$\boxed{1.1}$

Based on the information in the slide

$1\ ZB = 2^{10} \times 2^{10} \times 2^{10} \times 2^{10}\ GB$

$\therefore 180\ ZB = 180 \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10}\ GB$

8 billion people $= 8 \times 10^9$ people

Data per person $= \dfrac{180 \times 2^{10} \times 2^{10} \times 2^{10} \times 2^{10}}{8 \times 10^9}$

$= 24739.01162$ GB per person.

$1\ ZB = 10^3 \times 10^3 \times 10^3 \times 10^3\ GB$

$= 10^{12}\ GB$

$180\ ZB = 180 \times 10^{12}\ GB$

8 billion people

$= 8 \times 10^9$ people

Data per person $= \dfrac{180 \times 10^{12}}{8 \times 10^9}$

$= 22500$ GB per person

Scanned with CamScanner

$\boxed{1.2}$

For a single Node : (exact calculation)

1 TB = 1024 GB

1 GB = 1024 MB

$\therefore$ 100TB = 100 × 1024 × 1024 MB

= 104,857,600 MB

speed is 100 MB/s

so, time in seconds = $\frac{104,8457,600}{100}$ sec

= 1048576 sec

$= \frac{1048576}{24 \times 60 \times 60}$ days

$\approx$ 12.13 days

can also be done with the information from slide

1 TB = $2^{10} \times 2^{10}$ MB

$\therefore$ 100 TB = $10^2 \times 2^{10} \times 2^{10}$ MB

time in seconds = $\frac{\cancel{10^2} \times 2^{10} \times 2^{10}}{\cancel{100}}$

$= 2^{10} \times 2^{10}$

= 1048576 sec

So the rest of the calculation is basically the same as like on the left side.

if the cluster of 200 nodes with each being 100 MB/s

$\therefore$ speed will be = 100 × 200 MB/s

= 20000

t in seconds with cluster = $\frac{104857600}{20000}$ sec

= 5242.88 sec

$\approx \frac{5242.88}{60}$ = 87.38 sec.

Scanned with CamScanner

$\boxed{1.3}$

**1.2** →

(a) False (b) False (c) True (d) True

**1.6**

(a) False (b) True (c) True (d) True

**1.7**

The options are given below :-

(a) option 1 : 1 computer, 8 GB of RAM, 4 cores, 1 TB drive

(b) option 2 : 2 computers, 4 GB of RAM, 2 cores each, 1 TB drive each

here option 1 is scale up and option 2 is scale out, Depending of the work or goal or analyses we chose either option 1 (scale up) or option 2 (scale out). But I would chose option 2 which is the scale out as the advantages fair outweighs the traditional scale up approach.

The advantages and disadvantages are given below-

Scanned with CamScanner

Advantages of scaleup (option 1) and scale out (option 2)

**scale up (option 1)**

1. less energy consumption
2. less expense in cooling system
3. easier to implement solutions

**scale out (option 2)**

1. cheaper.
2. fault tolerance possible.
3. easier to grow.

**Disadvantages**

**scale up (option 1)**

1. Price
2. No fault tolerance
3. limited hardware upgrades

**scale out (option 2)**

1. More physical space
2. Energy costs (electricity and cooling)
3. Network equipment required.

Scanned with CamScanner

1.9

The key difference between a tradition HPC cluster and a Big data cluster is given below –

| HPC | Big Data |
| --- | --- |
| ① Focus on computation - intensive jobs | ① Focus on data intensive jobs |
| ② surprised by hardware failures | ② Hardware failure common. |
| ③ code : simulation, optimization | ③ code : Data science, graphs. |
| ④ Mix of ~~CPU~~ CPU and GPU | ④ usually mix of CPU and GPU and data |
| ⑤ Data moved to where it will be processed. | ⑤ Job moved to where the data is located. |
| ⑥ SIMD / MIMD model more general parallelism | ⑥ SIMD model data parallelism. |
| ⑦ Needs specialized hardware | ⑦ commodity hardware acceptable. |

Scanned with CamScanner

1.11

Data locality is what distinguishes big data clusters from HPC clusters. The idea of data locality is to use the local drives of the computing nodes as distributed storage so that during execution of a ~~to~~ Big data solution, we should ensure that each computing nodes uses the data stored in it. This ~~s~~ is because we ~~can~~ want less data transfer through the communication network as little as possible and also keeps the idea of ~~k~~ moving the job into where the data is located for big data clusters.

1.12

This is Because in Big data the emphasis is on scaling out rather than scaling up so we dont always require processors with the highest ~~spes~~ speed. This implies that we can spread the data and processing responsibilities among numerous common, commodity hardware nodes rather than depending on a single, sophisticated machine. Because these systems operate in parallel, we can manage big datasets without the requirement for expensive or powerful devices. This makes the method more scalable, cost efficient and effective.

Scanned with CamScanner

# Q. 1.4

## 2.10

map (k , v) returns (v * k , k)

key value pair : [(0,15) ,(15 , 0) ,(2 , 0)] = data

code :

```
data = [(0,15) ,(15,0) ,(2,0)]
result = list(map(lambda pair : (pair[1] * pair[0], pair[0]),
                                                  data))
print (result)
```

output :

[(0, 0) , (0, 15) , (0, 2)]

## 2.11

data is the output from the previous step. we are to ~~data~~ apply ~~schedule~~ shuffle →

map

(0,0)

(0,15)

(0,2)

shuffle

(0, [0,15,2])

output : { 0: [0, 15, 2] }

code:

```
shuffle = { }
for k,v in result:
    if k in shuffle:
        shuffle[k].append(v)
    else:
        shuffle[k] = [v]
Print (shuffle)
```

Scanned with CamScanner

$\boxed{2 \cdot 12}$

<u>code</u>:

```
# data from previous shuffle :)
shuffle = { 0 : [0, 15, 2] }
reduced = {k : reduce (lambda x, y : x*y , v) for k, v in
                                        shuffle.items()}
print (reduced)
```

<u>output</u>:

```
{ 0 : 0 }
```

$\boxed{2 \cdot 13}$

If we use subtraction instead of product the code looks like this →

```
shuffle = { 0 : [0, 15, 2] }
reduced = {k : reduce (lambda x, y : x-y , v) for k, v
                    in shuffle.items() }
print (reduced)
```

<u>output</u>:

```
{ 0 : -17 }
```

Scanned with CamScanner

As this is a subtraction operation we can have some potential issues as below –

① subtraction is Non commutative in nature because the order in which the elements are subtracted affects the result.

i.e. $a - b \neq b - a$

$a = 5$
$b = 10$
$a - b = 5 - 10$
$= -5$
$b - a = 10 - 5$
$= 5$
$5 \neq 5$

② subtraction is non-associative in nature as grouping of operands affects the result.

$(a - b) - c \neq a - (b - c)$

$a = 10$
$b = 5$
$c = 2$
$(a - b) - c \neq a - (b - c)$
$(10 - 5) - 2 \neq 10 - (5 - 2)$
$5 - 2 \neq 10 - 3$
$3 \neq 7$

So keeping these two in mind subtraction leads to negative results which might not be desired in some work as the output can be difficult to interpret depending on the list of values.

Scanned with CamScanner

2.14

```
def special (x, y):
    if x > y:
        return x
    else:
        return y.
```

For the function to be ~~associative~~ commutative we have to prove that

special (x, y) = special (y, x)

let, $x = 5$ for special (5, 2) it ~~set~~ return 5

$y = 2$. for special (2, 5) it return 5

so it can be proved that special (x, y) is commutative in nature.

On the other hand, to prove the associative nature we need to prove,

special (special (x, y), z) = special ( x, special (y, z))

let, $x = 7$ special (x, y) ~~return~~ which is special (7, 3)

$y = 3$ returns 7. Then we do special (7, z) where

$z = 5$ z = 5 also return 7.

so, special (special (7, 3), 5) = 7.

Scanned with CamScanner

on the Right hand side we see that,

special ( x, $\underbrace{\text{special } (y, z)}$ )

$\hookrightarrow$ special (3, 5) returns 5

then special ( ~~5~~ 7, 5) returns 7.

$\therefore$ ~~LHS~~ = so, 7 = 7

this proves that special is both commutative and associative in nature.

Scanned with CamScanner

Q. 1.5

**Map(k, string) → list((len(word), 1))**

Input: Key and value

output: list of key values

**shuffle (Network)**

output: key, list of values

**Reduce**

x, y → (x+y)

(1, welcome to the big) → (7, 1), (2, 1), (3, 1), (3, 1)

(2, World of Big Big) → (5, 1), (2, 1), (3, 1), (3, 1)

(3, Data Welcome World bye) → (4, 1), (7, 1), (5, 1), (3, 1)

(7, [1, 1]) → (7, 2)

(2, [1, 1]) → (2, 2)

(3, [1, 1, 1, 1, 1]) → (3, 5)

(5, [1, 1]) → (5, 2)

(4, [1]) → (4, 1)

$$\frac{\sum \text{key} * \text{value}}{\sum \text{value}}$$

$$= \frac{(7\times2)+(2\times2)+(3\times5)+(5\times2)+(4\times1)}{2+2+5+2+1}$$

$$= \frac{47}{12}$$

$$= 3.9167$$

$$\approx 3.92$$

if rounded up

Scanned with CamScanner

Map (post office no. with address, list of houses with address) → list(post office no., (house no., d))

shuffle (post office no. (list of house no., distance))

reduce

(~~key~~ office number, min(h, distance)[o

$(o1, (10,12)) : (h1, (1,2))$ ; → $(o_1, (h1, 13.45))$

$(h2, (6, 18))$ ; → $(o_1, (h2, 7.21))$

$(h3, (20, 2))$ → $(o_1, (h3, 14.14))$

→ $(o_1, [(h1, 13.45), (h2, 7.21), (h3, 14.14)])$ → $(o1, h2)$

$(o2, (-3,4)) : (h4, (-6, 10))$ ; → $(o_2, (h4, 6.70))$

$(h1, (1,2))$; → $(o_2, (h1, 4.47))$

$(h5, (-2,-2))$; → $(o_2, (h5, 6.08))$

$(h2, (6,18))$ → $(o_2, (h2, 16.64))$

→ $(o_2, [(h4, 6.70), (h1, 4.47), (h5, 6.08), (h2, 16.64)])$ → $(o_2, h1)$

$(o3, (1, -2)) : (h1, (1,2))$ ; → $(o_3, (h1, 4))$

$(h5, (-2,-2))$ → $(o_3, (h5, 3.6))$

→ $o_3, [(h1, 4), (h5, 3)]$ → $(o_3, h5)$

1.6 (a) * $d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$

Scanned with CamScanner

Map(post office no, list of houses with address) → list(house no, (post office no., d))

shuffle(house no., list of office, distance)

reduce

house no, min(0, distance) [o

(o1, (10,12)): (h1, (1,2));
(h2, (6,18));
(h3, (20,2));

(o2, (-3,4)): (h4, (-6,10));
(h1, (1,2));
(h5, (-2,-2));
(h2, (6,18));

(o3, (1,-2)): (h1, (1,2));
(h5, (-2,-2))

→ (h1, (o1, 13.45))
→ (h2, (o1, 7.21))
→ (h3, (o1, 14.14))
→ (h4, (o2, 6.70))
→ (h1, (o2, 4.47))
→ (h5, (o2, 6.08))
→ (h2, (o2, 16.64))
→ (h1, (o3, 4))
→ (h5, (o3, 3))

(h1, [(o1, 13.45), (o2, 4.47) (o3, 4)]) → (h1, o3)

(h2, [(o1, 7.21), (o2, 16.64),]) → (h2, o1)

(h3, [(o1, 14.14)]) → (h3, o1)

h4, [(o2, 6.70)] → (h4, o2)

h5, [(o2, 6.08), (o3, 3)] → (h5, o3)



Scanned with CamScanner

Map

(Person , friend) → sorted (person A, person B), remaining friends of person A)

Input

Eli : Lila, Ada

Lila : Eli, Ada, Eva

Ada : Eli, Lila, Eva

Eva : Lila, Ada

output

((Eli, Lila), Ada)

((Ada, Eli), Lila)

((Eli, Eva), (Lila, Ada))

((Eli, Lila), (Ada, Eva))

((Ada, Lila), (Eli, Eva))

((Eva, Lila), (Eli, Ada))

((Ada, Eli), (Lila, Eva))

((Ada, Lila), (Eli, Eva))

((Ada, Eva), (Eli, Lila))

((Eli, Eva), (Lila, Ada))

((Ada, Eva), (Lila))

((Eva, Lila), (Ada))

Shuffle

(Eli, Lila), [(Ada), (Ada, Eva)]

(Ada, Eli), [(Lila), (Lila, Eva)]

(Eli, Eva), [(Lila, Ada), (Lila, Ada)]

(Ada, Lila), [(Eli, Eva), (Eli, Eva)]

(Eva, Lila), [(Eli, Ada), (Ada)]

(Ada, Eva), [(Eli, Lila), (Lila)]

Reduce

x, y → Intersection(x, y)

(Eli, Lila) : Ada

(Ada, Eli) : Lila

(Eli, Eva) : Lila, Ada

(Ada, Lila) : Eli, Eva

(Eva, Lila) : Ada

(Ada, Eva) : Lila.



Scanned with CamScanner