
CLUSTER

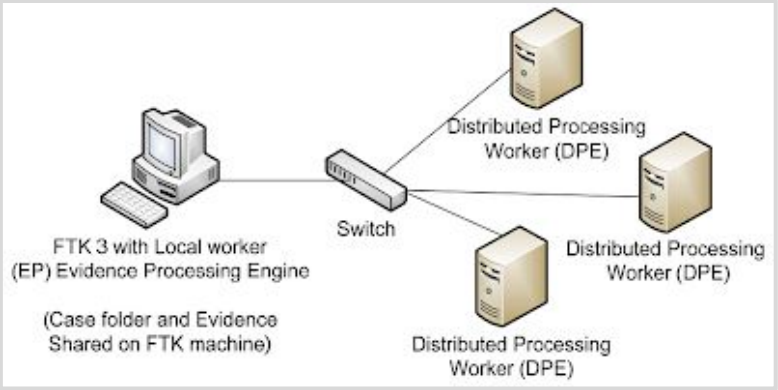
# Tipo de Cluster

## Cluster de Processamento

*“Uma grande tarefa computacional pode ser dividida em pequenas tarefas que são distribuídas ao redor das estações (nodos), como se fosse um supercomputador massivamente paralelo.”*

## Algoritmos Rodados:

- Soma dos números primos
- Quebra de md5



# Materiais

- 4 x Raspberry Pi 3 (B+)
- 4 x MicroSD SanDisk 16 GB
- Desktop Switch TP-LINK
- 4 x Cabos de rede

**Custo Estimado: R$ 1135,99**

# Computação Paralela

Uso do processamento de elementos simultâneos para a resolução eficiente de um problema.

Exemplos de aplicações são:

- Databases e Data mining
- Simulação de sistemas em tempo real
- Gráficos avançados e Realidade Virtual

# Parallel Python

PP é um módulo de python que provê mecanismos para execução paralela de código em SMP (Sistemas com Processadores Múltiplos) e clusters.

# Parallel Python - Algoritmo de Break de MD5

```python
import math, sys, md5, time
import pp

def md5test(hash, start, end):
    """Calculates md5 of the integerss between 'start' and 'end' and compares
    for x in xrange(start, end):
        if md5.new(str(x)).hexdigest() == hash:
            return x

print """Usage: python reverse_md5.py [ncpus]
    [ncpus] - the number of workers to run in parallel,
    if omitted it will be set to the number of processors in the system
"""

# tuple of all parallel python servers to connect with
ppservers = ("192.168.0.41","192.168.0.42","192.168.0.43")
#ppservers = ("10.0.0.1",)

if len(sys.argv) > 1:
    ncpus = int(sys.argv[1])
    # Creates jobserver with ncpus workers
    job_server = pp.Server(ncpus, ppservers=ppservers)
else:
    # Creates jobserver with automatically detected number of workers
    job_server = pp.Server(ppservers=ppservers)

print "Starting pp with", job_server.get_ncpus(), "workers"

#Calculates md5 hash from the given number
hash = md5.new("1829182").hexdigest()
print "hash =", hash
#Now we will try to find the number with this hash value
```

# Parallel Python - Algoritmo de Break de MD5

```
start_time = time.time()
start = 1
end = 2000000

# Since jobs are not equal in the execution time, division of the problem
# into a 100 of small subproblems leads to a better load balancing
parts = 128

step = (end - start) / parts + 1
jobs = []

for index in xrange(parts):
    starti = start+index*step
    endi = min(start+(index+1)*step, end)
    # Submit a job which will test if a number in the range starti-endi has given md5 hash
    # md5test - the function
    # (hash, starti, endi) - tuple with arguments for md5test
    # () - tuple with functions on which function md5test depends
    # ("md5",) - tuple with module names which must be imported before md5test execution
    jobs.append(job_server.submit(md5test, (hash, starti, endi), (), ("md5",)))

# Retrieve results of all submited jobs
for job in jobs:
    result = job()
    if result:
        break
```

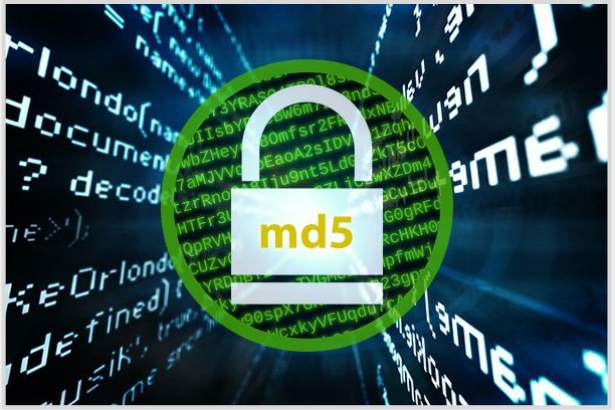
```
# Print the results
if result:
    print "Reverse md5 for", hash, "is", result
else:
    print "Reverse md5 for", hash, "has not been found"

print "Time elapsed: ", time.time() - start_time, "s"
job_server.print_stats()
```

# Execução

Senha: **ecm254Apresentacao**

Hash MD5 utilizada:
**a4957c9277761bfaae209ad4298990e2**




1 Raspberry

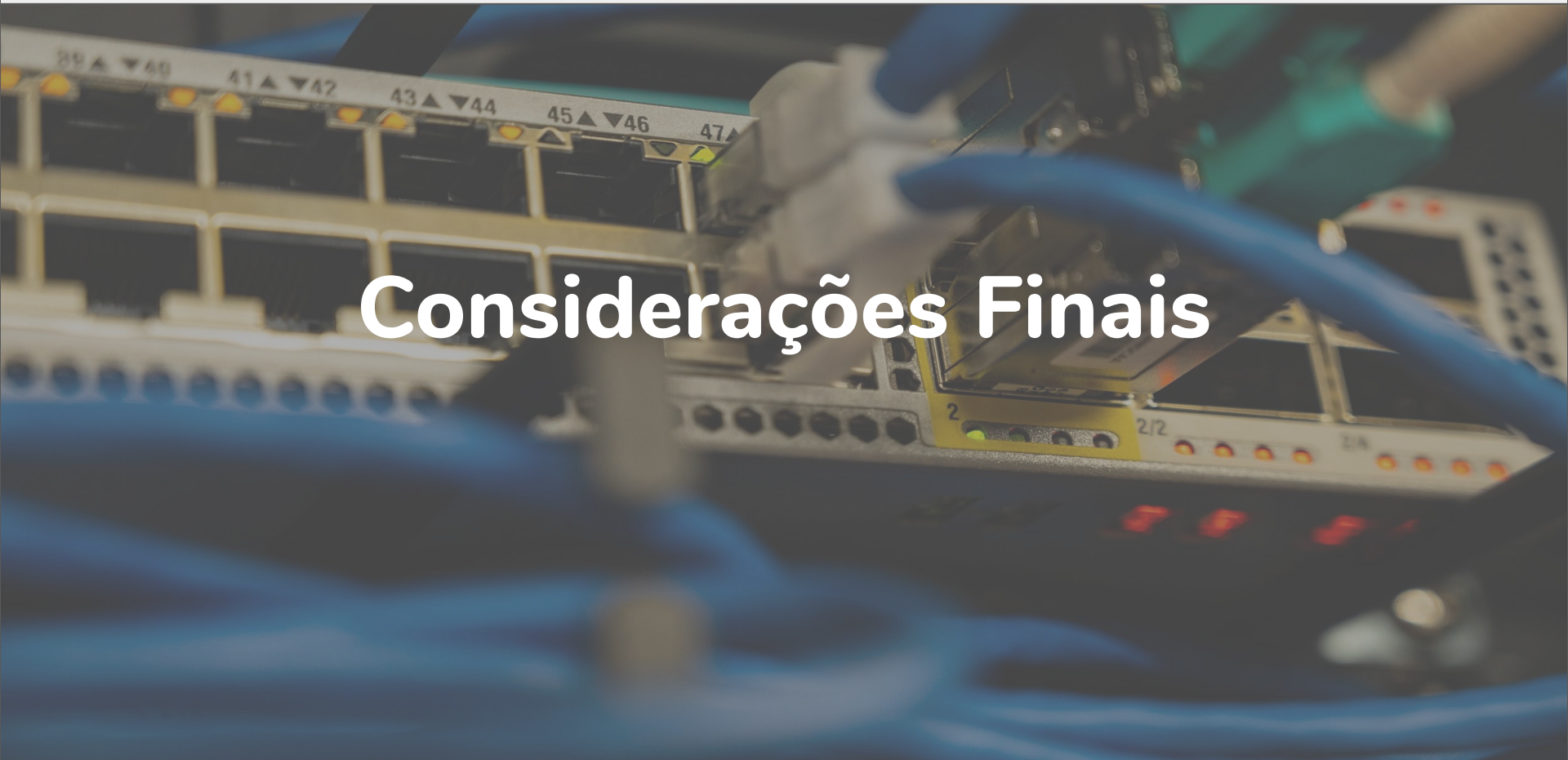
Tempo de execução: 4,14 s

Cluster (4 Raspberry's)

Tempo de execução: 1,78 s

Ganho de **232,6%** de velocidade de execução

# Vídeo - Execução do Cluster



# Considerações Finais

# Obrigado

| | |
|---|---|
| Breno Thomaz Savanini | 17.00815-8 |
| Guilherme Tagliati da Silva | 17.00375-0 |
| Júlia Catarina | 16.00645-3 |
| Lucas Negresiolo Libório | 17.01165-5 |
| Lucas Primati Menezes | 16.00683-6 |
| Raphael de Jesus | 16.00378-0 |