Engenharia de Computação

Especialização Lato Sensu em Ciência de Dados e Analytics

Soluções em Processamento para Big Data

{ Hadoop }

Prof. Jairson Rodrigues
jairson.rodrigues@univasf.edu.br

{ hadoop }

AGENDA

- HDFS
- MapReduce

{ o que é hadoop? }

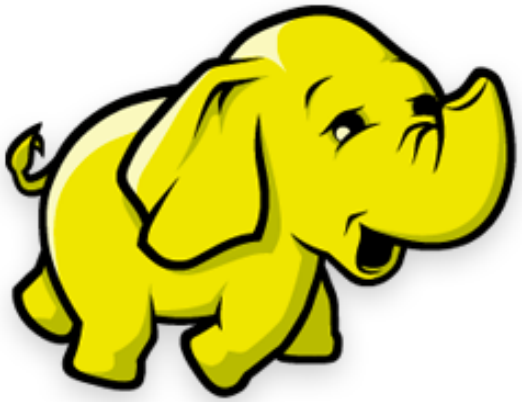
A biblioteca de software Apache Hadoop é um framework para processamento distribuído de grandes conjuntos de dados através de clusters de computadores usando modelos de programação simples. É projetado para ganhar escala a partir de servidores individuais até milhares de máquinas, cada uma oferecendo computação e armazenamento local. Melhor que confiar em hardware para alcançar alta disponibilidade, a biblioteca por si só é projetada para detectar e gerenciar falhas na camada de aplicação. [1]

{ hadoop - primórdios }

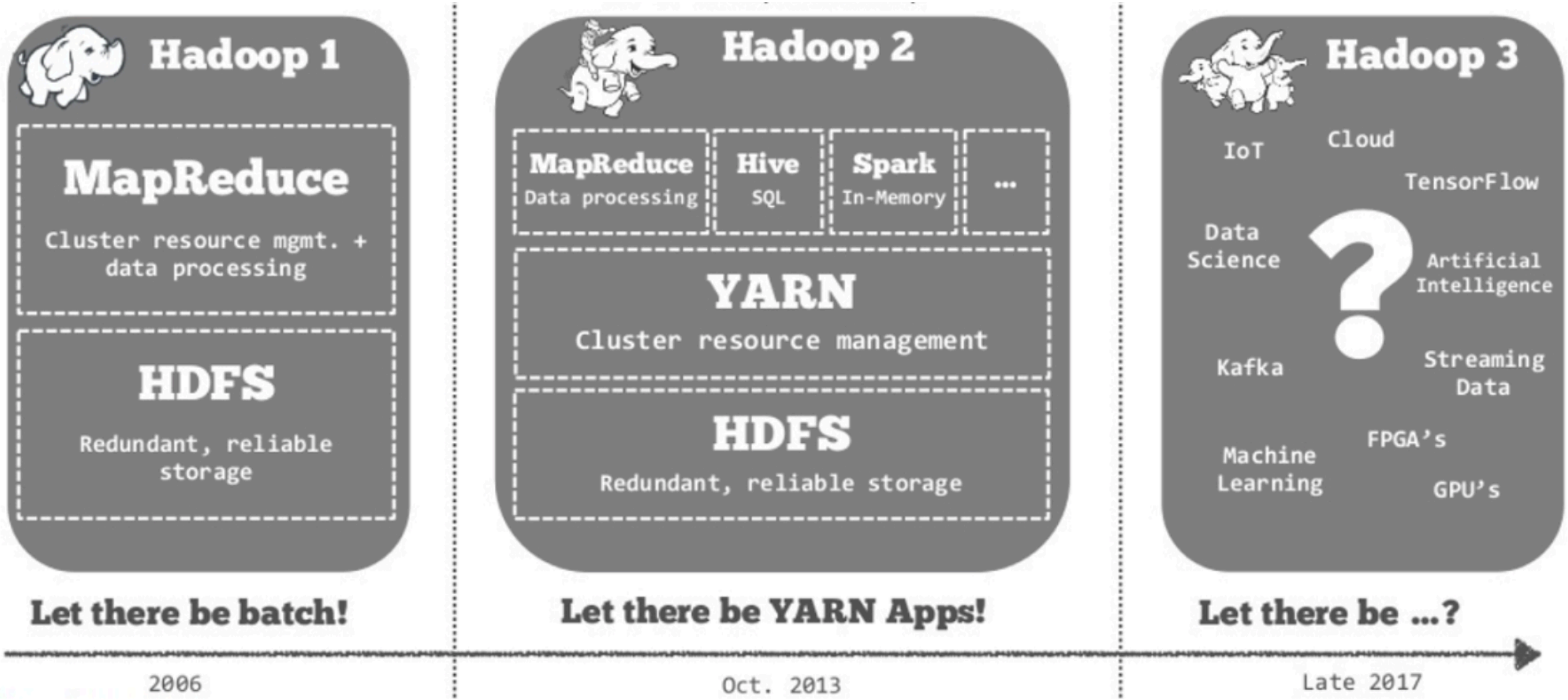
- 2003 - Google publica artigo sobre o GFS (SOSP'03) [3]
  - The Google file system
  - http://dl.acm.org/citation.cfm?id=945450
- 2004 Google publica artigo sobre o MapReduce (OSDI'04) [4]
  - MapReduce: simplified data processing on large clusters
  - http://dl.acm.org/citation.cfm?id=1327492

{ hadoop - primórdios }

- 2005 - Doug Cutting e Mike Cafarella criam o Hadoop
- O nome do projeto advém do nome do elefante de brinquedo do filho de Doug.



{ hadoop vs hadoop }



{ hadoop - módulos }

- Hadoop Common
- Hadoop Distributed File System - HDFS
- Hadoop YARN
- Hadoop MapReduce

{ hadoop - premissas e metas }

- Falha de hardware é norma, não exceção
- Ênfase na alta vazão de dados ao invés de baixa latência
- Arquivo típico possui gibabytes ou terabytes / grandes volumes
- Modelo simples e coerente, escreve uma vez, leia muitas, acrescente (append) quando necessário
- Mover a computação é mais barato que mover dados
- Portabilidade entre plataformas de hardware e software

{ quem usa hadoop? [2] }

- Facebook
  - Maior cluster: 1100 máqunas, 8800 núcleos, 12 Pb
  - Segundo maior cluster: 300 máquinas, 2400 cores, 3 Pb
  - Nó individual: 8 núcleos, 12 Tb HD
  - Aplicação: armazenar cópias de log interno para relatórios/análise e aprendizagem de máquina
- Yahoo
  - + 100 mil núcleos, + de 40 mil máquinas
  - Maior cluster: 4.500 máquinas, 36 mil núcleos, 17 Pb
  - Nó individual: 8 núcleos, 4 Tb HD, 16 Gb RAM
  - Aplicação: buscador e suporte à pesquisa para publicidade
  - + 60% dos jobs Yahoo são scripts Apache Pig Latim

{ quem usa hadoop? [2] }

- Alibaba
  - 15 máqunas, 120 núcleos, 11 Tb
  - Nó individual: 8 núcleos, 1.4 Tb HD, 16 Gb RAM
  - Aplicação: pré-processamento de dados para o engenho de busca
- AOL
  - 150 máquinas, 600 núcleos, 117 Tb
  - Nó individual: 4 núcleos, 800 Gb HD, 16 Gb RAM
  - Aplicação: ETL para análise
- eBay
  - 532 máquinas, 4256 núcleos, 5.3 Pb
  - Aplicação: otimização de busca e P&D

{ quem usa hadoop? [2] }

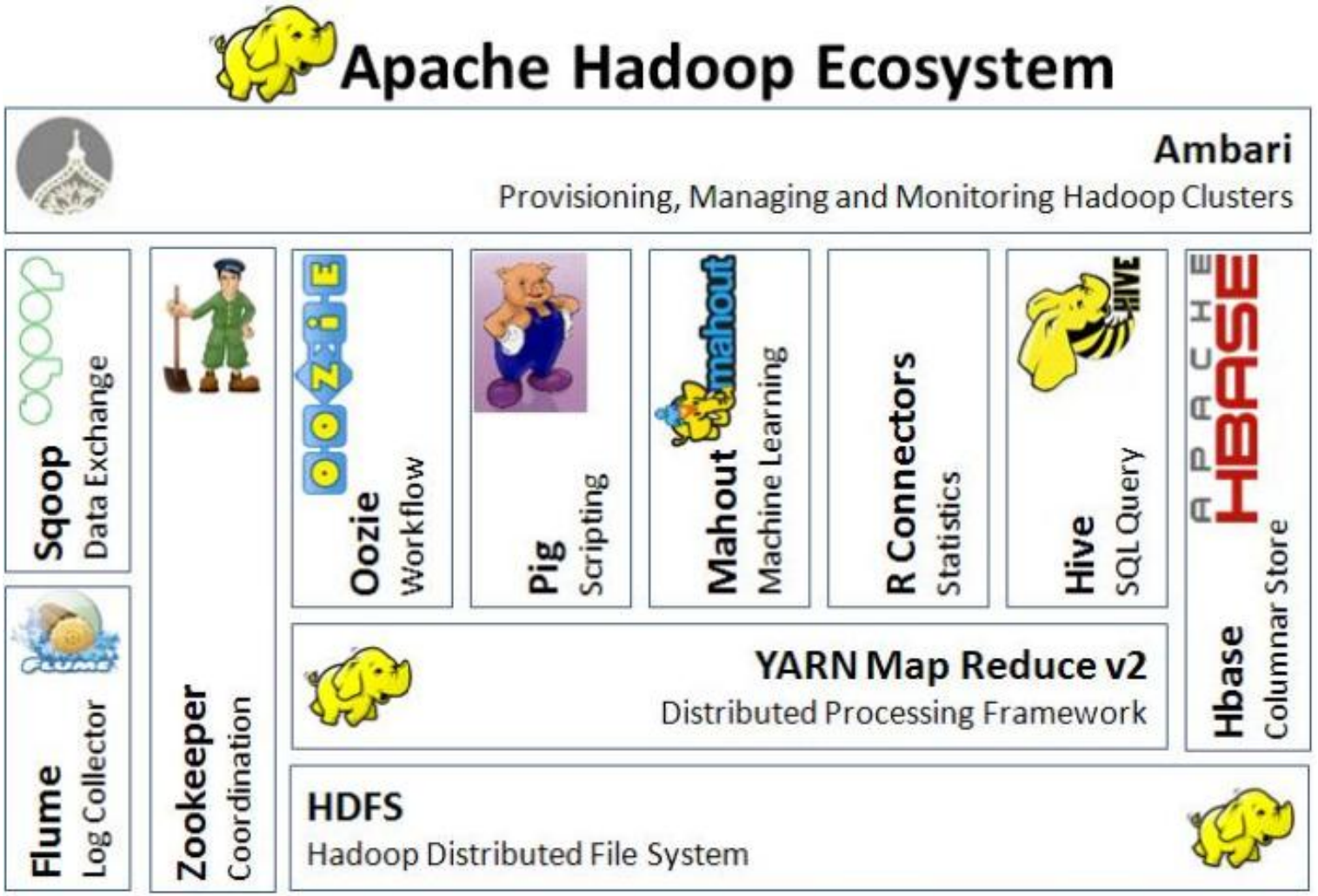
- Mercado Libre
  - 20 máquinas, 240 núcleos, 53.3 Tb
  - Nó individual: 12 núcleos, 2.5 Tb HD, 32 Gb RAM
  - Aplicação: processamento de log de vendas
- Spotify
  - 1650 máquinas, 43 mil núcleos, 70 Tb RAM, 65 Pb
  - Aplicação: geração de conteúdo, relatórios, análise, recomendação de músicas etc.

{ quem usa hadoop? [2] }

- Prof. Jairson, rsrsrsrs
  - Cluster virtual, Fiware, Amazon
  - Aplicação: ensino, execução de ETL e algoritmos de machine learning - naive bayes, random forest, regressão logística etc

- Agora, vocês!
- E uma infinidade de outros mais...
  - https://wiki.apache.org/hadoop/PoweredBy

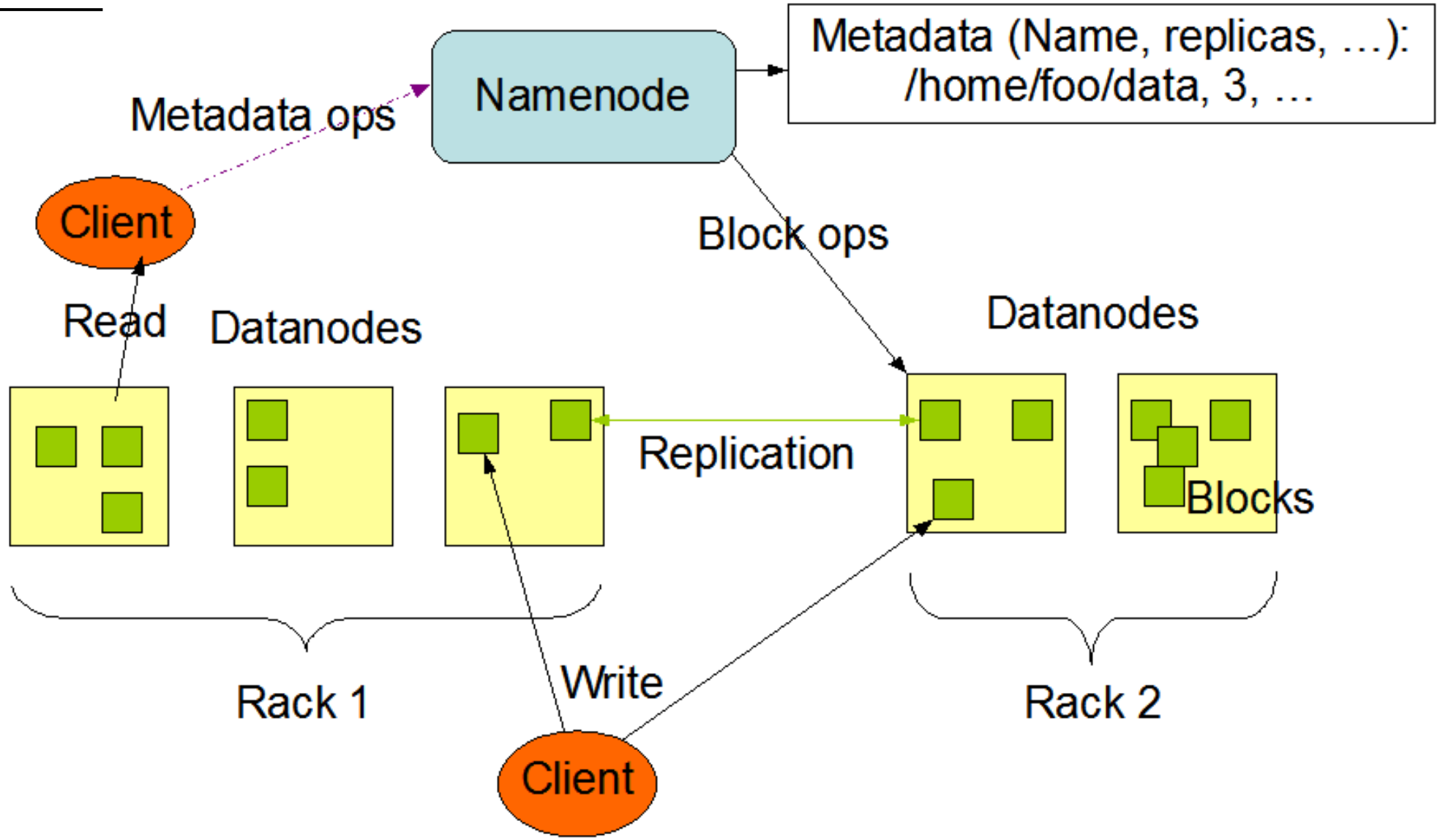
{ ecossistema hadoop }



{ hdfs }


grande
Poli-UPE
variedade
MapReduce
inteligência
máquina
aprendizagem
dados
volume
Hadoop
Latu-Sensu
Especialização
Ciência
velocidade
variabilidade
vocabulário
HDFS
viralidade
validade
veracidade
viscosidade
valor
análise

{ arquitetura HDFS }



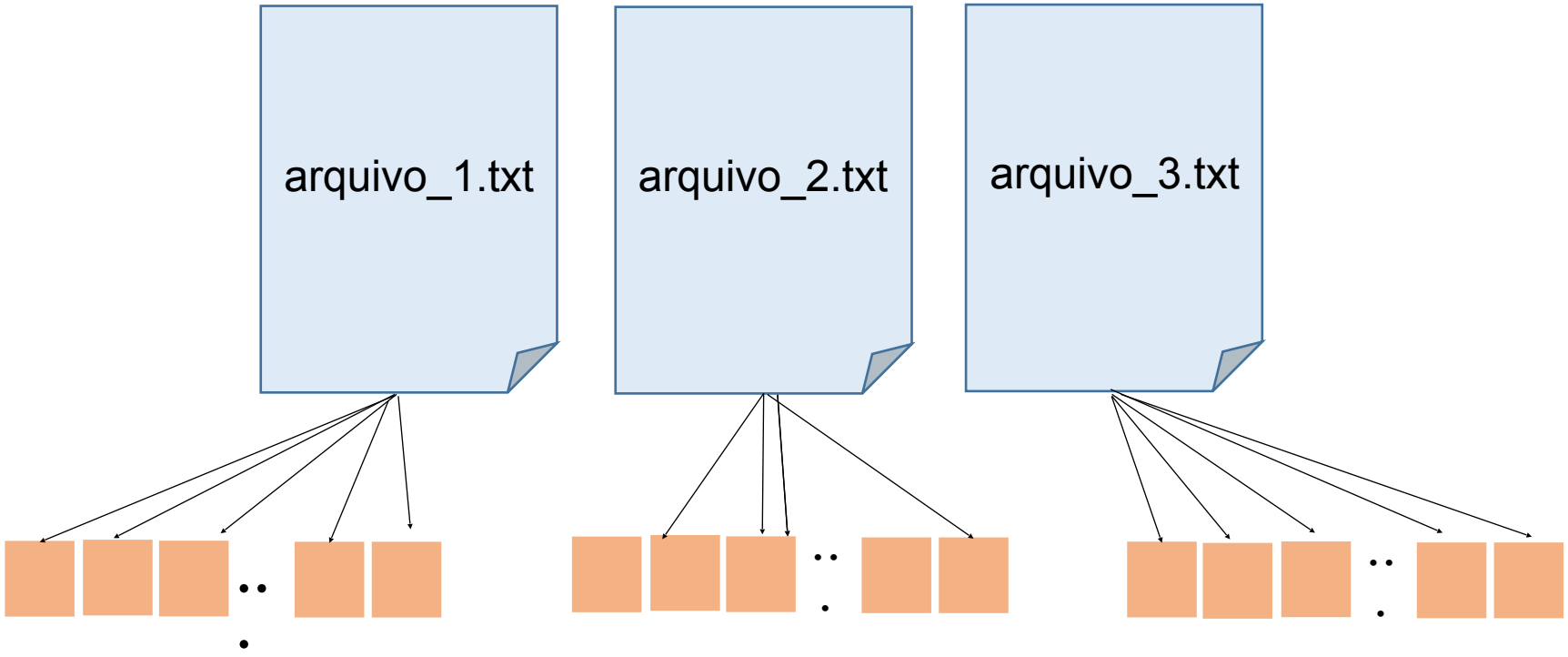
{ namenodes e datanodes }

- Arquitetura mestre/escravo
- Um Namenode
  - gerencia o namespace
  - controla o acesso a arquivos
- N Datanodes
  - geralmente um por nó
  - armazena blocos
  - serve operações de leitura e escrita nos blocos
  - processa tarefas computacionais

{ namenodes e datanodes }

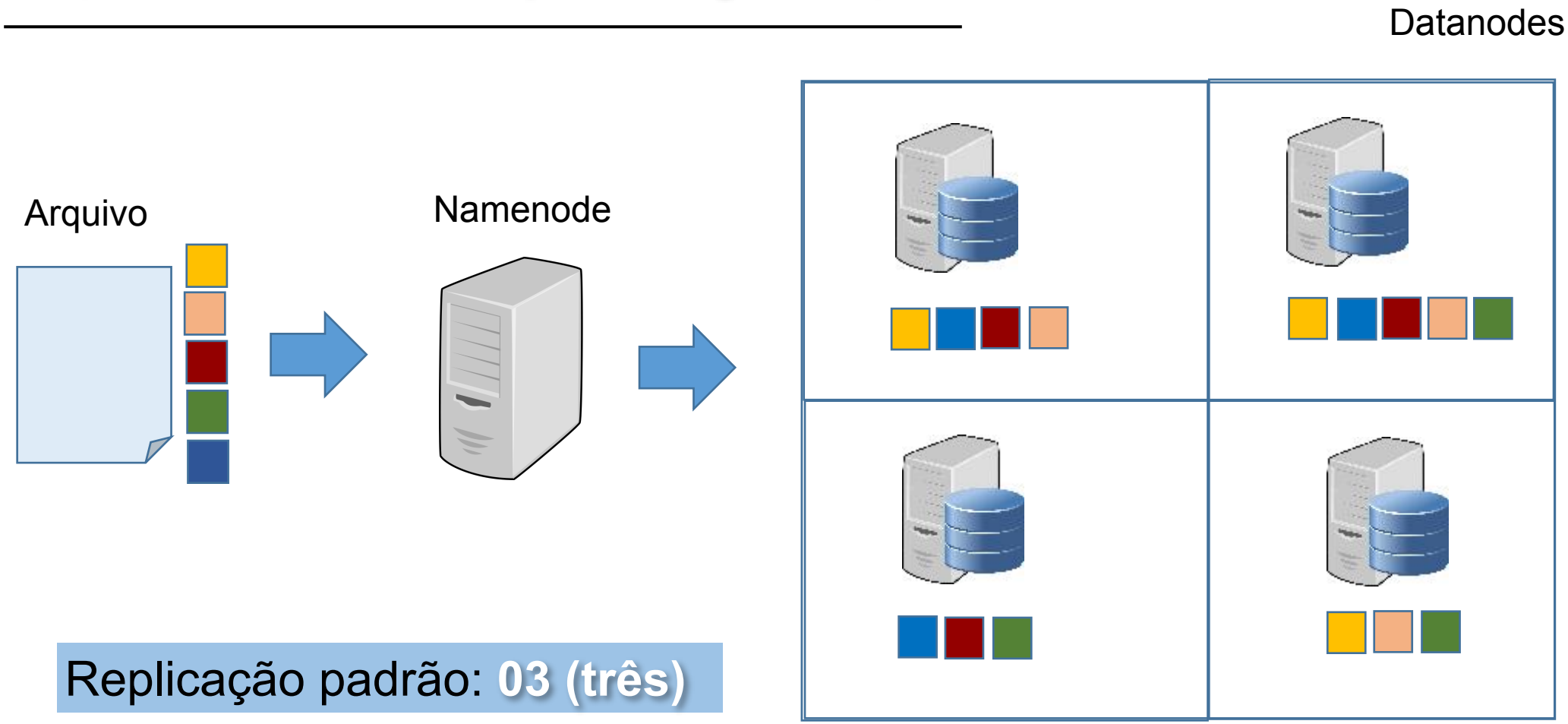
- Tipicamente executam em um sistema operacional GNU/Linux
- HDFS (e Hadoop como um todo) é construído sobre Java
- Namenode é o árbitro das operações e o gerenciador dos metadados sobre o sistema de arquivo

{ blocos de dados }



Tamanho padrão: **128 Mb**
Blocos podem variar, em geral, de 64 Mb a 512 Mb

{ fator de replicação }

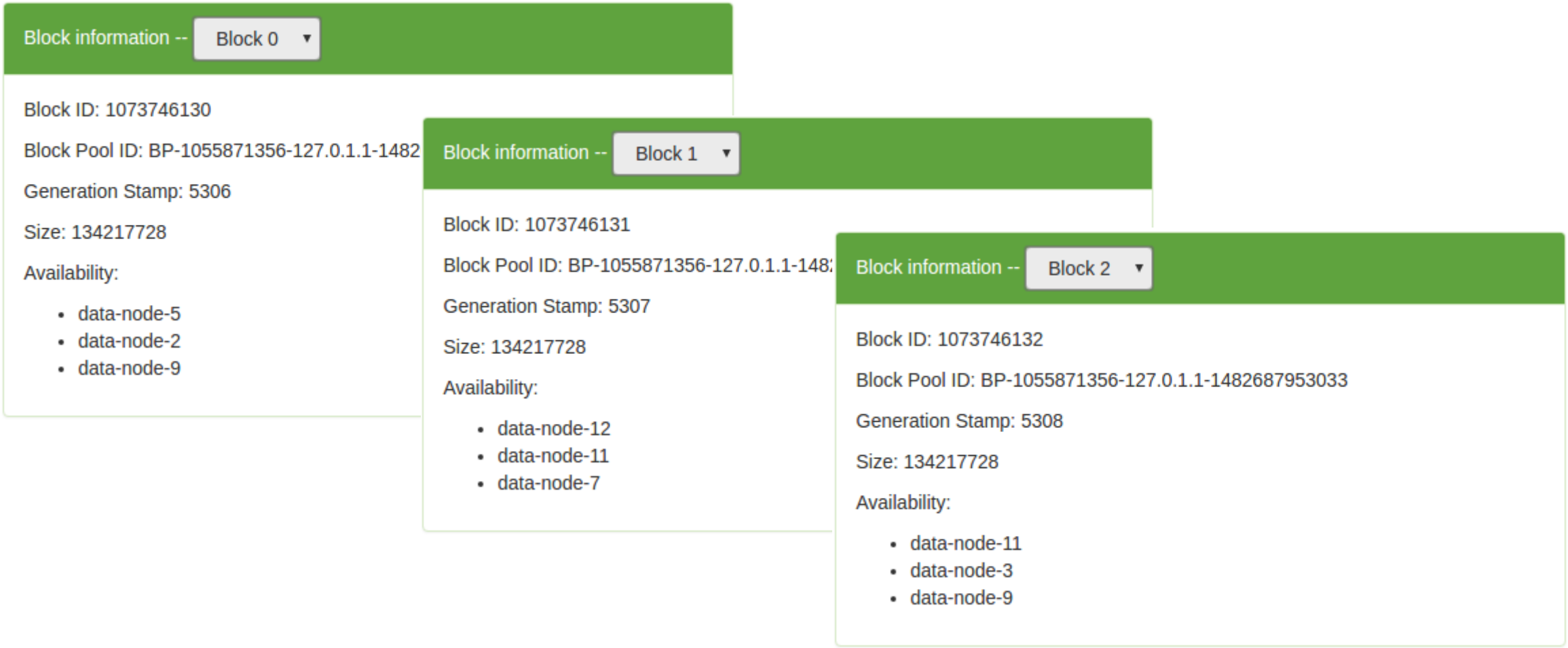


Replicação padrão: **03 (três)**

{ fator de replicação }

- Tamanho padrão de bloco: 128 Mb
- Fator de replicação padrão: 3 datanodes
- Replicação promove tolerância a falhas
- Todos os blocos possuem mesmo tamanho
  - Exceto o último
- A replicação pode ser escolhida por arquivo, no momento da criação e pode ser modificado posteriormente

{ fator de replicação }



{ política de posicionamento de réplica }

- Crítico para confiabilidade e performance
- Grandes clusters possuem muitos racks
  - Comunicação entre racks demanda switches
- Política padrão (FR 3):
  - posiciona o primeiro bloco em um datanode de um rack
  - posiciona o segundo bloco em outro datanode do mesmo rack
  - posiciona o terceiro bloco em outro datanode de outro rack
- Política padrão (FR > 3):
  - A partir do quarto bloco, posicionamento randômico segundo a regra `(replicas - 1) / racks + 2` por rack

{ pergunta }

Qual o máximo número de réplicas que uma instância (cluster) HDFS pode armazenar?

{ falhas - datanode e partição de rede }

- Datanode envia mensagem heartbeat periodicamente para Namenode
- Um datanode ou um subconjunto (partição) de datanodes param de enviar status para Namenode
- Namenode
  - para de enviar I/O para datanode/partição perdida
  - reanalisa que blocos precisam ser re-replicados
- Re-replicação ocorre quando
  - Datanode cai
  - Bloco corrompe
  - HD do datanode falha

{ datanode heartbeat (1) }

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used
data-node-12:50010 (192.168.4.228:50010)	1	In Service	39.34 GB	129.12 MB	5.29 GB	33.92 GB	1	129.12 MB (0.32%)
data-node-5:50010 (192.168.4.232:50010)	0	In Service	39.34 GB	129.13 MB	5.11 GB	34.1 GB	1	129.13 MB (0.32%)
data-node-7:50010 (192.168.4.226:50010)	0	In Service	39.34 GB	197.84 MB	5.29 GB	33.85 GB	2	197.84 MB (0.49%)
data-node-3:50010 (192.168.4.223:50010)	2	In Service	39.34 GB	387.13 MB	4.93 GB	34.03 GB	3	387.13 MB (0.96%)
data-node-2:50010 (192.168.4.222:50010)	2	In Service	39.34 GB	129.13 MB	4.93 GB	34.28 GB	1	129.13 MB (0.32%)
data-node-4:50010 (192.168.4.224:50010)	0	In Service	39.34 GB	129.13 MB	5.29 GB	33.92 GB	1	129.13 MB (0.32%)
data-node-11:50010 (192.168.4.229:50010)	0	In Service	39.34 GB	258.13 MB	4.93 GB	34.16 GB	2	258.13 MB (0.64%)
data-node-9:50010 (192.168.4.225:50010)	0	In Service	39.34 GB	258.12 MB	4.57 GB	34.51 GB	2	258.12 MB (0.64%)
data-node-6:50010 (192.168.4.230:50010)	0	In Service	39.34 GB	129.11 MB	4.57 GB	34.64 GB	1	129.11 MB (0.32%)
data-node-8:50010 (192.168.4.233:50010)	0	In Service	39.34 GB	68.86 MB	4.93 GB	34.34 GB	1	68.86 MB (0.17%)
data-node-1:50010 (192.168.4.227:50010)	0	In Service	39.34 GB	68.83 MB	5.12 GB	34.16 GB	1	68.83 MB (0.17%)
data-node-10:50010 (192.168.4.231:50010)	0	In Service	39.34 GB	258.15 MB	5.29 GB	33.79 GB	2	258.15 MB (0.64%)

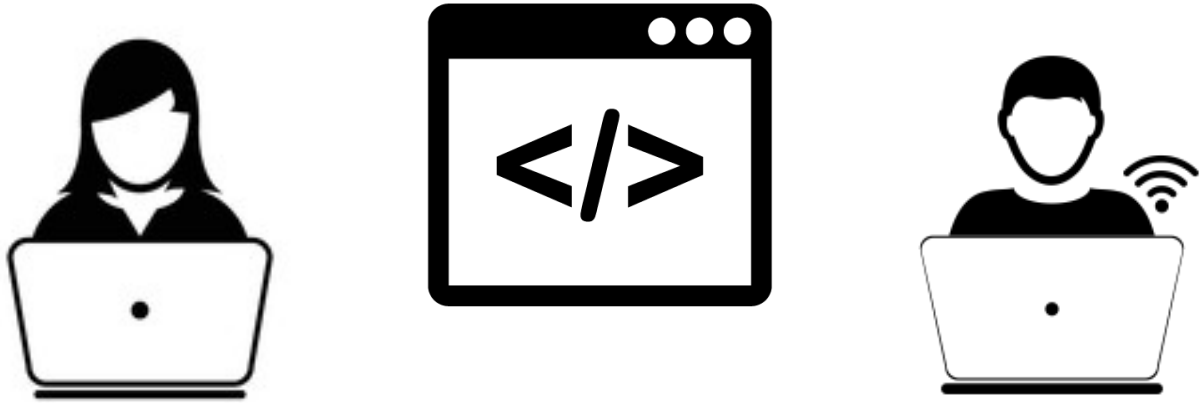
(1) http://81.14.183.23:50070/dfshealth.html#tab-datanode

{ interfaces web }

- Namenode Web Interface:
  - http://192.168.100.101:50070/
- Resource Manager Web Interface
  - http://192.168.100.101:8088/
- Spark Master URL:
  - spark://192.168.100.101:7077/
- HDFS URL:
  - hdfs://192.168.100.101:8020/

{ vamos consolidar a teoria }

Prática: HDFS Shell



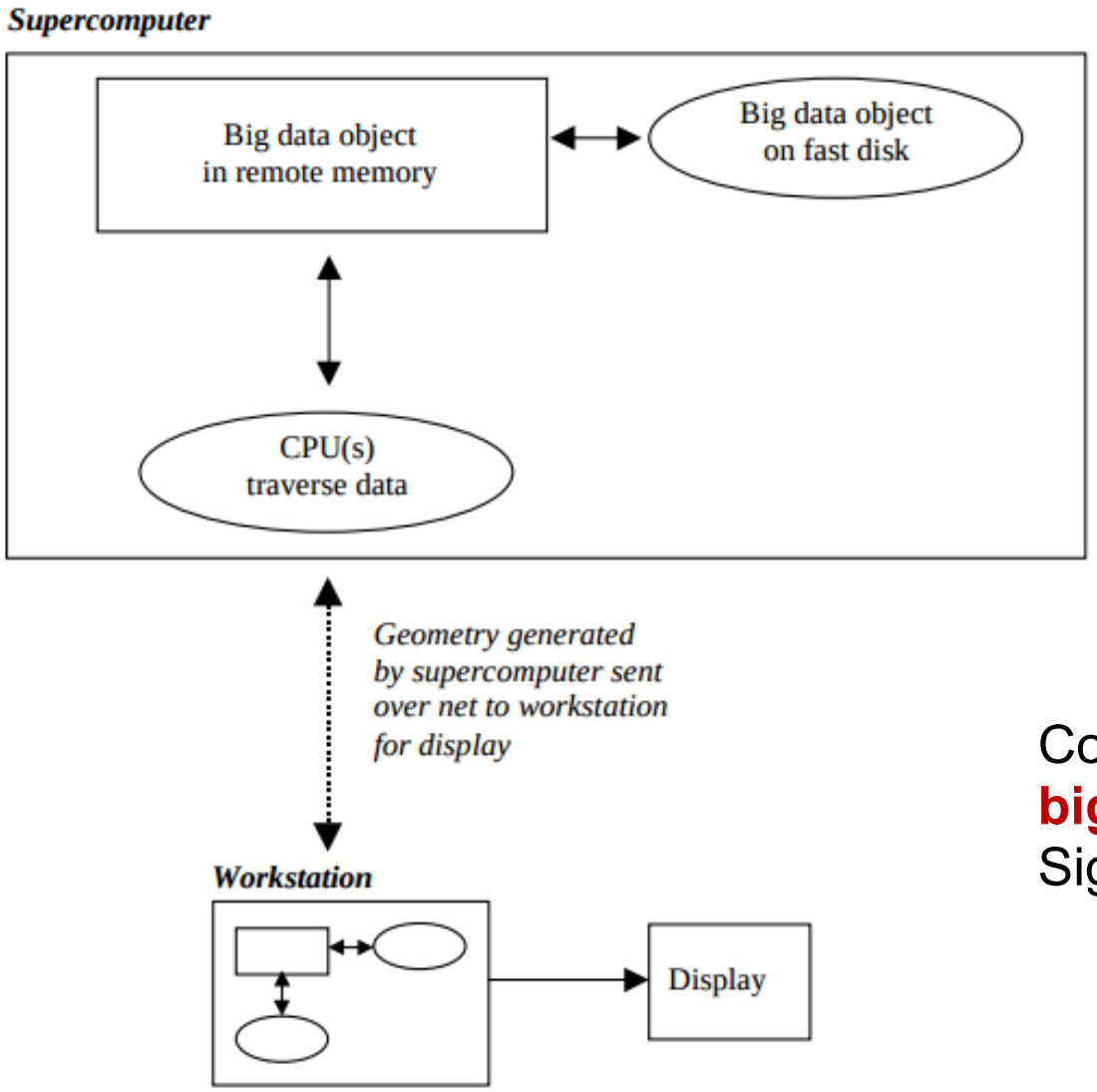
{ mapReduce }


grande
Poli-UPE
variedade
MapReduce
inteligência
máquina
dados
aprendizagem
volume
Hadoop
Latu-Sensu
Especialização
Ciência
velocidade
variabilidade
vocabulário
HDFS
viralidade
validade
veracidade
viscosidade
valor
análise

{ processamento de grandes volumes }

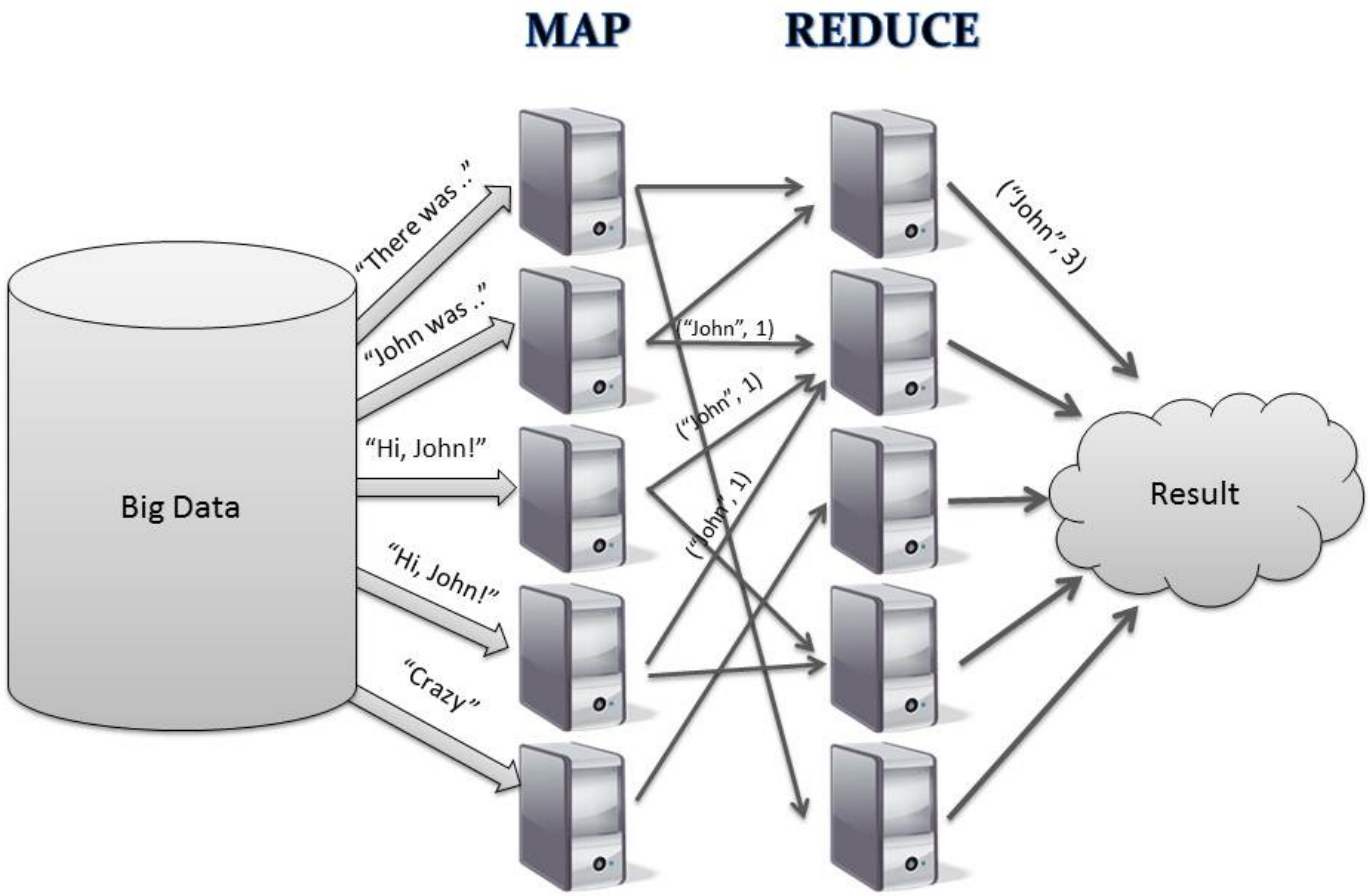
- Uma vez que sabemos armazenar e recuperar 1 Pb...
- Como processar 1 Pb?
- Uma solução => computação paralela distribuída
  - ambientes de memória compartilhada?
  - OpenMP?
  - Message Passage Interface (MPI)?
- Em todos os casos o programador deve se preocupar em gerenciar os recursos
  - mutexes, semáforos, detalhes da infraestrutura adjacente, dependência de plataforma, sistema operacional, linguagem...

{ mover a computação até o dado }

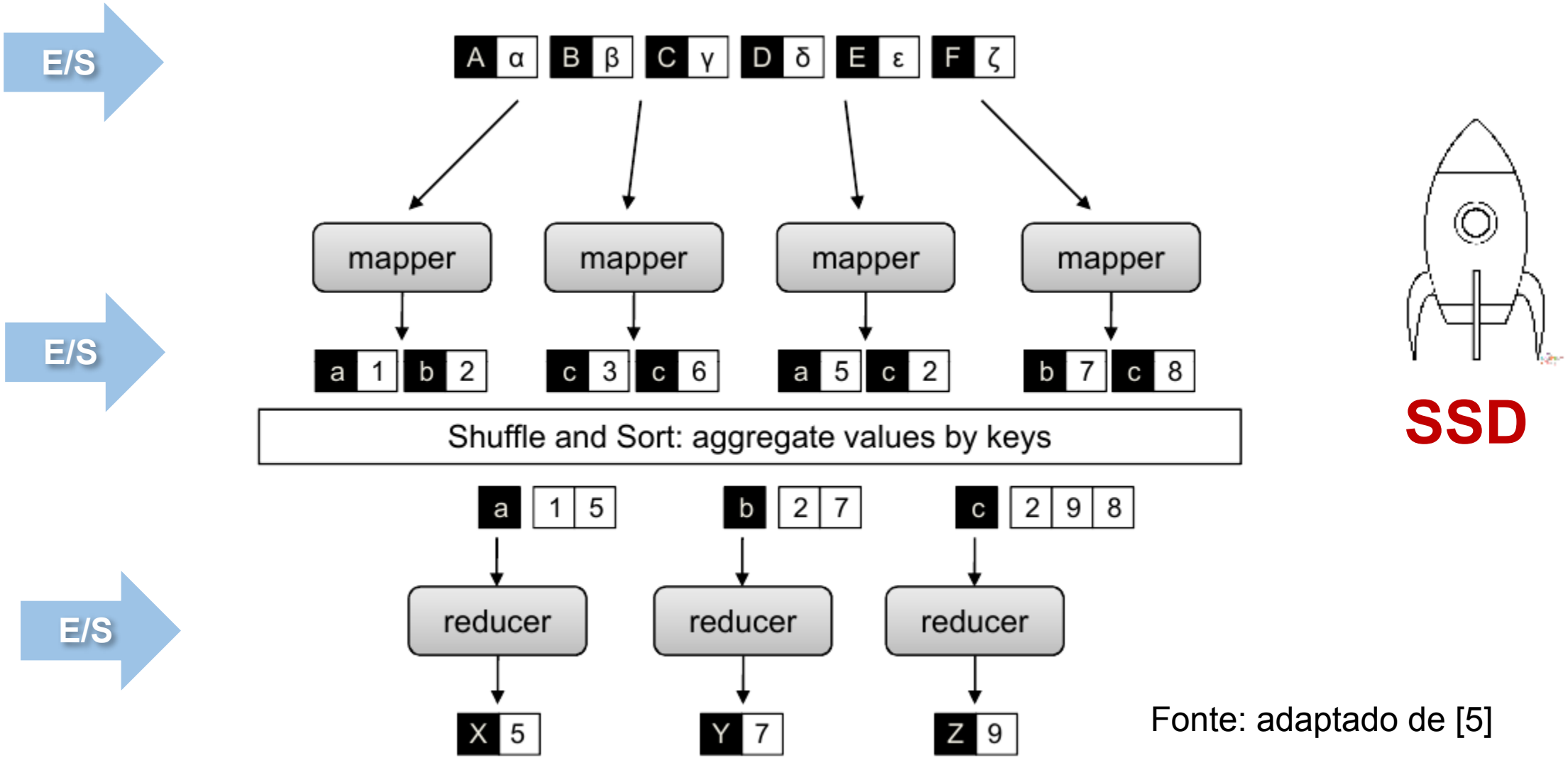


Cox, Michael, and David Ellsworth. "Managing **big data** for scientific visualization." ACM Siggraph. Vol. 97. **1997**.

{ mapreduce - visão simplificada }



{ mapreduce - outra visão simplificada }



SSD

Fonte: adaptado de [5]

{ mapreduce - modelo de programação }

- Opera exclusivamente em pares *<key,value>*
- Framework enxerga a entrada como conjunto de pares *<key,value>* e produz um outro conjunto de pares *<key, value>* como saída
- Inspirando fundamentalmente em paradigmas funcionais
- Formalmente:

map: (k1 , v1) → [(k2 , v2)]
reduce: (k2 , [v2]) → [(k3 , v3)]

{ princípios **bem básicos** de programação funcional [5] }

- Para compreender...
- Imagine somar todos os quadrados de uma lista de números
  - Entrada: 2, 3, 5, 7
  - Processamento: $2^2 + 3^2 + 5^2 + 7^2$
  - Saída: 87
- Vamos processar o mesmo problema com funções de alta ordem
  - map
  - fold (sim, fold)

{ princípios **bem básicos** de programação funcional [5] }

- Dada uma lista
  - map -> recebe como argumento uma função f (que recebe apenas um parâmetro) e a aplica a todos os elementos da lista
  - fold -> recebe como argumento uma função g e um valor inicial, que é aplicado à função g, cujo resultado é acumulado e novamente aplicado/acumulado ao segundo, terceiro e assim por diante a todos os elementos da lista.

{ exemplo ** }

- lista: 2, 3, 5, 7
  - map -> f(x) = x^2
  - fold -> g([f(x)], +)
- resultado do map
  [4, 9, 25, 49]
- aplicação do fold
  4
  4 + 9 = 13
  13 + 25 = 38
  38 + 49 = 87

'+' significa: função operação de adição

** notação adaptada para efeitos didáticos

{ uma abordagem map/reduce em python }

- x = [2, 3, 5, 7]
- y = map(lambda z: z**2, x)
- print y

  [4, 9, 25, 49]
- q = reduce(lambda m,n : m+n, y)
- print q

  87

{ mappers e reducers }

- Pares <key, value> formam a estrutura básica de um programa MapReduce
- Chaves e valores podem ser primitivas básicas como inteiros, floats, strings, bytes ou qualquer estrutura mais complexa, tais como listas, tuplas etc.
- Algumas analogias:
  - Para um conjunto de páginas web
    - key: URL
    - value: conteúdo HTML da página
  - Para um grafo
    - key: ID do nó
    - value: lista de nós adjacentes ao nó (key)

{ mappers e reducers }

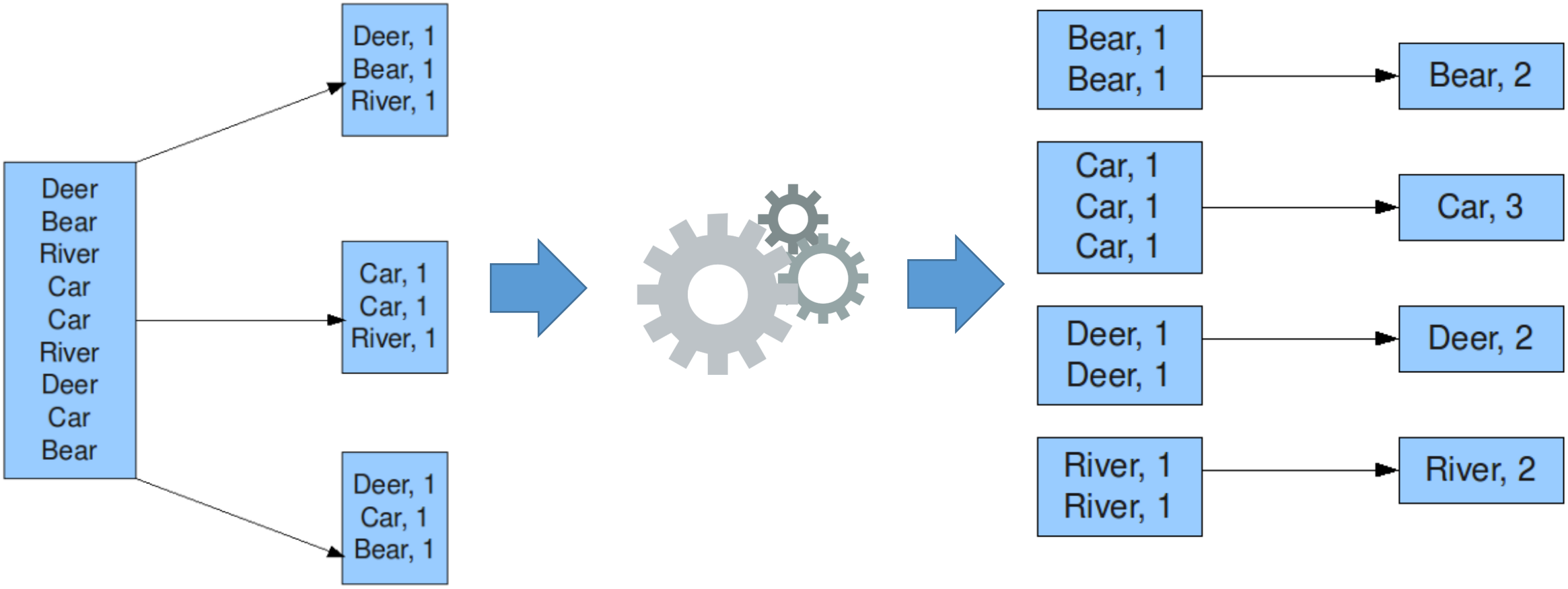
- map é aplicada a todo par <key, value> distribuído ao longo dos blocos do filesystem
- reduce é aplicada a todos os valores associados com chaves (keys) intermediárias geradas por map

{ "hello world" mapreduce / word count }

- Dado um arquivo de entrada, contar a ocorrência de cada palavra
- Entrada 1: bigtext.txt (Projeto Gutemberg - 6Mb)
- Entrada 2: bigtext.txt (SF Bay Area Bike Share - USA - 600Mb)
- Mapper: wordcount_map.py
- Reducer: wordcont_reducer.py

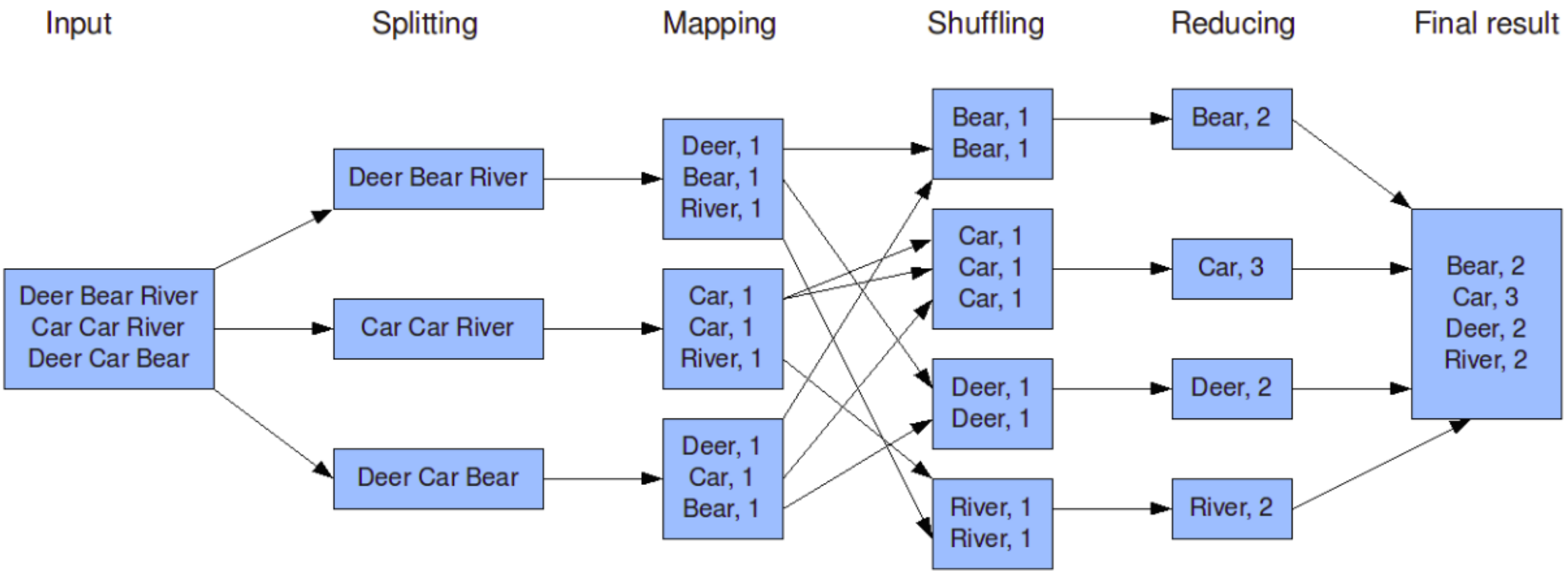
{ fase map }

{ fase reduce}



Fonte: https://blog.trifork.com/2009/08/04/introduction-to-hadoop/

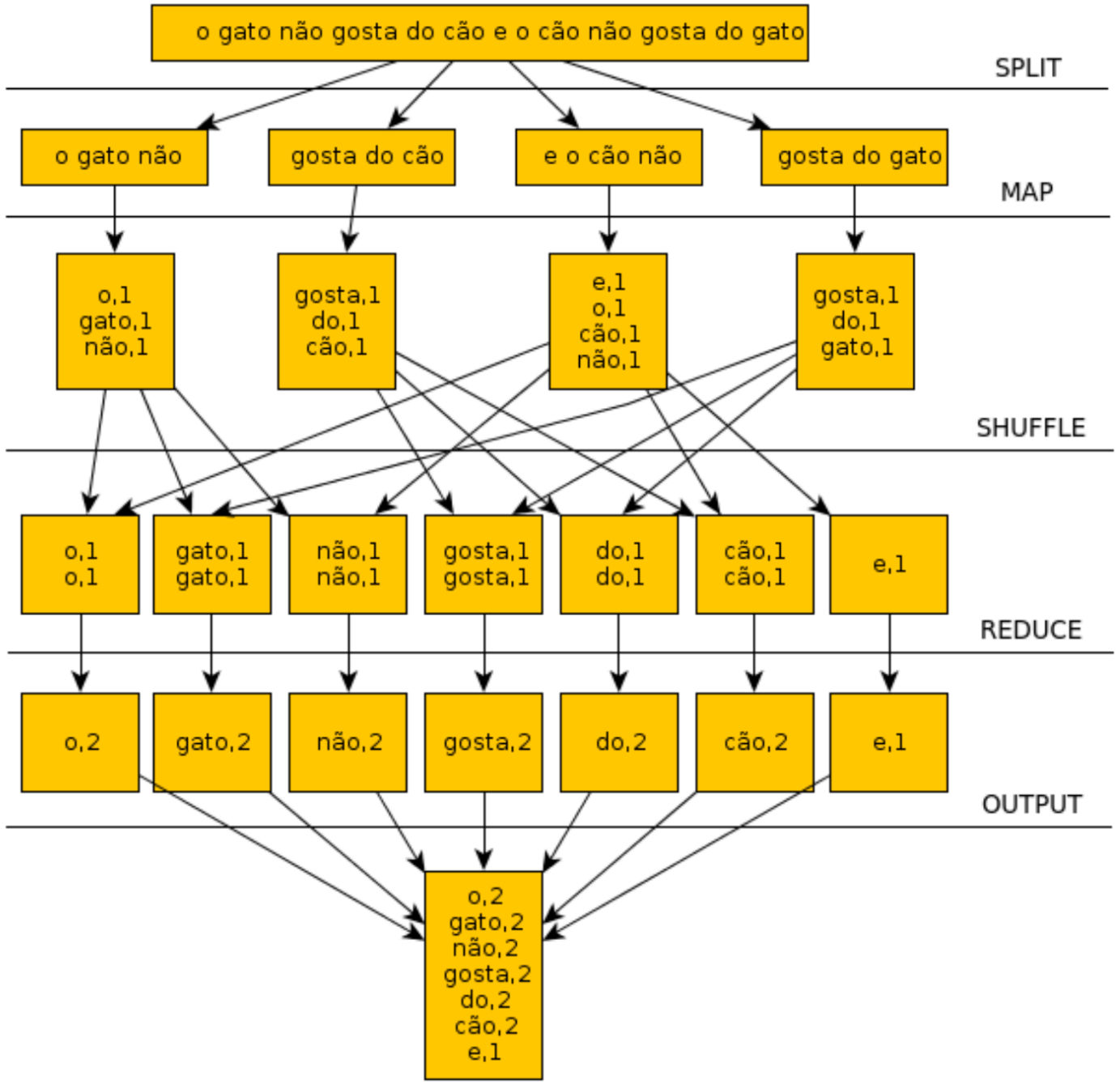
{ hello world mapreduce }



Fonte: https://blog.trifork.com/2009/08/04/introduction-to-hadoop/

{ hello world mapreduce (2) }

Fonte: Picoli, I. L., de Almeida, L. B., de Almeida, E. C. Otimização de Desempenho em Processamento de Consultas MapReduce.



{ wc_map.py }

```
1  #!/usr/bin/env python
2  import sys
3
4  # toma linhas da entrada padrao
5  for line in sys.stdin:
6      # remove espacos em branco
7      line = line.strip()
8      # divide linha em palavras
9      words = line.split()
10
11     for word in words:
12         # escreve resultado na saida padrao
13         # que sera entrada para a fase reduce
14         # delimitador: tab
15         print '%s\t%s' % (word, 1)
```

{ wc_reducer.py }

```python
#!/usr/bin/env python
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove espaços sobrando
    line = line.strip()
    # captura a saida de wc_map_vx.py
    word, count = line.split('\t', 1)

    try:
        count = int(count)
    except ValueError:
        # por simplicidade
        # ignorar/descartar erro
        continue

    # hadoop ordena saida map por chave
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # escreve na saida final
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# ultima iteracao do for
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

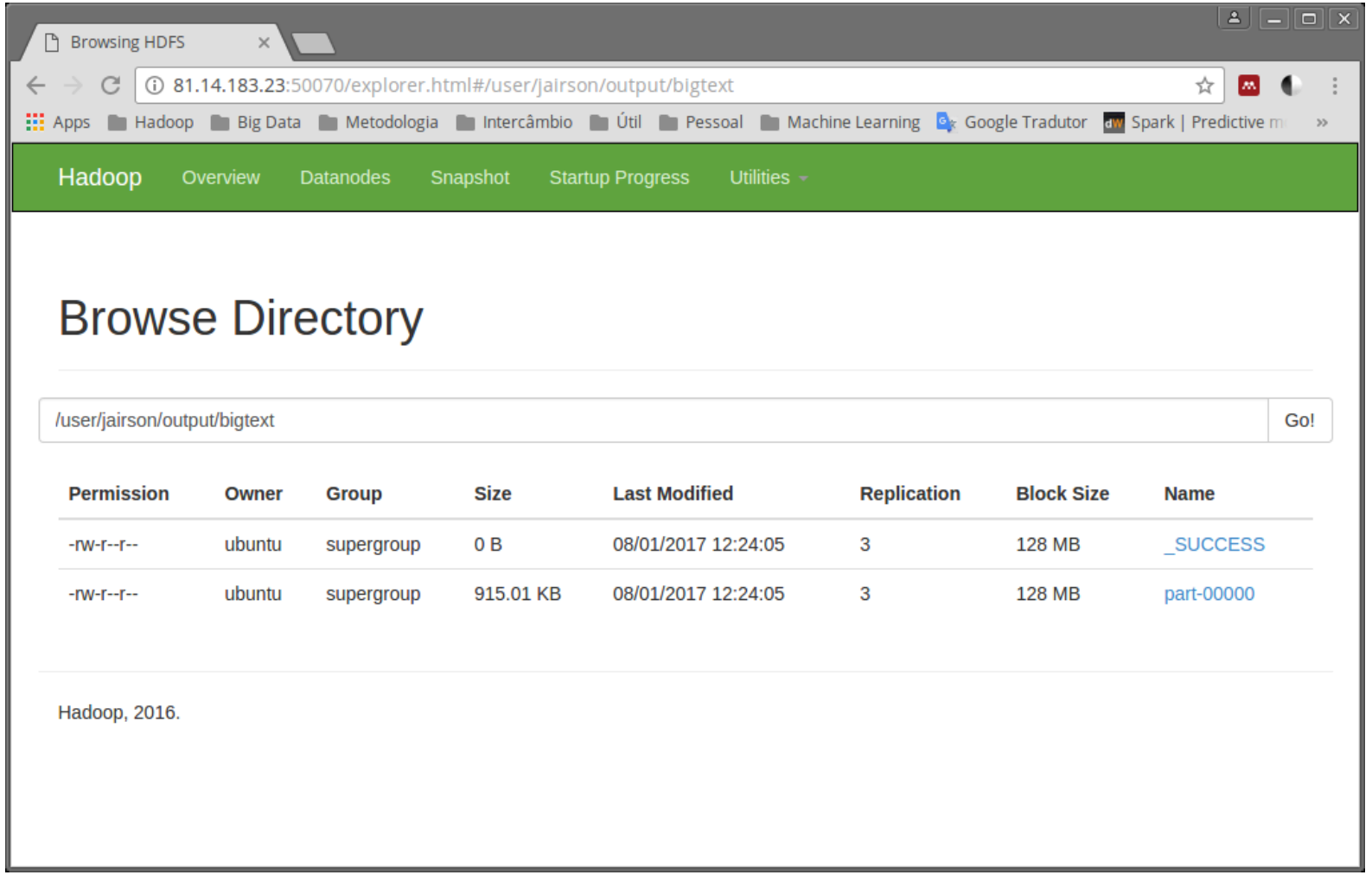
{ executando no cluster - entrada 1 }

cluster

```
ubuntu@name-node:~/python/wordcount$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar -file mapper.py -mapper
mapper.py -file reducer.py -reducer reducer.py -input /user/jairson/input/bigtext/* -output /user/jairson/output/bigtext/
17/01/08 15:23:44 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [mapper.py, reducer.py, /tmp/hadoop-unjar2360298793577516210/] [] /tmp/streamjob1335253702011949279.jar tmpDir=null
17/01/08 15:23:45 INFO client.RMProxy: Connecting to ResourceManager at /192.168.4.221:8032
17/01/08 15:23:45 INFO client.RMProxy: Connecting to ResourceManager at /192.168.4.221:8032
17/01/08 15:23:46 INFO mapred.FileInputFormat: Total input paths to process : 1
17/01/08 15:23:46 INFO mapreduce.JobSubmitter: number of splits:2
17/01/08 15:23:46 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1482799310881_0017
17/01/08 15:23:47 INFO impl.YarnClientImpl: Submitted application application_1482799310881_0017
17/01/08 15:23:47 INFO mapreduce.Job: The url to track the job: http://name-node:8088/proxy/application_1482799310881_0017/
17/01/08 15:23:47 INFO mapreduce.Job: Running job: job_1482799310881_0017
17/01/08 15:23:52 INFO mapreduce.Job: Job job_1482799310881_0017 running in uber mode : false
17/01/08 15:23:52 INFO mapreduce.Job:  map 0% reduce 0%
17/01/08 15:23:58 INFO mapreduce.Job:  map 50% reduce 0%
17/01/08 15:23:59 INFO mapreduce.Job:  map 100% reduce 0%
17/01/08 15:24:06 INFO mapreduce.Job:  map 100% reduce 100%
17/01/08 15:24:07 INFO mapreduce.Job: Job job_1482799310881_0017 completed successfully
17/01/08 15:24:07 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=10802167
                FILE: Number of bytes written=21970717
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=6492986
                HDFS: Number of bytes written=936970
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Killed map tasks=1
                Launched map tasks=2
                Launched reduce tasks=1
                Rack-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=8963
                Total time spent by all reduces in occupied slots (ms)=5121
                Total time spent by all map tasks (ms)=8963
                Total time spent by all reduce tasks (ms)=5121
                Total vcore-milliseconds taken by all map tasks=8963
                Total vcore-milliseconds taken by all reduce tasks=5121
                Total megabyte-milliseconds taken by all map tasks=9178112
                Total megabyte-milliseconds taken by all reduce tasks=5243904
```

{ saída - entrada 1 }

```
"Why,    42
"Why,"   4
"Why... 1
"Why?    7
"Why?"   5
"Will    22
"Wine?   1
"Wish    1
"With    14
"Without         2
"Without,        1
"Witness         1
"Witness:        8
"Women   1
"Women's         1
"Women, 2
"Women,"         1
"Won't   3
"Wonderful!"     1
"Wonderful!...   1
"Worse   1
"Wostov!         1
"Wostov,         3
"Would   10
--More--
```



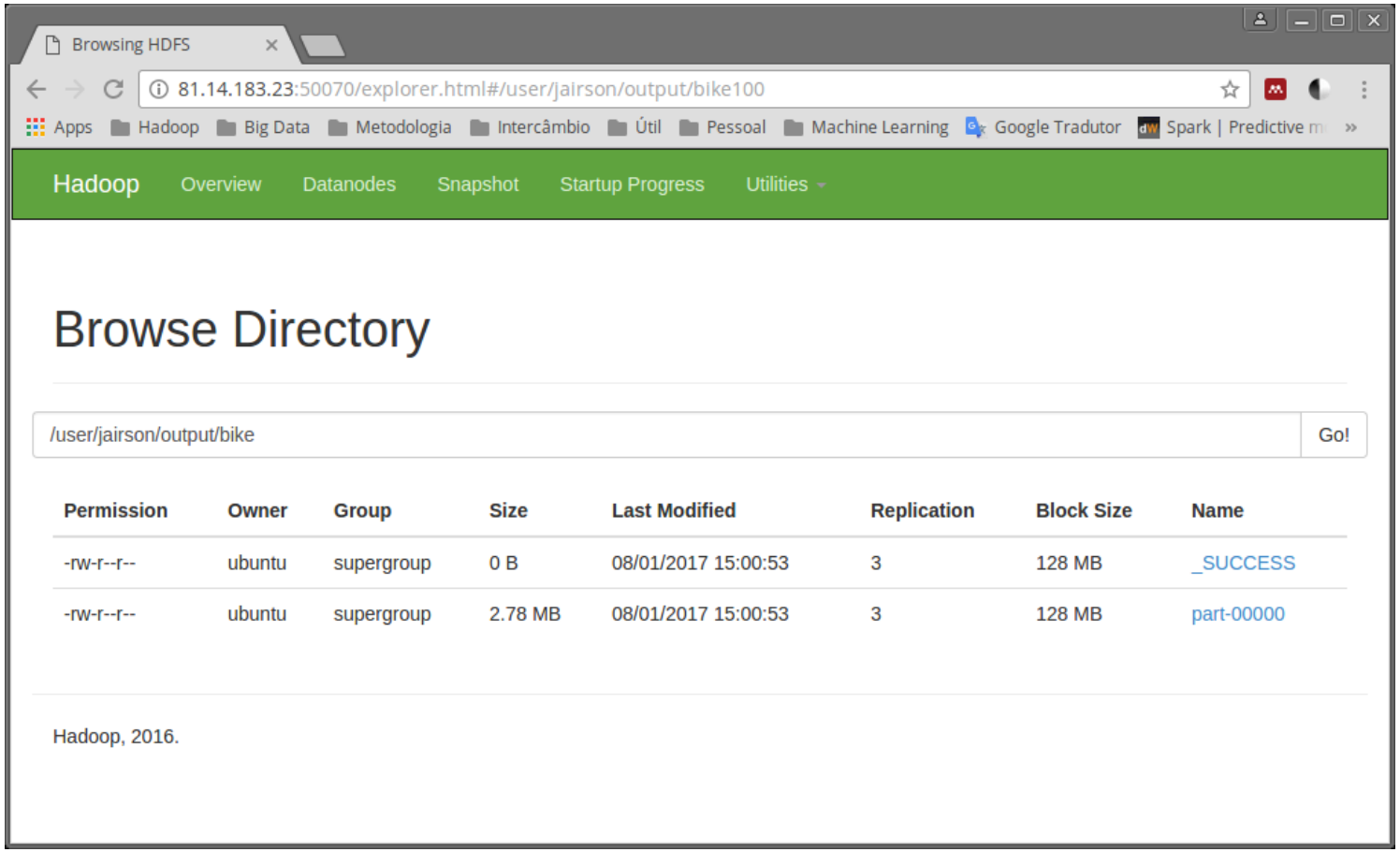
{ executando no cluster - entrada 2 }

cluster

```
17/01/08 17:52:01 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [wordcount_mapper.py, wordcount_reducer.py, /tmp/hadoop-unjar7337301466230706456/] [] /tmp/streamjob2472913245112318658.jar tmpDir=null
17/01/08 17:52:02 INFO client.RMProxy: Connecting to ResourceManager at /192.168.4.221:8032
17/01/08 17:52:02 INFO client.RMProxy: Connecting to ResourceManager at /192.168.4.221:8032
17/01/08 17:52:03 INFO mapred.FileInputFormat: Total input paths to process : 1
17/01/08 17:52:03 INFO mapreduce.JobSubmitter: number of splits:5
17/01/08 17:52:03 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1482799310881_0021
17/01/08 17:52:04 INFO impl.YarnClientImpl: Submitted application application_1482799310881_0021
17/01/08 17:52:04 INFO mapreduce.Job: The url to track the job: http://name-node:8088/proxy/application_1482799310881_0021/
17/01/08 17:52:04 INFO mapreduce.Job: Running job: job_1482799310881_0021
17/01/08 17:52:09 INFO mapreduce.Job: Job job_1482799310881_0021 running in uber mode : false
17/01/08 17:52:09 INFO mapreduce.Job:  map 0% reduce 0%
17/01/08 17:52:20 INFO mapreduce.Job:  map 17% reduce 0%
17/01/08 17:52:22 INFO mapreduce.Job:  map 31% reduce 0%
17/01/08 17:52:23 INFO mapreduce.Job:  map 42% reduce 0%
17/01/08 17:52:25 INFO mapreduce.Job:  map 43% reduce 0%
17/01/08 17:52:26 INFO mapreduce.Job:  map 53% reduce 0%
17/01/08 17:52:28 INFO mapreduce.Job:  map 62% reduce 0%
17/01/08 17:52:29 INFO mapreduce.Job:  map 65% reduce 0%
17/01/08 17:52:31 INFO mapreduce.Job:  map 69% reduce 0%
17/01/08 17:52:32 INFO mapreduce.Job:  map 81% reduce 0%
17/01/08 17:52:34 INFO mapreduce.Job:  map 91% reduce 0%
17/01/08 17:52:35 INFO mapreduce.Job:  map 93% reduce 0%
17/01/08 17:52:37 INFO mapreduce.Job:  map 95% reduce 0%
17/01/08 17:52:40 INFO mapreduce.Job:  map 99% reduce 0%
17/01/08 17:52:41 INFO mapreduce.Job:  map 100% reduce 0%
17/01/08 17:52:45 INFO mapreduce.Job:  map 100% reduce 69%
17/01/08 17:52:48 INFO mapreduce.Job:  map 100% reduce 72%
17/01/08 17:52:51 INFO mapreduce.Job:  map 100% reduce 75%
17/01/08 17:52:54 INFO mapreduce.Job:  map 100% reduce 78%
17/01/08 17:52:57 INFO mapreduce.Job:  map 100% reduce 82%
17/01/08 17:53:00 INFO mapreduce.Job:  map 100% reduce 85%
17/01/08 17:53:03 INFO mapreduce.Job:  map 100% reduce 88%
17/01/08 17:53:06 INFO mapreduce.Job:  map 100% reduce 91%
17/01/08 17:53:09 INFO mapreduce.Job:  map 100% reduce 92%
17/01/08 17:53:12 INFO mapreduce.Job:  map 100% reduce 94%
17/01/08 17:53:15 INFO mapreduce.Job:  map 100% reduce 96%
17/01/08 17:53:18 INFO mapreduce.Job:  map 100% reduce 97%
17/01/08 17:53:21 INFO mapreduce.Job:  map 100% reduce 99%
17/01/08 17:53:23 INFO mapreduce.Job:  map 100% reduce 100%
17/01/08 17:53:23 INFO mapreduce.Job: Job job_1482799310881_0021 completed successfully
17/01/08 17:53:23 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=1482727192
                FILE: Number of bytes written=2224824134
                FILE: Number of read operations=0
```

{ saída - entrada 2 }

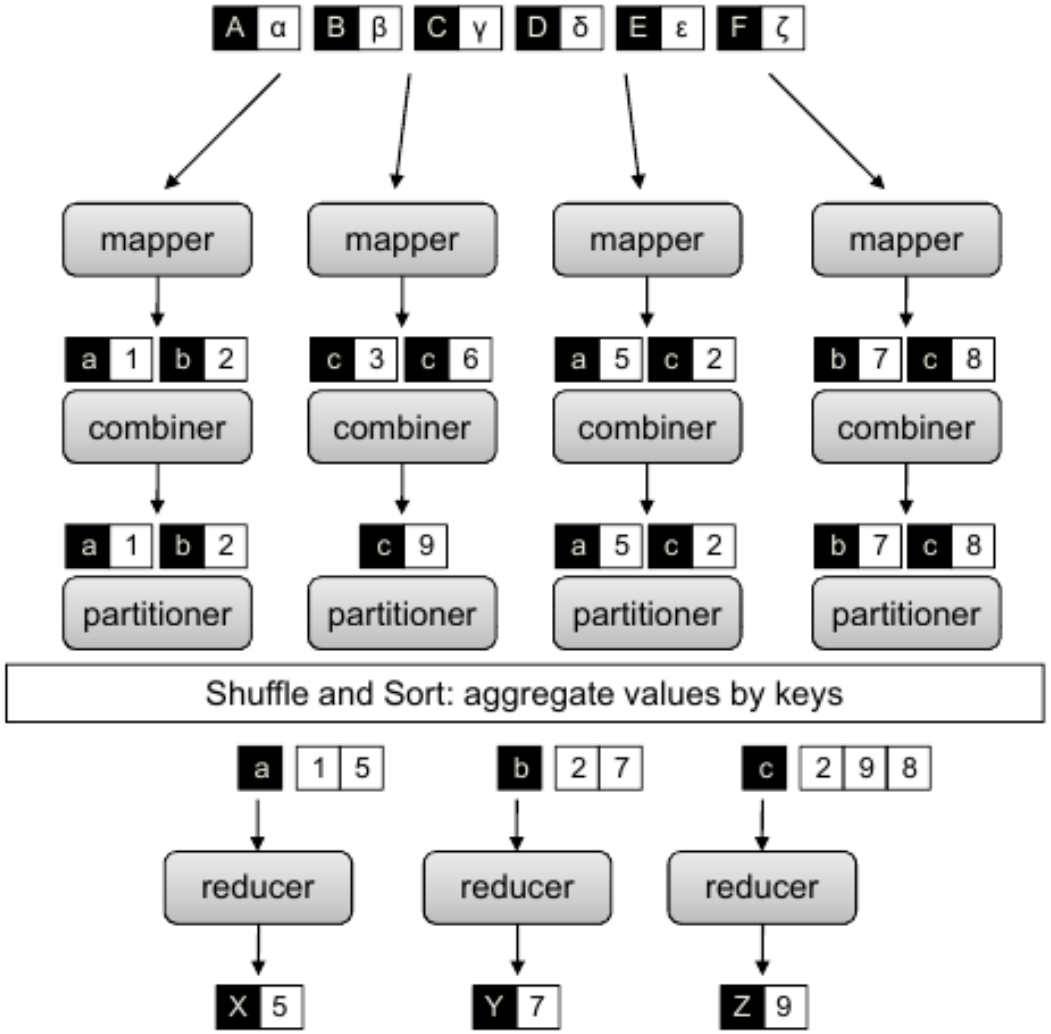
```
"10","10","5","2013/09/13"      113
"10","10","5","2013/09/16"      217
"10","10","5","2013/09/17"      4
"10","10","5","2013/09/18"      104
"10","10","5","2013/09/19"      331
"10","10","5","2013/09/22"      811
"10","10","5","2013/09/23"      529
"10","10","5","2013/09/28"      177
"10","10","5","2013/09/29"      508
"10","10","5","2013/10/01"      92
"10","10","5","2013/10/02"      225
"10","10","5","2013/10/05"      1
"10","10","5","2013/10/06"      314
"10","10","5","2013/10/11"      137
"10","10","5","2013/10/14"      3
"10","10","5","2013/10/15"      11
"10","10","5","2013/10/21"      180
"10","10","5","2013/10/24"      37
"10","10","5","2013/10/26"      1
"10","10","5","2013/10/29"      5
"10","10","5","2013/11/01"      51
"10","10","5","2013/11/04"      24
"10","10","5","2013/11/07"      14
"10","10","5","2013/11/11"      381
--More--
```



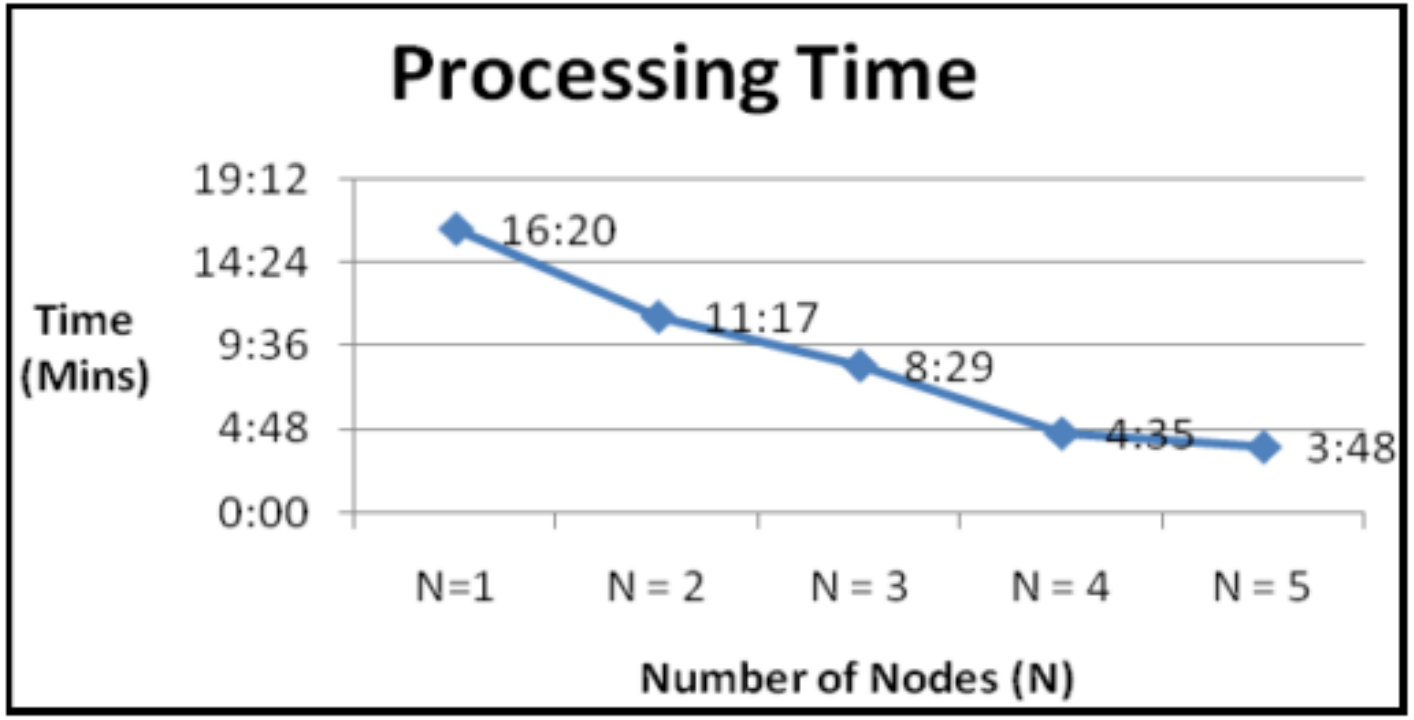
{ combinadores e particionadores }

- Combinadores ou "mini reducers"
  - atuam antes da fase shuffle/sort
  - operam isoladamente em cada mapper
  - agregação local por chave na saída do mapper
  - minimizam tráfego de dados no cluster
- Particionadores
  - Dividem o espaço de chaves intermediárias
  - Determinam o reducer que vai receber o conjunto de chaves
    - Comumente: hash (key) % num_reducers

{ mapreduce - uma visão mais completa [5] }



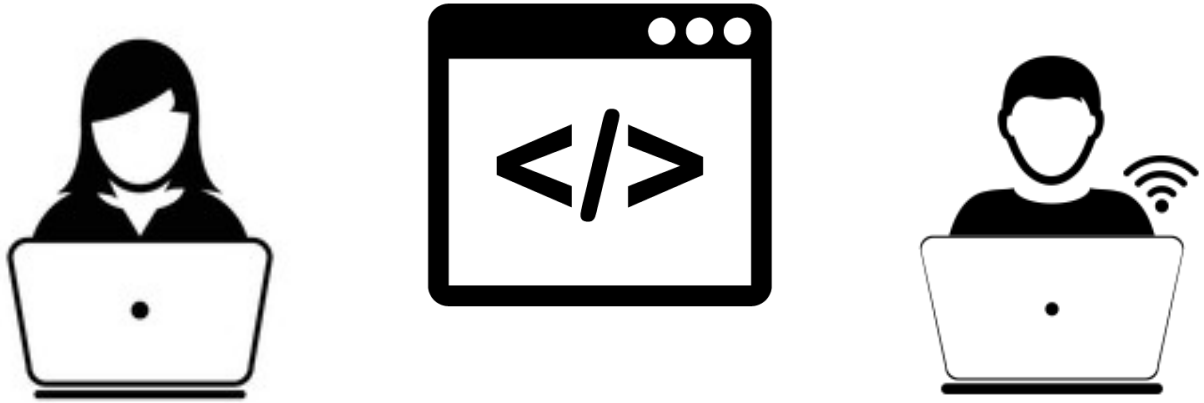
{ tempo de processamento versus número de nós [6] }



Ordenação de um arquivo de texto de 1 Gigabyte, com 16 mappers e um reducer

{ vamos consolidar a teoria }

Prática: MapReduce



{ referências }

[1] Apache Hadoop Project - http://hadoop.apache.org/

[2] Who uses Hadoop? - https://wiki.apache.org/hadoop/PoweredBy

[3] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In ACM SIGOPS operating systems review (Vol. 37, No. 5, pp. 29-43). ACM.

[4] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Commun. ACM, vol. 51, no. 1, p. 107, 2008.

[5] Lin, J., & Dyer, C. (2010). Data-intensive text processing with MapReduce. Synthesis Lectures on Human Language Technologies, 3(1), 1-177.

[6] P. Bedi, V. Jindal, and A. Gautam, “Beginning with big data simplified,” in 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC), 2014, pp. 1–7.