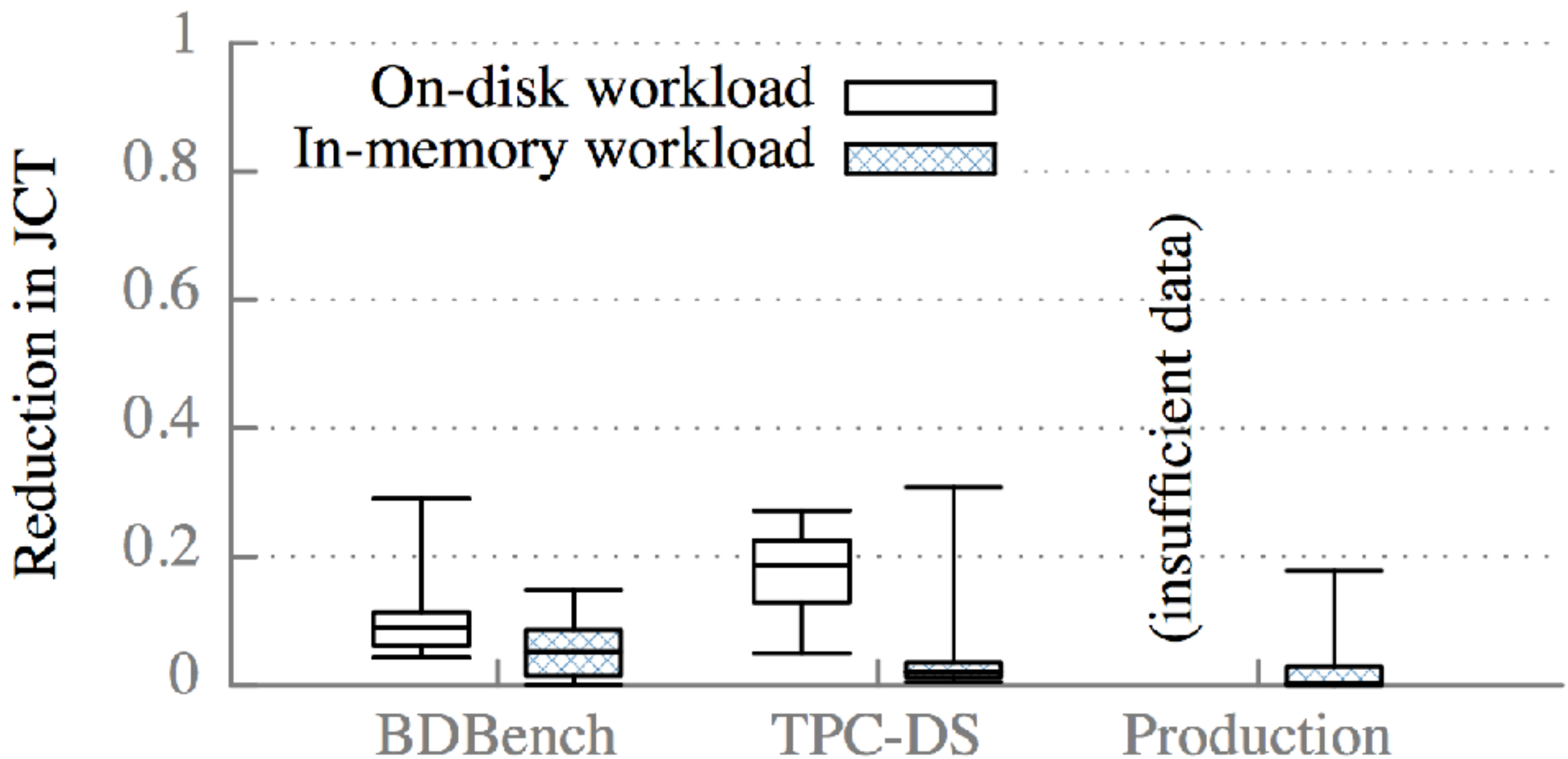
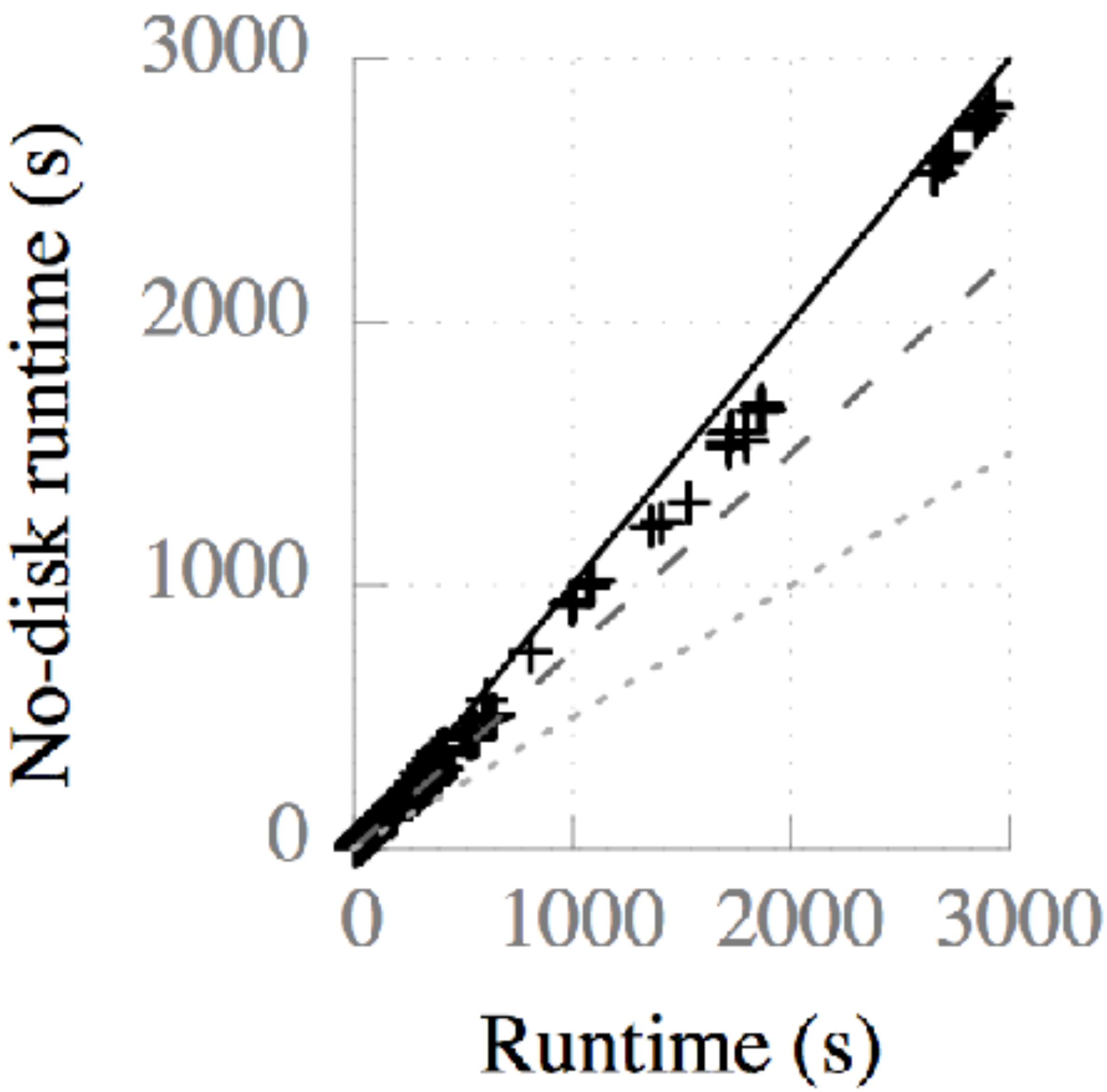
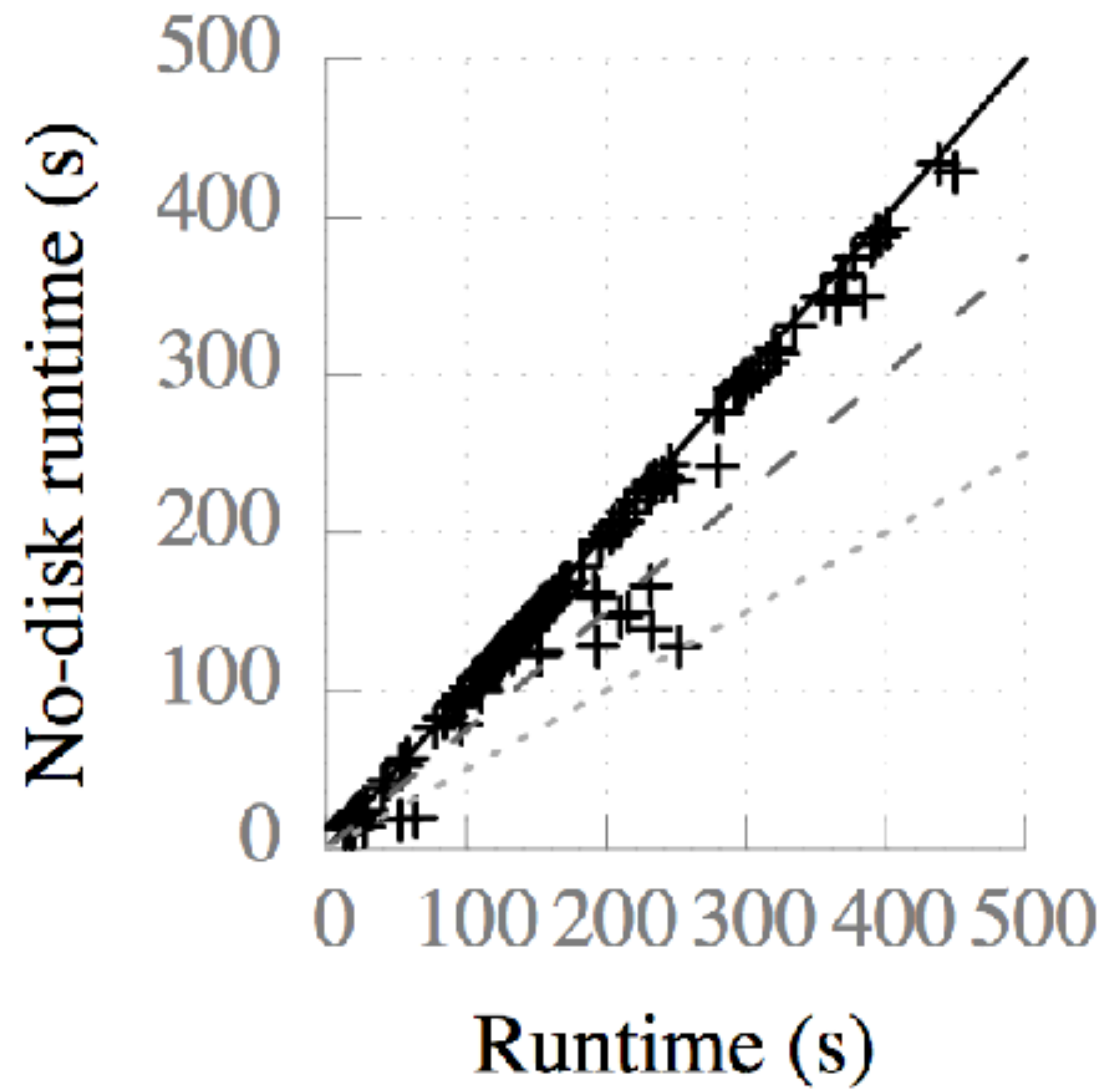



Figure 3: Improvement in job completion time (JCT) as a result of eliminating all time spent blocked on disk I/O. Boxes depict 25th, 50th, and 75th percentiles; whiskers depict 5th and 95th percentiles.




(c) TPC-DS, disk




(d) TPC-DS, in-memory

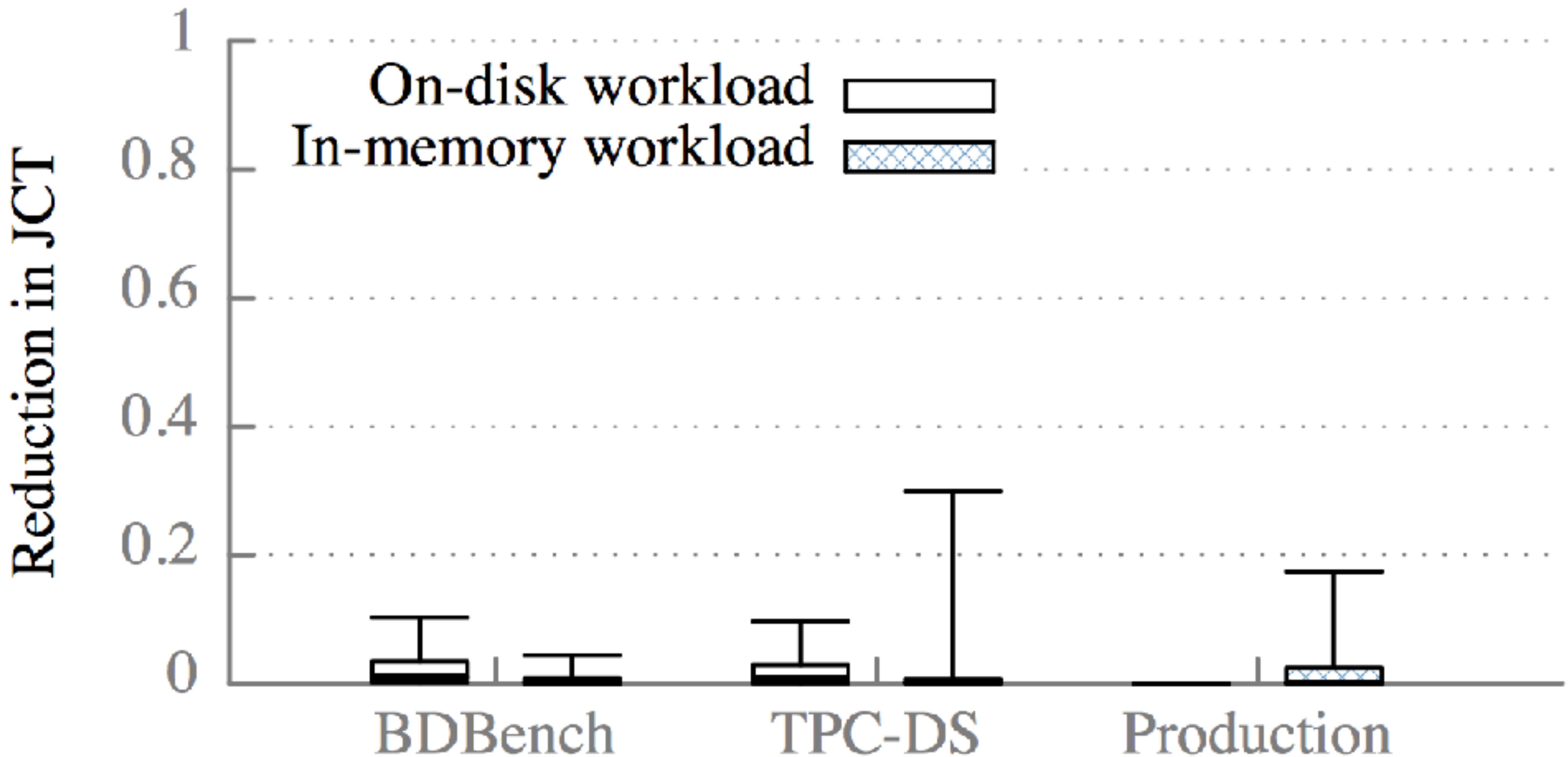



Figure 8: Improvement in job completion time as a result of eliminating all time blocked on network.

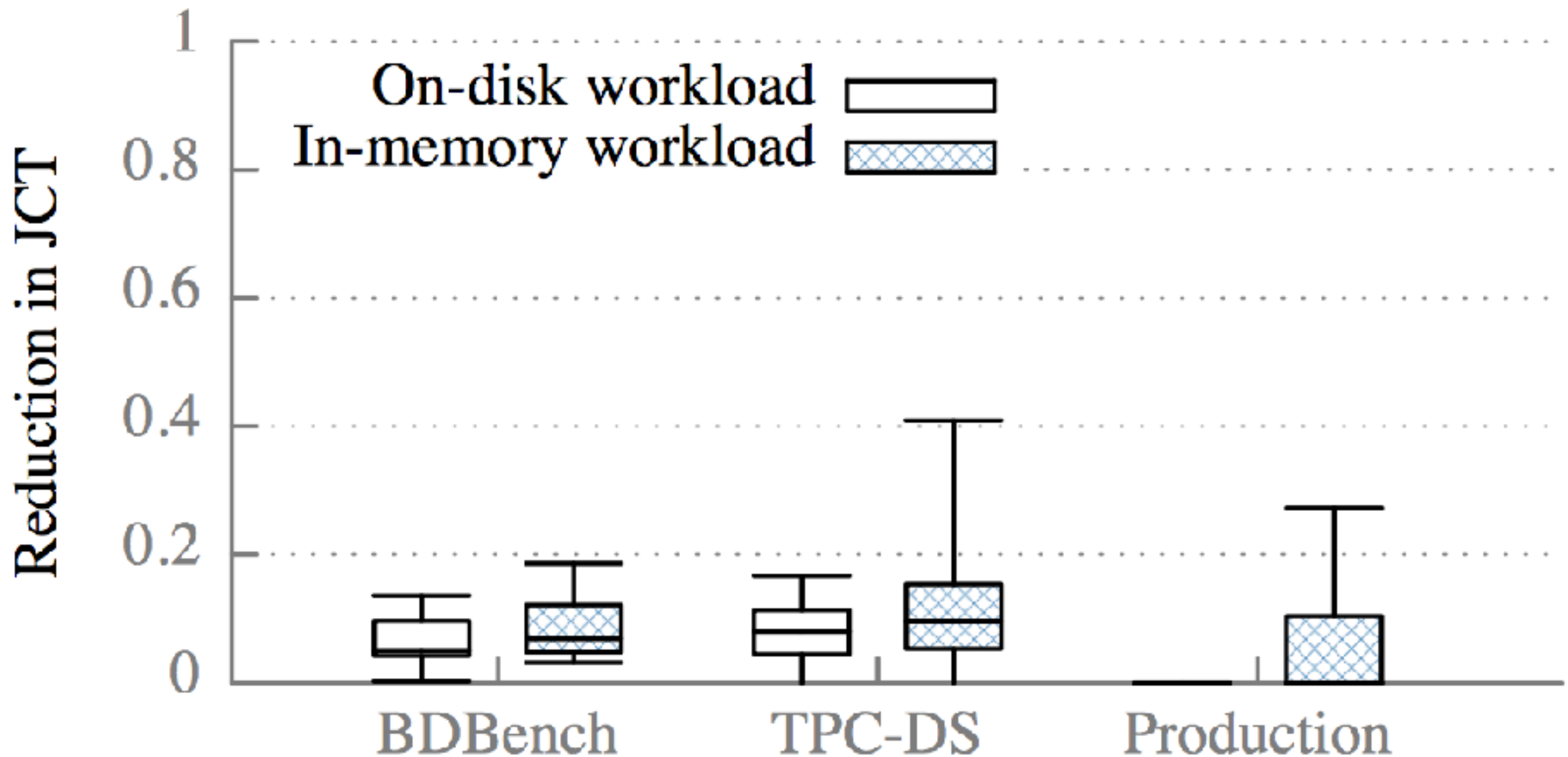
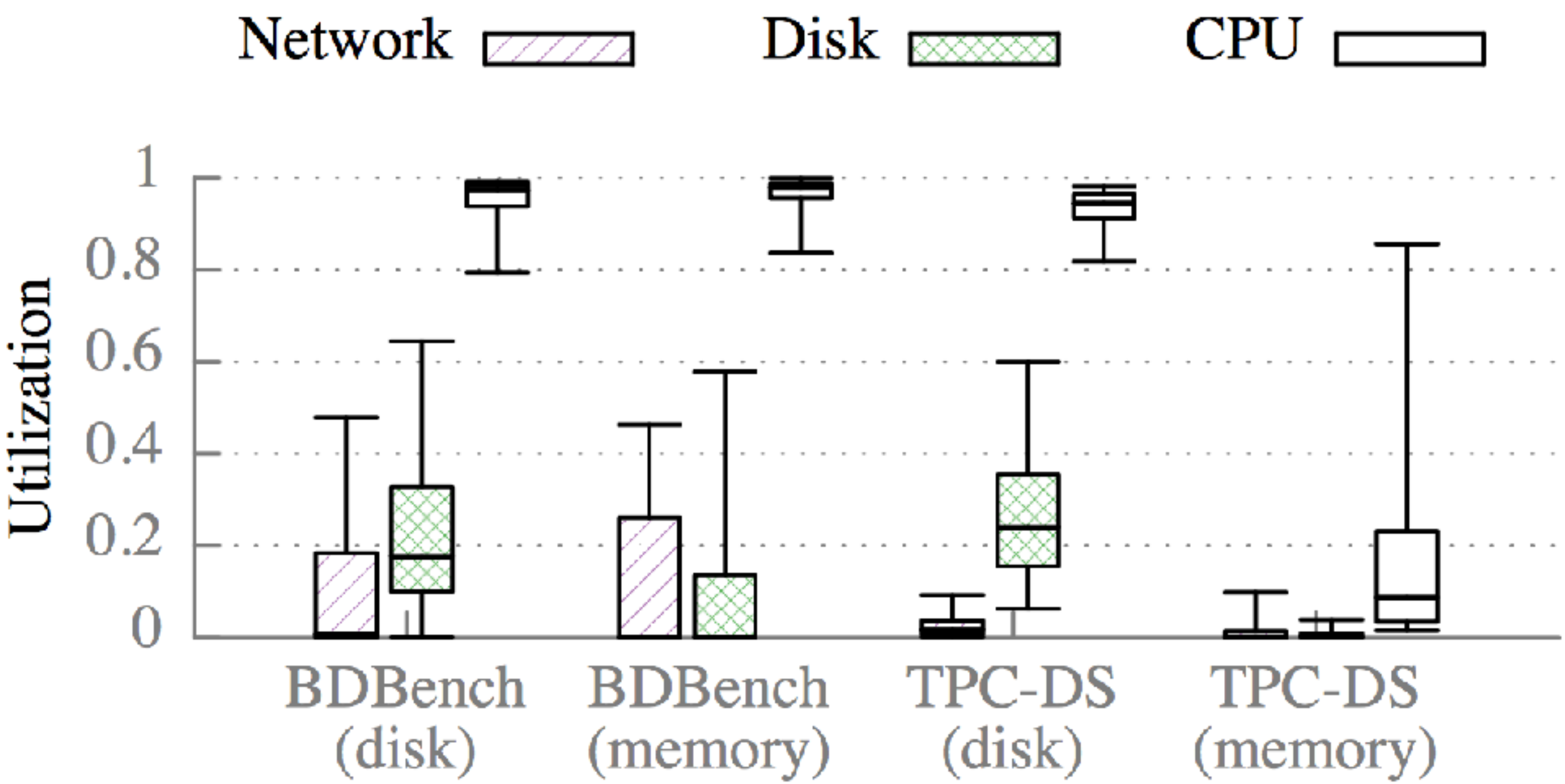



Figure 12: Potential improvement in job completion time as a result of eliminating all stragglers. Results are shown for the on-disk and in-memory versions of each benchmark workload.



Network
Disk
CPU
Utilization
1
0.8
0.6
0.4
0.2
0
BDBench (disk)
BDBench (memory)
TPC-DS (disk)
TPC-DS (memory)

# Beyond Map/Reduce

- GraphLab - vertex centric computation
- https://turi.com
- repeat {
  gather_from_edges
  process_at_vertex
  scatter_to_edges
  }

# Beyond Map/Reduce

- TensorFlow - construct dataflow graph.
  - https://www.tensorflow.org
  - vertices = operations.
  - Edges = flow of tensors (multidimensional arrays)

# Beyond M/R

- Microsoft CNTK
  - https://github.com/Microsoft/CNTK
  - Don't know much about it but looks awesome. Works on Windows and Linux. Designed for NN training on clusters of systems with CPU/GPUs.

# Beyond Map/Reduce

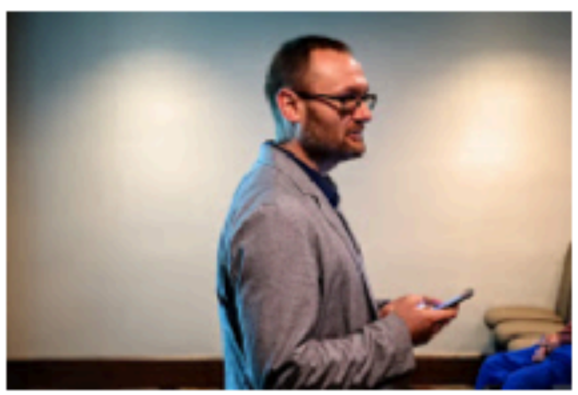
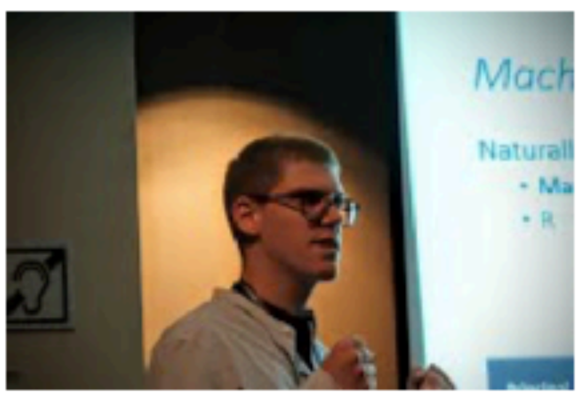
- NoSQL - e.g. MongoDB
  - data stored as key:value or [key:value]
- Useful for data that isn't well structured or where the structure isn't known ahead of time ("agile development")
- In my experience, very useful for simple things. Very hard to use for complex ones. Updates across key:value pairs…

Home
Call for Papers
Important Dates
Invited Speakers
**Program**
Program Committee
Submission
Workshop Organizers
Sitemap

# Program

**Session 1 8h30 - 10h00**

- 8h30 - 8h40 Welcome and Introduction of keynote
- 8h40 - 9h40 Keynote: *Ion Stoica*, "Spark: Past, Present, and Future"
- 9h40 - 10h00 "Bridging the gap: Towards optimization across linear and relational algebra", *Andreas Kunft, Alexander Alexandrov, Asterios Katsifodimos and Volker Markl*

| | | |
|---|---|---|
|  |  |  |
| **Introduction**: Jacek Sroka | Ion Stoica | Andreas Kunft |

**nftSession 2 10h30 - 12h00**

- 10h30 - 10h50 "Faucet: a user-level, modular technique for flow control in dataflow engines", *Andrea Lattuada, Frank McSherry and Zaheer Chothia*
- 10h50 - 11h10 "Model-Centric Computation Abstractions in Machine Learning Applications", *Bingjing Zhang, Peng Bo and Judy Qiu*
- 11h10 - 11h35 "DFA Minimization in Map-Reduce", *Gösta Grahne, Shahab Harrafi, Iraj Hedayati and Ali Moallemi*

# Parting thoughts on M/R

- Should you use a map/reduce framework? Yes, if:
    - you're processing peta-byte scales of data
    - you algorithm fits well within the paradigm
    - your data is already in HDFS and not an a RDMS
  - Informally, I've been told a lot of organizations use Hadoop for two things: HDFS and YARN.  (file storage and job-dispatch).  The Map/Reduce aspect comes in handy on occasion but isn't the core of what they do.
    - You tell me, is this true?
- What do I use?  I don't.  For large scale graph analytics I use a PGAS framework we wrote (Grappa) and for general analytics (of which I do a lot in the finance world) I use SQL and C.
  - My data is in the GB range not TB or PB range, however.