Center for Mathematical Modeling
University of Chile

CMM
Center for Mathematical Modeling

HPC 123

Scientific Computing on HPC systems

Module 2/2

By
Juan Carlos Maureira B.
<*jcm@dim.uchile.cl*>

v1.0 - 10/08/2018

Overview

- From multiprocessing to spark. A practical example.
- MapReduce 1: grouping by key a large key-value file with Spark
- MapReduce 2: extracting sources in an astronomical image with spark.
- Wrapping up: The take aways.

A practical guide to go

From multiprocessing to spark

The problem: PowerCouples*

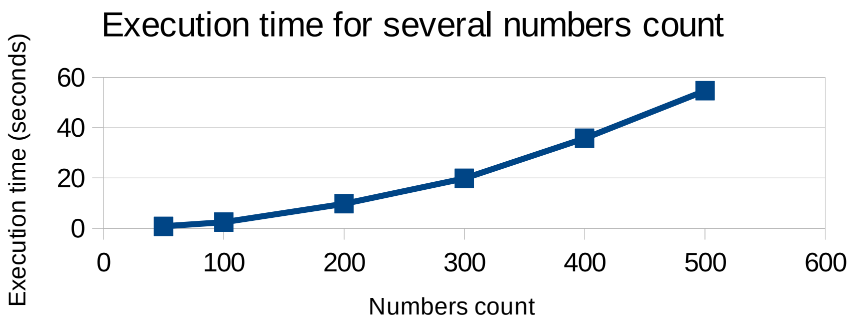
- Find the max(x^y) from a sequence of integer numbers.
- Input:
  - Label: $x_1,x_2,x_3,x_4,....,x_n$
- Output:
  - Label: x_i,x_j

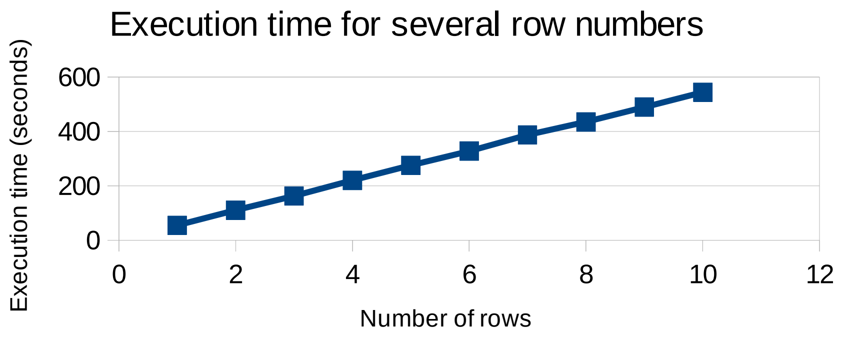
We know the best solution is to order the sequence of numbers and take the last two of them. However, for illustrative purposes, we want to really compute each x^y for all combinations in order to have an inefficient solution to optimize with parallel/distributed processing.

* Thanks to Nicolas Loira for this example

Base code for solving PowerCouples

```
import os, sys, argparse, csv, itertools

def pow(x):
    return x[0]**x[1]

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pow)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
            description='PowerCouples Serial native version')
    parser.add_argument('-i','--input',
            dest="input_csv", help="input file in csv format",
            required=True, type=argparse.FileType('r'))
    parser.add_argument('-o','--output',
            dest="output_csv", help="output file in csv format",
            default=sys.stdout, type=argparse.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)
    for row in csv.reader(args.input_csv):
        name = row[0]
        numbers = [int(i) for i in row[1:] ]
        pc = find_powerCouple(numbers)
        out.writerow( (name, pc[0], pc[1]) )
```





Multiprocessing version for PowerCouples

```
import os, sys, argparse as ap, csv, itertools
import pew.pew as pw
import multiprocessing as mp

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return pw.pew(x[0],x[1])

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pewT)

def worker(infile,out_q):
    try:
        results = []
        print "processing %s",infile
        for row in csv.reader(infile):
            name = row[0]
            numbers = [int(i) for i in row[1:] ]
            pc = find_powerCouple(numbers)
            results.append( (name, pc[0], pc[1]) )
        out_q.put(results)
    except:
        print "worker failed"
    finally:
        print "done"
```

```
if __name__ == "__main__":
    parser = ap.ArgumentParser()
    parser.add_argument('-i','--inputs',nargs='+',
            dest="inputs_csv", help="list of input files",
            required=True, type=ap.FileType('r'))
    parser.add_argument('-o','--output', dest="output_csv",
            help="output file in csv format", default=sys.stdout,
            type=ap.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)
    m = mp.Manager()
    result_queue = m.Queue()

    jobs = []
    for infile in args.inputs_csv:
        jobs.append( mp.Process(target=worker,
                    args=(infile,result_queue)))
        jobs[-1].start()

    for p in jobs:
        p.join()

    num_res=result_queue.qsize()
    while num_res>0:
        out.writerows(result_queue.get())
        num_res -= 1
```

Thread version for PowerCouples

```
import os
import sys
import argparse
import csv
import itertools
import pew.pew as pw
import threading

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return pw.pew(x[0],x[1])

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pewT)

def worker(infile,out,lock):
    for row in csv.reader(infile):
        name = row[0]
        numbers = [int(i) for i in row[1:] ]
        pc = find_powerCouple(numbers)
        with lock:
            out.writerow( (name, pc[0], pc[1]) )

    return True
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i','--inputs',nargs='+',
        dest="inputs_csv", help="list of input file in csv format",
        required=True, type=argparse.FileType('r'))

    parser.add_argument('-o','--output', dest="output_csv",
        help="output file in csv format", default=sys.stdout,
        type=argparse.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)

    out_lck = threading.Lock()
    threads = []
    for infile in args.inputs_csv:
        t = threading.Thread(target=worker,
                        args=(infile,out,out_lck))
        threads.append(t)
        t.start()

    print "waiting for termination"
    for t in threads:
        t.join()

    print "done"
```

Parallel Python version for PowerCouples

```
import os, sys, argparse, csv, itertools
import pew.pew as pw
import pp

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return pw.pew(x[0],x[1])

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pewT)

def worker(infile):
    results = []
    for row in csv.reader(infile):
        name = row[0]
        numbers = [int(i) for i in row[1:] ]
        pc = find_powerCouple(numbers)
        results.append( (name, pc[0], pc[1]) )

    return results
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i','--inputs',nargs='+',
            dest="inputs_csv",
        help="list of input file in csv format", required=True,
        type=argparse.FileType('r'))
    parser.add_argument('-o','--output', dest="output_csv",
        help="output file in csv format", default=sys.stdout,
        type=argparse.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)
    ncpus = 10
    jobs = []
    ppservers = ()
    job_server = pp.Server(ncpus, ppservers=ppservers)

    for infile in args.inputs_csv:
        f = list(infile,)
        jobs.append(job_server.submit(worker,(f,),
                (find_powerCouple,pewT,pow),
                ("csv","itertools","pew.pew as pw")))

    for job in jobs:
        out.writerows(job())

    job_server.print_stats()
```

Parallel Python version + Slurm

```
#!/bin/bash
#
# PowerCouples Parallel Python version
#
# starting script
# 2016 (c) Juan Carlos Maureira, CMM - Uchile

IN_FILES=($@)
NUM_FILES=${#IN_FILES[@]}
CORES=20
NUM_WORKERS=`echo "scale=1; \
          ($NUM_FILES / $CORES) + 0.5" | bc | cut -f 1 -d"."`
PORT=5000
SECRET="my_secret"

module load python/2.7.10

function deploy_workers() {
    let NODES=$1

    RESOURCES=""

    if [ $NODES -le 1 ]; then
        CORES=$NUM_FILES
        RESOURCES="-n1 -c $CORES"
    else
        RESOURCES="-N $NODES -c $CORES"
    fi
```

```
    echo "running for $1 workers"

    srun --exclusive –reservation=cursomop \
     $RESOURCES -J ppserver ~/.local/bin/ppserver.py \
    -w $CORES -a -p $PORT -s $SECRET

    echo "closing workers..."
}

if [ $NUM_WORKERS -eq 0 ]; then
    echo "No input files given"
    exit 1
fi

deploy_workers $NUM_WORKERS &

sleep 1
python ./powercouples-pp.py -i ${IN_FILES[@]}
sleep 1

scancel --name ppserver -s INT

wait

echo "done"
```

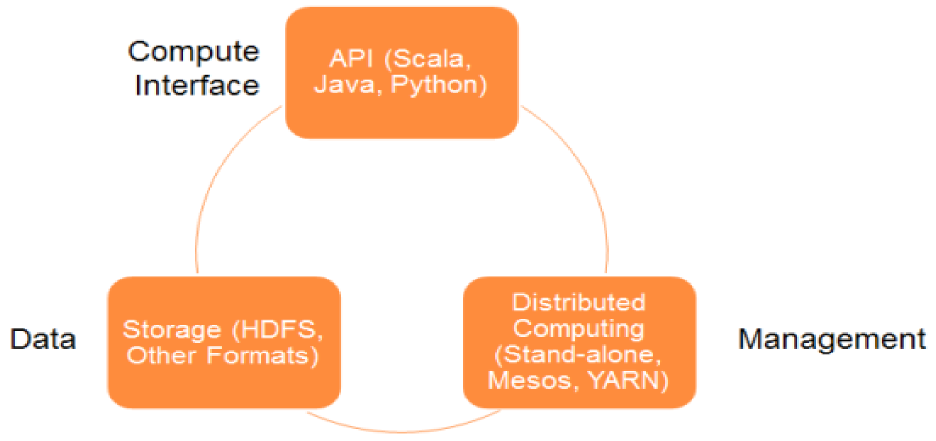
PowerCouples with Spark

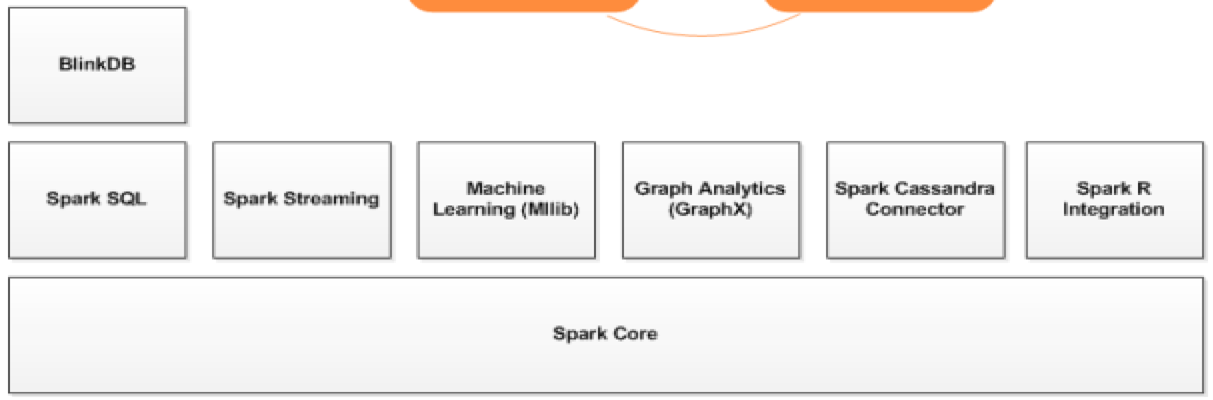
(Introducing distributed processing with Spark)

Data Orchestration with Spark

Spark's Basic Concepts

- Runs on top of Hadoop (yes, using java)
- Job's workflow are DAGs
- Scheduler: YARN (yet another resource negotiator)
  - A weakness of spark
- Not only MapReduce and HDFS
  - A Strength of spark
- Lazy Evaluation, Broadcast, shared vars
- Scala / Java / Python!!!!
- Many data connectors (not for binary data)
- Master-Slave deployment
  - Require a deployment of services in order to create a spark cluster.


Compute Interface
API (Scala, Java, Python)
Data
Storage (HDFS, Other Formats)
Distributed Computing (Stand-alone, Mesos, YARN)
Management



Spark version of PowerCouples

```
# PowerCouples
# Spark (v2.0) Version
# Juan Carlos Maureira

import os
import sys
import argparse
import csv
import itertools
import pew.pew as p

from pyspark import SparkContext,SparkConf

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return p.pew(x[0],x[1])

def find_powerCouple(raw_row):

    row = raw_row.split(",")
    name = str(row[0])
    numbers = [int(i) for i in row[1:] ]
    tuples = itertools.permutations(numbers,2)
    pc =  max(tuples, key=pewT)
    return [name, pc[0], pc[1]]
```

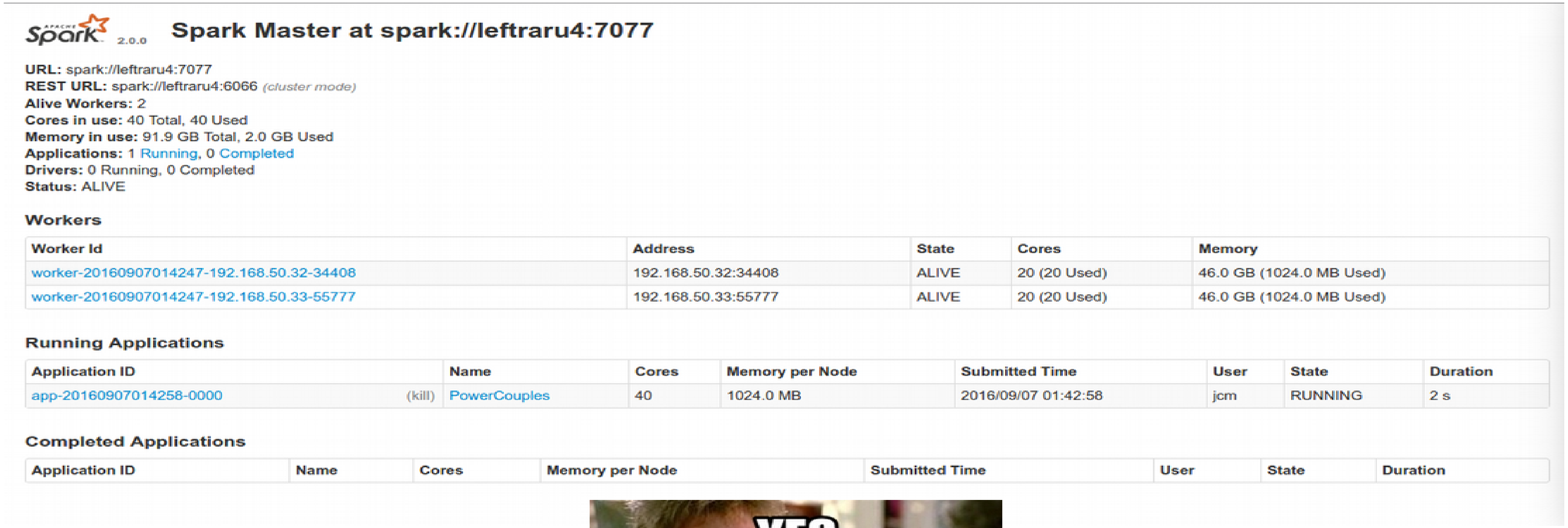
```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i','--input', dest="input_csv",
            help="input file in csv format", required=True)
    parser.add_argument('-o','--output', dest="output_csv",
            help="output file in csv format", default=sys.stdout,
            type=argparse.FileType('w'))

    args = parser.parse_args()

    # set the spark context
    conf = SparkConf()
    conf.setMaster("local[4]")
    conf.setAppName("PowerCouples")
    sc = SparkContext(conf=conf)

    # compute power couples
    infile = sc.textFile(args.input_csv,4)
    result = infile.map(find_powerCouple)
            .map(lambda elem: elem[0]+","
                   +str(elem[1])+","+str(elem[2])).collect()

    # write results
    out = csv.writer(args.output_csv)
    for row in result:
        out.writerow([row])
```

Spark + Slurm

```
#!/bin/bash
#
# Spark 2.0 Slurm submision script
# Deploy master and workers, then submit the python script
#
# 2016 (c) Juan Carlos Maureira
# Center for Mathematical Modeling
# University of Chile

# make the spark module available
module use -a $HOME/modulefiles
module load spark
Module load python/2.7.10

NUM_WORKERS=2
CORES_PER_WORKER=20

if [ "$1" == "deploy" ]; then
    # deploy spark workers on nodes

    MASTER=$2
    HOST=`hostname -a`
    echo "starting slave at $HOST"
    $SPARK_HOME/sbin/start-slave.sh \
            --cores $CORES_PER_WORKER \
            spark://$MASTER:7077

    tail -f /dev/null
```

```
else
    # main routine

    MASTER=`hostname -a`
    echo "Using as master $MASTER"
    $SPARK_HOME/sbin/start-master.sh

    srun --exclusive -n $NUM_WORKERS \
            --reservation=cursomop \
            -c $CORES_PER_WORKER \
            -J spark $0 deploy $MASTER &

    sleep 10

    spark-submit --master spark://$MASTER:7077 $@

    # clean up
    scancel --name spark
    $SPARK_HOME/sbin/stop-master.sh
    echo "done"
fi
```

Caso Figura: PowerCouples
Spark Web UI



Spark Master at spark://leftraru4:7077

URL: spark://leftraru4:7077
REST URL: spark://leftraru4:6066 *(cluster mode)*
Alive Workers: 2
Cores in use: 40 Total, 40 Used
Memory in use: 91.9 GB Total, 2.0 GB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20160907014247-192.168.50.32-34408	192.168.50.32:34408	ALIVE	20 (20 Used)	46.0 GB (1024.0 MB Used)
worker-20160907014247-192.168.50.33-55777	192.168.50.33:55777	ALIVE	20 (20 Used)	46.0 GB (1024.0 MB Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160907014258-0000 (kill)	PowerCouples	40	1024.0 MB	2016/09/07 01:42:58	jcm	RUNNING	2 s

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Going further with Spark

Reducing data with combine by key

The problem: Transactions grouping

- Group all items on the same transaction
- Variable transaction length
- Tuple (tr_id,item) are not ordered
- Input file:
  - Each line is a tuple
- Numbers of this example:
  - Number of tuples: 449.295
  - Number of items: 28.000
  - Number of transactions : 150.000

```
jcm@movil:~/codes/spark/combine$ head trx-sampled.csv
1,41794
1,21015
1,22927
1,25711
2,1906
3,33044
3,1670
3,40819
3,30374
3,18250
jcm@movil:~/codes/spark/combine$
```

Combine data with spark

```
#!/usr/bin/env python3
#
# Spark 2.3 combine by key example
#
# 2018 (c) Juan Carlos Maureira
# Center for Mathematical Modeling
# University of Chile

from __future__ import print_function
from pyspark import SparkContext,SparkConf

import os
import csv

#
# Main routine
#
if __name__ == "__main__":

    conf = SparkConf()
    conf.setAppName("Combine example")
    sc = SparkContext(conf=conf)

    # reduce logging
    log4j = sc._jvm.org.apache.log4j
    log4j.LogManager.getRootLogger().setLevel(log4j.Level.ERROR)
```

```
    # Open the input file
    data = sc.textFile("./trx-sampled.csv")

    # Split file by transaction_id and item
    pairs_id_items = data.map(lambda line: line.strip().split(',')).
                                          map(lambda x: (x[0], x[1]))

   # combine
    trxs = pairs_id_items.combineByKey(lambda x: [ x ],
                                     lambda x, y: x + [y],
                                     lambda x, y: x + y )
  # remove duplicates
    items = trxs.map(lambda trx : list(set(trx[1])))

    # write the output file
    its = items.collect()
    num_trxs = len(its)

    # write outputs
    output_file='tx-joined.csv'
    with open(output_file, 'w',) as f_out:
        writer = csv.writer(f_out, delimiter=',')
        for tx in its:
            writer.writerow(tx)

    sc.stop()
    print("done")
```

Combine data with spark

- For really-big file (427MB)
  - Spark: 1m 4s
  - Itertools: 57s
  - Pandas: 20s

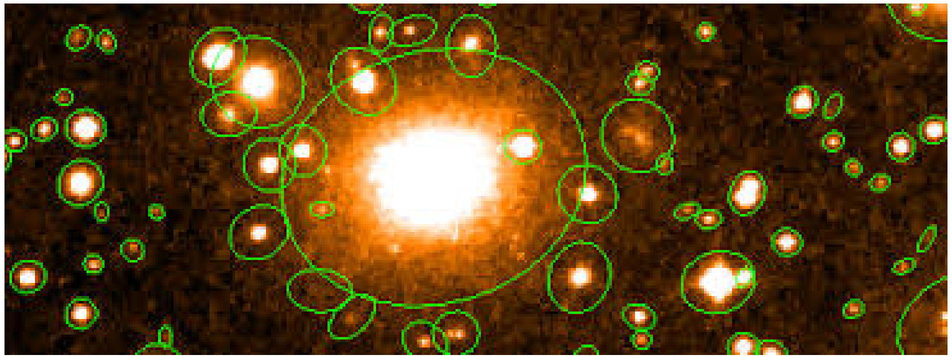
It is really worth to use spark then ?

- In this case no
- Buy when data is big enough as to not fit into memory of a single computer. Spark worth in order to use distributed memory

Knowing each other with Spark

Data Orchestration

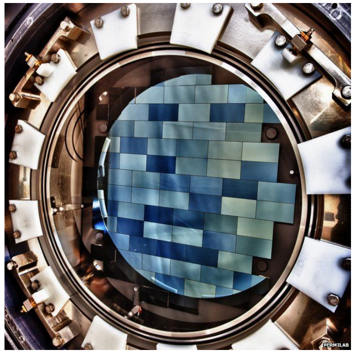
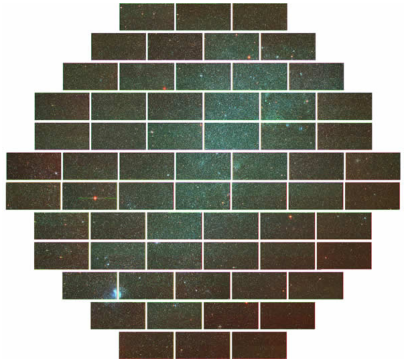
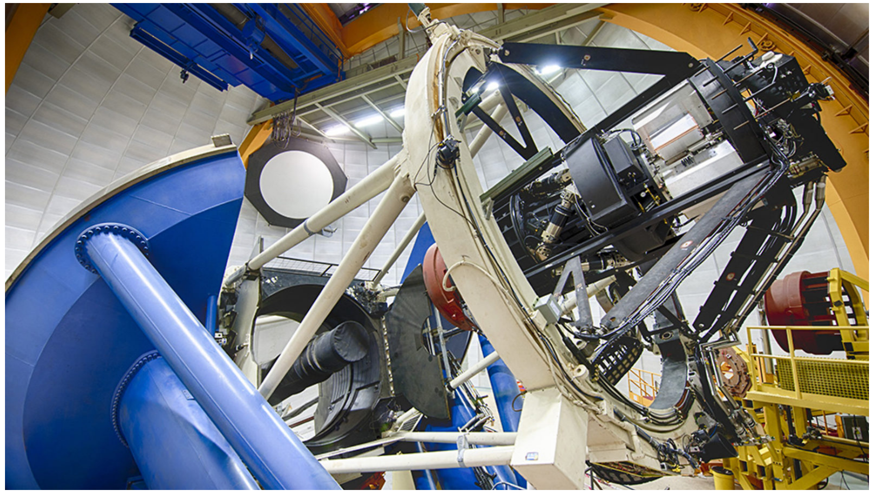
The problem: Identifying sources of light in astronomy



- Known as Source extraction
- First step for catalog building
- Background determination and object discrimination by using a neural network.
- Most popular tool: sextractor (https://www.astromatic.net/software/sextractor)

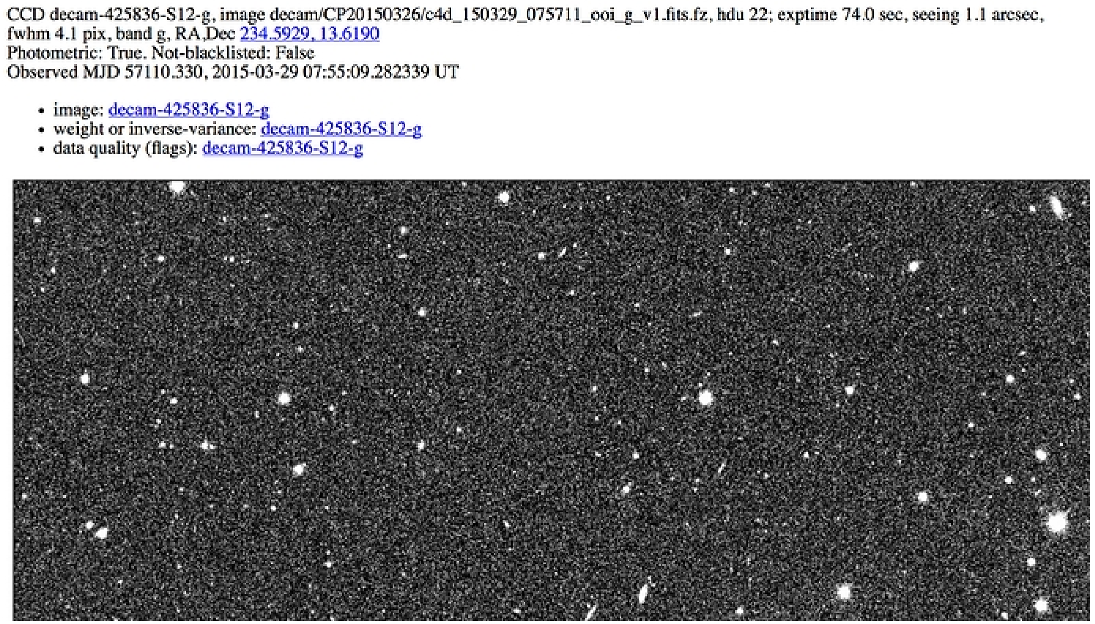
The data: astronomical images

- Single filter images
- Each pixel counts the number of photons received.
- Mosaic Images
  - Decam: 64 CCDs
    - 8Mpix each one
    - 540 Mpix in total
  - Suprime-Cam: 10 CCDs
    - 8 Mpix each one
    - 80 Mpix in total
  - PAN-STARRS GPC1: 60 CCDs
    - 16Mpix each one
    - 1.4 Gpix in total

The data: astronomical images

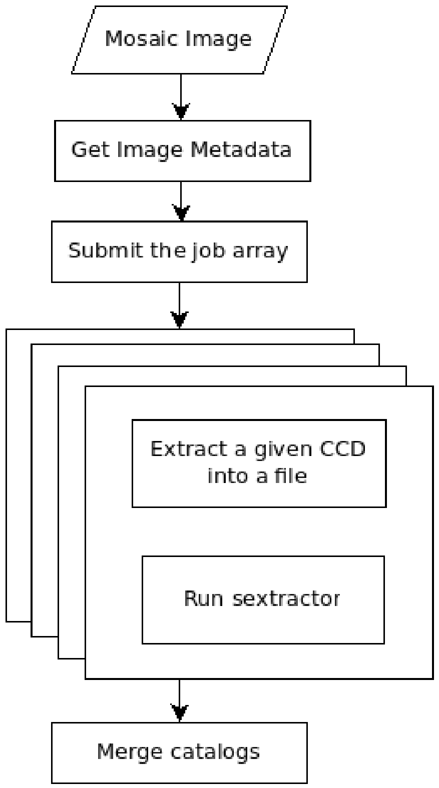
- Decam Image:
  - Header and Data
  - FITS format
  - 60 HDUs
    (4 ccds are broken)
  - 600MBytes each image
- tools
  - Astropy and sextractor



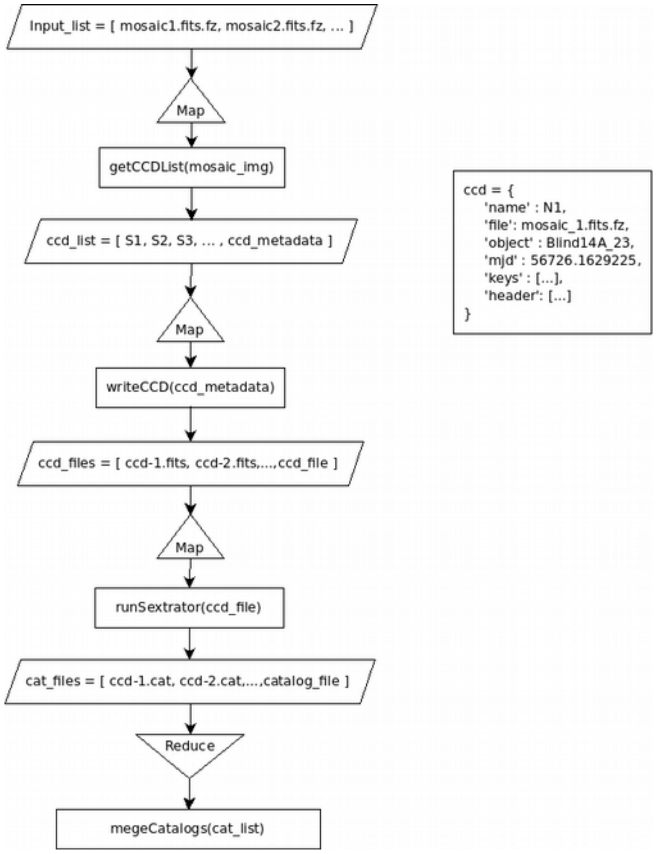
CCD decam-425836-S12-g, image decam/CP20150326/c4d_150329_075711_ooi_g_v1.fits.fz, hdu 22; exptime 74.0 sec, seeing 1.1 arcsec, fwhm 4.1 pix, band g, RA,Dec 234.5929, 13.6190
Photometric: True. Not-blacklisted: False
Observed MJD 57110.330, 2015-03-29 07:55:09.282339 UT

- image: decam-425836-S12-g
- weight or inverse-variance: decam-425836-S12-g
- data quality (flags): decam-425836-S12-g

Extracting sources from several mosaic images



Using the RDD to
orchestrate data processing


Input_list = [mosaic1.fits.fz, mosaic2.fits.fz, ...]
Map
getCCDList(mosaic_img)
ccd_list = [S1, S2, S3, ... , ccd_metadata]
Map
writeCCD(ccd_metadata)
ccd_files = [ccd-1.fits, ccd-2.fits,...,ccd_file]
Map
runSextrator(ccd_file)
cat_files = [ccd-1.cat, ccd-2.cat,...,catalog_file]
Reduce
megeCatalogs(cat_list)

```
ccd = {
    'name' : N1,
    'file': mosaic_1.fits.fz,
    'object' : Blind14A_23,
    'mjd' : 56726.1629225,
    'keys' : [...],
    'header': [...]
}
```

Orchestrating processes with Spark

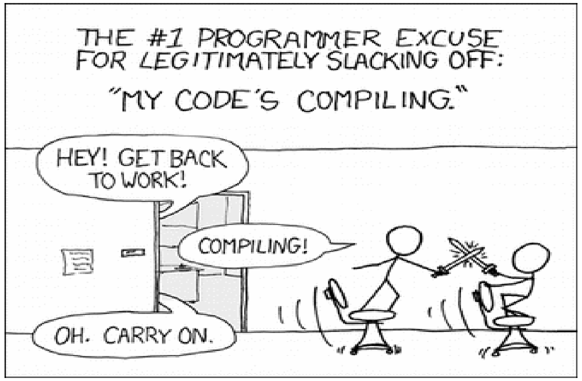
```
# Distributed Sextractor using Spark
# Simple Example 1
# JcM
from pyspark import SparkContext
import pyfits
import os

def getCCDList(file):
    hdulist = pyfits.open(file)
    prihdr = hdulist[0].header
    num_ccds = prihdr["NEXTEND"]
    hdu_list = [];
    for idx, hdu in enumerate(hdulist):
        name = hdu.name
        keys = hdu.header.ascard
        print idx, name, len(keys)
        if idx != 0:
            hdu_list.append({
                'id':idx,  'file':file, 'name':hdu.name, 'header':keys, 'object':prihdr['OBJECT'],
                  'mjd':prihdr['MJD-OBS'], 'key_num': len(keys)})

    hdulist.close()
    return hdu_list

def writeCCD(ccd_handler):
    data = pyfits.getdata(ccd_handler['file'], extname=ccd_handler['name'])
    hdu = pyfits.ImageHDU(data)

    ccd_file = "%s-%s-%s.fits" %(ccd_handler['object'],
              ccd_handler['name'],ccd_handler['mjd'])
    for card in ccd_handler['header']:
        hdu.header.append(card)

    hdu.writeto(ccd_file)
    ccd_handler["ccd_file"] = ccd_file
    return ccd_handler
```

```
def runSextractor(ccd_handler):
    catalog_file="%s.catalog" %(ccd_handler["ccd_file"])
    cmd="sextractor %s -c etc/default.sex -CATALOG_NAME %s"
             %(ccd_handler["ccd_file"],catalog_file)
    os.system(cmd)
    ccd_handler["catalog"] = catalog_file
    return ccd_handler

def mergeCatalogs(cats):
    merged_catalog = "%s.catalog" % (cats[0])

    cmd = "cat "
    for c in cats[1]:
        cmd = "%s %s" %(cmd,c)

    cmd = "%s > %s" %(cmd, merged_catalog)
    os.system(cmd)
    return merged_catalog
######################### MAIN ######################

print "Distributed Sextractor"
sc = SparkContext("local[4]", "Distributed Sextractor")

in_files = [ 'in/tu2208329.fits.fz', 'in/tu2214935.fits.fz', 'in/tu2216725.fits.fz' ]
ccds = sc.parallelize(in_files).flatMap(getCCDList).collect()
fits = sc.parallelize(ccds).map(writeCCD).collect()
cats_per_object = sc.parallelize(fits).map(runSextractor).
                map(lambda o: (o['object'], [ o['catalog'] ])).
                reduceByKey(lambda a,b: a+b ).collect()

cat_list = sc.parallelize(cats_per_object).map(mergeCatalogs).collect()

print cat_list
print "Done"
```

Finalizing ….

HPC programming with Python et al.
Compile! Compile! Compile!

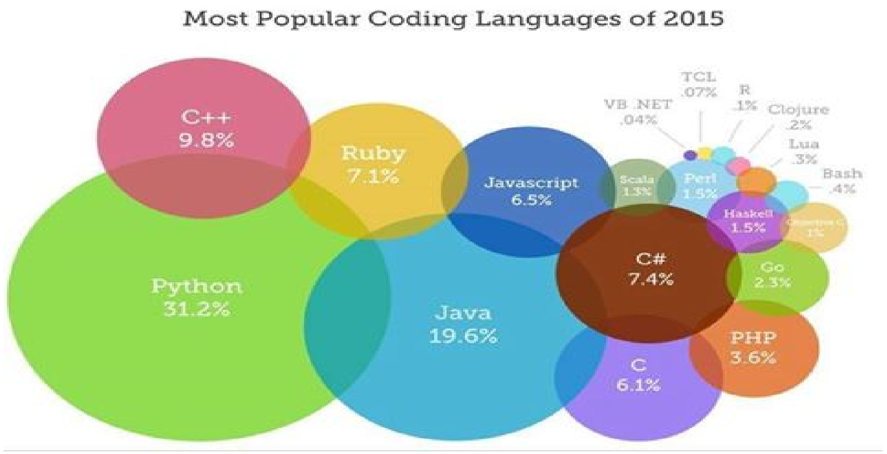
- How to make python run faster?
  - Use optimized versions of Numpy/Scipy
    - They use Blas, Lapack, fftw, TBB, GMP, etc.
  - Each python module based on C/Fortran code should be compiled with optimization flags.
    - MKL de Intel → blas, fftw, tbb, gmp all-in-one!!
    - CuBLAS de Cuda → blas optimized with GPU!!!
    - Compile targeting your processor (mostly intel)
      - Fp model → strict, precise, source, fast
      - Compilation Flags : xHost, O3, ip, etc.
  - Build your tool-chain from scratch (compiling with optimization flags)
  - Write in C/Fortran your critical functions and wrap them to python
  - **Not an easy task**, but it is the shortest way to get a free-ride to performance.

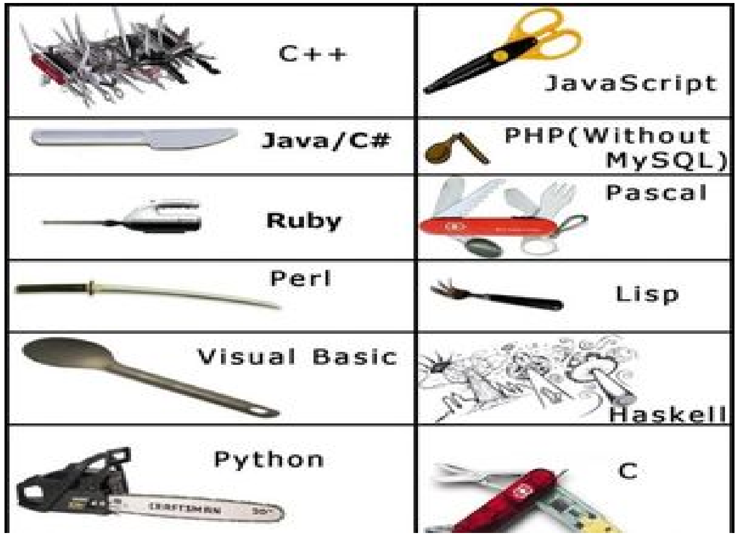


Practical Advices

Languages

“Choose wisely the weapon you will use at each battle.“

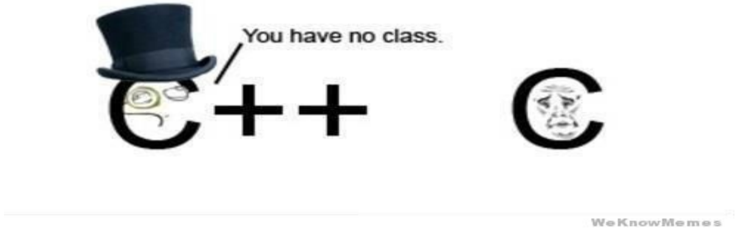


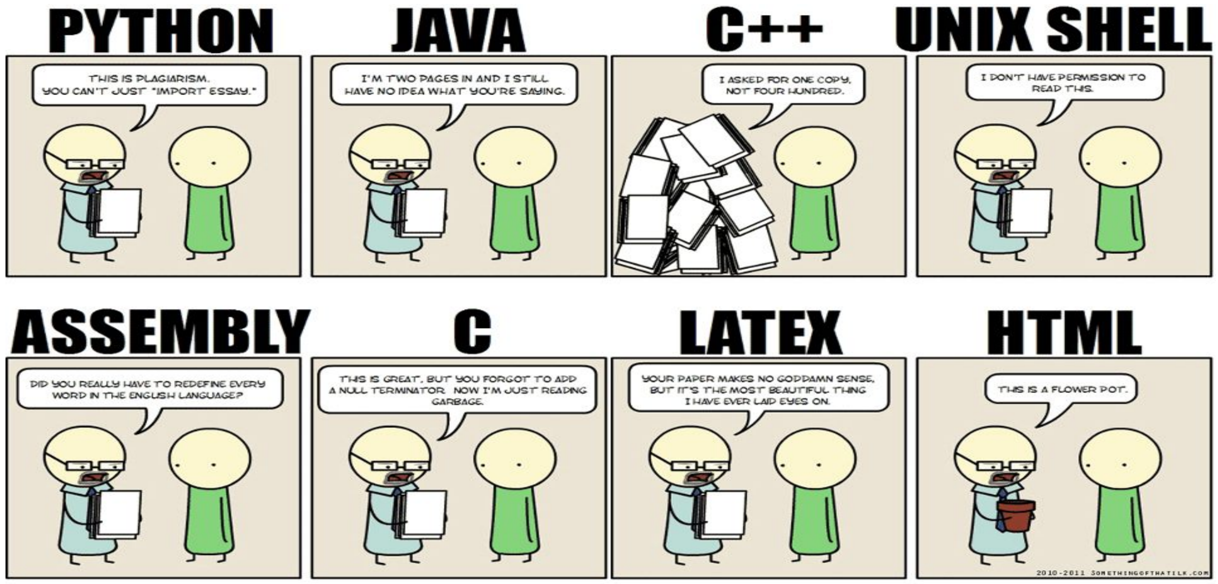


Practical Advices

Languages

“Learn several languages”




PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.
2010-2011 SOMETHINGOFTHATILK.COM

Center for Mathematical Modeling
University of Chile

HPC 123

Scientific Computing on HPC systems

Module 2/2

Thanks for your attention

By
Juan Carlos Maureira B.
<*jcm@dim.uchile.cl*>

v1.0 - 10/08/2018