Student ID (*Matricola*)

0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2
3	3	3	3	3	3
4	4	4	4	4	4
5	5	5	5	5	5
6	6	6	6	6	6
7	7	7	7	7	7
8	8	8	8	8	8
9	9	9	9	9	9

Computing Infrastructures

Course 095897

P. Cremonesi, M. Gribaudo

08-09-2015

Last Name / Cognome:

.....

First Name / Nome:

.....

Answers must be given exclusively on the answer sheet (last sheet): DO NOT FILL ANY BOX IN THIS SHEET

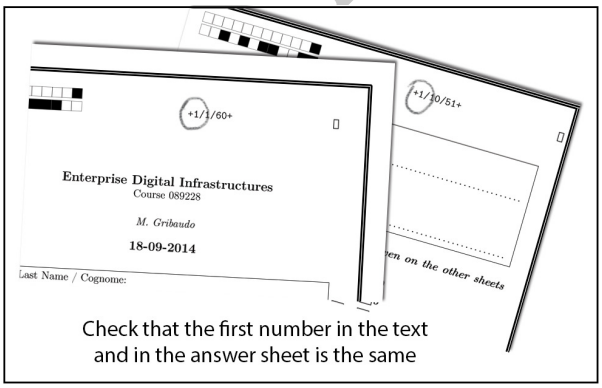
Students must use pen (black or blue) to mark answers (no pencil).
Students are permitted to use a non-programmable calculator.

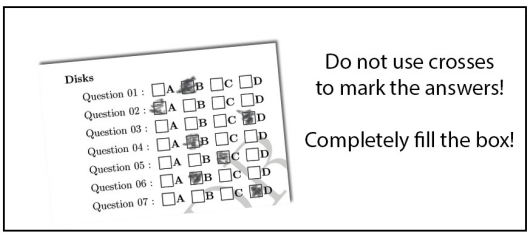
Students are NOT permitted to copy anyone else's answers, pass notes amongst themselves, or engage in other forms of misconduct at any time during the exam.

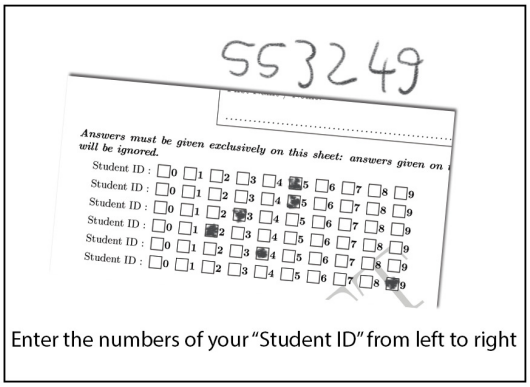
Students are NOT permitted to use mobile phones and similar connected devices.

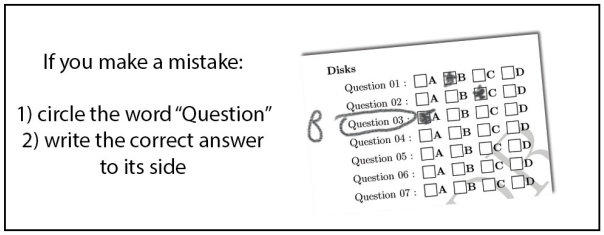
Scores: correct answers +1 point, unanswered questions 0 points, wrong answers -0.333 points.

Reserve questions **must NOT** be answered, unless explicitly stated during the exam.









Disks

A large company wants to set up a RAID 6+0, with 10 groups of 6 disks each. All the disks are identical, and characterized by a capacity of 1 TB, an $MTTF = 1500$ days and an $MTTR = 20$ days.

Question 1 Which is the total capacity of the array?

A 58 TB. B 40 TB. C 59 TB. D 50 TB.

SOLUTION:
Each group uses RAID 6, so of the 6 disks, only four contains useful data: each group has thus a capacity of 4 TB. The total capacity of the RAID 6+0 array is thus $4 \cdot 10 = 40$ TB.

Question 2 Which is the mean time to data loss of the system?

A $14062,5$ days
B $82,189$ days
C $3906,25$ days
D 84375 days

SOLUTION:
$MTTDL = \frac{2 \cdot 1500^3}{60 \cdot 5 \cdot 4 \cdot 20^2} = 14062,5$ days.

Question 3 Which could be the main cause of an $MTTR = 20$ days?

A Time to identify the fault.
B Time required to reconstruct 1 TB of data from the surviving disks
C Sum of the times to identify, provision and replace the broken disk
D Time to find a disk identical to the one already installed

SOLUTION:
Time required to reconstruct 1 TB of data from the surviving disks: since 5 disks must be accessed in parallel with the normal operation of the system

Each of disks in the previous questions has a rotational speed of 10800 RPM, a transfer rate of 128 MB / sec, an average seek time of 5 ms. Data is clustered in blocks of 4 KB. The controller overhead is negligible.

Question 4 Which is the rotational latency of the disks?

A 5.556 ms B 4.167 ms C 8.333 ms D 2.778 ms

SOLUTION:
$T_{lat} = 1/2 \cdot 1/RPM = 0.5/10800$ min $= 30000/10800$ ms $= 2.778$ ms.

Question 5 Which is the average transfer time required to read the data in one cluster?

A 0.2441 ms
B 0.0305 ms
C 3.9063 ms
D 31.25 ms

SOLUTION:
$4/(128 * 1024) * 1000 = 0.0305\ ms$.

Question 6 Which is the average service time required to read one cluster (without locality)?

A 7.8083 ms
B 41.8056 ms
C 9.4108 ms
D 17.2396 ms

SOLUTION:

The sum of the solution to the two previous questions, plus the average seek time: 7.8083 ms.

Question 7 **Reserve** - *Do not answer unless explicitly stated during the exam*
If there is one failure in the RAID 6+0 configuration described above, how many disks are needed to repair it?

A 58 B 4 C 59 D 5

SOLUTION:
4 since there is just one broken disk, only the other disks in the same group needs to be used. The data can be constructed in the following ways:

- if it is a parity disk (either P or Q), it can be reconstructed from the surviving 4 data disks;
- if it is a data disk, sincere there is just one failure, it can be constructed as in RAID 5, using the three data disks that have survived and parity P.;

Performance evaluation

A small company has an intranet composed by three PCs. The first PC hosts the web and the application servers and it is characterized by an average service time $S_{serv} = 10$ ms, and a throughput of $X_{serv} = 20$ req/s. The second PC hosts the DB: its demand is $D_{DB} = 35$ ms, and its utilization is $U_{DB} = 80\%$. The third PC hosts the storage server. Every job performs an average of 4 requests to the storage, which is characterized by a demand of $D_{stor} = 24$ ms. The throughputs and the utilizations are measured when $N = 200$ users, and the response time with that workload is $R = 250$ ms.

Question 8 : The throughput of the system is:

- [A] $X = 80$ job/s
- [B] $X = 20$ job/s
- [C] Cannot be computed with available data
- [D] $X = 22.857$ job/s

SOLUTION:
$U = X \cdot D$, $X = U_{DB}/D_{DB} = 0.8/35$ job/ms $= 0.8/35 \cdot 1000 = 22.857$ job / s.

Question 9 The visits to the web and application server v_{serv} are:

- [A] $v_{serv} = 1$.
- [B] $v_{serv} = 3.5$.
- [C] $v_{serv} = 0.875$
- [D] Cannot be computed with available data

SOLUTION:
$v_{serv} = X_{serv}/X = 20/22.857 = 0.875$

Question 10 The visits to the DB v_{DB} are:

- [A] Cannot be computed with available data
- [B] $v_{DB} = 4$
- [C] $v_{DB} = 3.5$
- [D] $v_{DB} = 1$

SOLUTION:
There is not enough information to determine the throughput or the service time of the DB. Visits cannot be computed

Question 11 The average service time of the storage S_{stor} is:

- [A] $S_{stor} = 24$ ms
- [B] $S_{stor} = 6$ ms
- [C] $S_{stor} = 2.7429$ ms
- [D] Cannot be computed with available data

SOLUTION:
From the description, we can understand that $v_{stor} = 4$. Then $S_{stor} = D_{stor}/v_{stor} = 24/4 = 6$ ms.

Question 12 The think time of the users Z is:

- [A] $Z = 8.5$ ms
- [B] Cannot be computed with available data
- [C] $Z = 9$ ms
- [D] Some of the data given must be wrong because if we apply the correct formula we obtain a negative think time Z

SOLUTION:
$R = N/X - Z$, $Z = N/X - R = 200/22.2857 - 250/1000 = 8.5$ s.

Dependability

In the following questions we will assume that both failure and repair events follow exponential distributions.

Question 13

A server with two processors and a single hard disk is considered. The two processors are in parallel together followed in series by the hard disk. Both processors have the following characteristics: $MTTF_{Pr} = 400\ days$, $MTTR_{Pr} = 10\ days$. The parameters of the hard disk are: $MTTF_{HD} = 500\ days$, $MTTR_{HD} = 2\ days$. The reliability of the system at $t = 100\ days$ is equal to:

A 0.7786 B 0.4165 C 0.95134 D 0.5813

SOLUTION:
$R_{Pr}(100) = e^{-100/400} = 0.7788$ $R_{HD}(100) = e^{-100/500} = 0.8187$ $R_{sys} = (1-(1-R_{Pr})^2) * R_{HD} = 0.7786$

Question 14 In the previous case, which is the minimum number of processors in parallel required to achieve a reliability at $t = 100$ greater than 0.81 ?

A 10 B It is not possible C 2 D 4

SOLUTION:
With 4: $(1-(1-0.7788)^4) * 0.8187 = 0.8167 > 0.81$

Question 15 The MTTF computed without repair of the two parallel processors described in Question 13 is:

A 600 B 142.857 C 350 D 99

SOLUTION:
$MTTF_{||} = (1+1/2)\ MTTF_{Pr} = 600$

Question 16 The MTTF computed without repair of the whole system described in Question 13 is:

A 99 B 142.857 C 272.7272 D 350

SOLUTION:
$MTTF_{||} = (1+1/2)\ MTTF_{Pr} = 600$
$MTTF_{Sys} = 1/(1/MTTF_{||} + 1/MTTF_{HD}) = 1/(1/600 + 1/500) = 272.7272$

Question 17 The availability of the whole system is equal to:

A 1 B 0.9954 C 0.625 D 0.9

SOLUTION:
$A_{Pr} = 400/(400+10) = 0.9756$ $A_{HD} = 500/502 = 0.996$
$A_{||} = 1-(1-A_{Pr})^2 = 0.9994$ $A_{sys} = A_{||}A_{HD} = 0.9954$

Question 18 **Reserve** - *Do not answer unless explicitly stated during the exam*

Consider a generic component D with $MTTF_D = 100$. Compute the minimum integer value of t such that the failure probability of the component is greater than 0.6.

A 77 B 92 C 3 D 10

SOLUTION:
$1 - e^{-t/100} \geq 0.6$ $0.4 \geq e^{-t/100}$ $ln0.4 \geq -t/100$ $t \geq -100ln(0.4)$ $t \geq 92$

Cloud Computing

Question 19 Which of the following is **not** an attribute of cloud computing?

A Scalability: cloud services can easily scale based on the user's needs
B Internet access: cloud services require users to have an Internet connection
C Virtualization: cloud services require users to run virtual machines
D Pay-per-usage: the cost of a cloud service increases with its utilization from the user

Question 20 Which of the following is the key benefit for small-medium enterprises when using Software-as-a-Service?

A Usage of virtualization
B Reduced long term IT costs
C Easy customization of the service and easy integration with other applications
D Reduced software and hardware investments

ITIL and DevOps

Question 21 Which of the following is a direct benefit deriving from a correct implementation of the Incident Management process?

A Increased mean-time-to-failure
B Reduced time to restore services
C Reduced number of incidents
D Increased reliability

Solution. Incident Management **reduces the time to solve incidents**, not their number, with the effect of **reducing mttf** and **increasing availability** (not reliability).

Question 22 Which of the following is one of the goals of the Change Management process?

A Evaluate risks and benefits of fixing bugs
B Upgrade applications
C Buy new hardware for the data-center
D Keep track of the configuration of the data-center

Solution. Evaluate risks and benefits of fixing bugs

Question 23 Which is the main goal of the DevOps philosophy?

A Perform periodic and planned releases of changes
B Use either Docker or Vagrant as provisioning tools
C Use light-weight virtual machines
D Reduce the time between a change request and its delivery to production

Solution. Light-weight virtual machines such as Docker and provisioning tools such as Vagrant are only some of the tools that can be used to support the DevOps approach.

Question 24 **Reserve** - *Do not answer unless explicitly stated during the exam*
Which is the main goal of the Problem Management process?

- [A] Reduce the time required to restore a service
- [B] Search for bugs in programs
- [C] Fix problems so that they will not happen again in the future
- [D] Find the cause of recurrent incidents and suggest fixes

Solution. Problem Management does not fix problems, but find the casue of incidents (the fixing of problems is performed by the release management. Problem Managemnt reduce the number of incidents, not the time required to solve them. Software bugs are only one of the possible problems investigatd by Problem Management.

Virtualization

Question 25 Which of the following is a definition of *binary-code translation*?

- [A] Privileged instructions of the guest operating system are modified by the hypervisor when loaded into the main memory
- [B] The source code of the guest operating system is recompiled in order to modify privileged instructions that otherwise could have accessed the hardware directly
- [C] Ring -1 of the processor modifies privileged instructions without adding any execution overhead
- [D] The source code of the operating system is modified in order to remove all privileged instructions

Question 26 Which of the following statements is correct when talking about *encapsulation* of virtual machines?

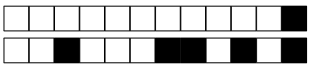
- [A] It is possible to save the state of a running virtual machine into a single file
- [B] It is possible to run a virtual machine inside a virtual machine
- [C] It is possible to guarantee to a virtual machine a minimum set of resources, regardless of the loads of other virtual machines running on the same host
- [D] It is possible to run different guest operating systems on the same host

Question 27 Which of the following statements is correct when talking about *hosted* hypervisor?

- [A] The virtual machine is hosted by a cloud provider
- [B] The processor provides support for hardware-assisted virtualization
- [C] The hypervisor is part of the host operating system, which is able to partition itself into virtual machines
- [D] The hypervisor runs as a user program inside the host operating system

Question 28 Which is the definition of **hardware-assisted** virtualization?

- [A] Each virtual machine runs on a dedicate core
- [B] All guest machines run the same operating system
- [C] The processor provides an additional ring to run instructions able to directly access the hardware
- [D] The hypervisor uses binary-code translation to run the guest operating system



Question 29 Which is the main advantage of OS-level virtualization with respect to full virtualization?

A OS-level virtualization allows to save the state of a virtual machine

B OS-level virtualization provides better insulation between virtual machines

C OS-level virtualization allows to run different host operating systems

D OS-level virtualization provides light-weight virtual machines

Big Data

The following Apache Spark code a user-item-context dataset,*toyDataset.txt* which contains the log of item purchasing history of ech user with contextual variables. Timestamps are UNIX timestamps (number of seconds from 1/1/1970). All other varibles (i.e.: user, item and city) are identified by their numeric id. Read carefully the code below and answer the questions. Pay attention, the code may contain some errors!

IMPORTANT NOTES:

Lines starting with three sharps ### (e.g., lines 1, 17 and 24) contain comments that explain what a function or a fragment of code is supposed to compute.

Lines starting with a single sharp # (e.g., lines 12, 30 and 54) report the output of the previous print call.

Use comments to understand what the content of an RDD or the outcome of an instruction should be. The correct answer to <FILL IN> sections is the one that produces the "exact" RDD structure needed to ensure the correctness of the whole program.

```
1  ### IMPORTS AND PARAMETER DEFINITION
2  from collections import namedtuple
3  import random
4
5  datapath = "toyDataset.txt"
6  yearRatio = 0.5
7  tsSplit = 1104537600+(60*60*24*365*yearRatio)
8  trainRatio = 0.7
9  topn = 4
10 dataRawRdd = sc.textFile(datapath)
11 print dataRawRdd.take(3)
12 # [u'489,231,1105260971,66', u'872,317,1112076979,19', u'582,302,1133854937,56']
13
14 TrainRow = namedtuple("train", ["userId", "itemId", "context"])
15 ContextRow = namedtuple("contextFull", ["ts", "city"])
16
17 ### Helper function to parse raw data
18 def parseData(line):
19     lineVect = line.split(",")
20     currentContext = ContextRow(ts=int(lineVect[2]),city=int(lineVect[3]) )
21     return TrainRow(userId=int(lineVect[0]),itemId=int(lineVect[1]),context=currentContext)
22
23 parsedDataRdd = dataRawRdd.map(parseData)
24 parsedDataRddFiltered = parsedDataRdd.filter(lambda x: x.context.ts < tsSplit)
25 xRdd = (parsedDataRddFiltered.map(lambda x: (x,random.random()))
26         .filter(lambda x: x[1] < trainRatio).map(lambda x: x[0]))
27 train = xRdd.cache()
28
29 print train.take(3)
30 # [train(userId=489, itemId=231, context=contextFull(ts=1105260971, city=66)),
31 # train(userId=375, itemId=369, context=contextFull(ts=1109030039, city=59)),
32 # train(userId=576, itemId=693, context=contextFull(ts=1120072075, city=73))]
33
34 ### Already Rated Item calculation
35 ReducedContext = namedtuple("reducedContext", ["time_of_day", "city"])
36
37 ### Helper function for reducing context dimensionality
38 def getReducedContext(context):
39     hour = (context.ts // 3600)%24
40     if hour >= 5 and hour < 12:
41         _time_of_day = "morning"
42     if hour >= 12 and hour < 17:
43         _time_of_day = "afternoon"
44     if hour >= 17 and hour < 21:
```

```
        _time_of_day = "evening"
    if hour >= 21 or hour < 5:
        _time_of_day = "night"
    return ReducedContext(time_of_day=_time_of_day, city=context.city)

alreadyRatedItems = (train
                     .map(lambda tr: (tr.userId,tr.itemId) )
                     .distinct()
                     .map(lambda x: (x[0],[x[1]]))
                     .<FILL IN> )
print alreadyRatedItems.take(3)
# [(0, [320, 772]), (2, [752, 759, 113]), (4, [586, 419, 471])]

### itemsDecreasingPopOrder is a list of items in decreasing popularity order
itemsDecreasingPopOrder = (train
                .map(lambda tr: (tr.itemId,1))
                .reduceByKey(lambda a,b: a+b)
                .<FILL IN>
                .collect()
                )
print itemsDecreasingPopOrder[:3]
# [192, 294, 13]

predictionsForUserAndReducedContext = (train
                               .map(lambda tr: (tr.userId,getReducedContext(tr.context) ) )
                               .distinct()
                               .map(lambda x: (x,itemsDecreasingPopOrder) ) )
print [(i[0],i[1][:5]) for i in predictionsForUserAndReducedContext.take(3)]
# [((523, reducedContext(time_of_day='night', city=68)), [192, 294, 13, 645, 779]),
# ((718, reducedContext(time_of_day='night', city=31)), [192, 294, 13, 645, 779]),
# ((270, reducedContext(time_of_day='night', city=85)), [192, 294, 13, 645, 779])]

RecommendationRow = namedtuple("recommendation", ["userId","reducedContext","recList"])

### Helper function to filter out already rated items from predictions
def filterPredictions(sortedPredictions,ratedItems,topn):
    outList = []
    for itemId in sortedPredictions:
        if itemId not in ratedItems:
            outList.append(itemId)
    return outList[:topn]

predictionsJoined = (predictionsForUserAndReducedContext<FILL IN>join(alreadyRatedItems) )
print predictionsJoined.first()
# (0, ((reducedContext(time_of_day='night', city=29), [192, 294, ..., 705]), [320, 772]))
### ( ... added for shortness )

recommendations = (predictionsJoined.map(lambda x: RecommendationRow(<FILL IN>)) )
print recommendations.take(3)
# [recommendation(userId=304, reducedContext=reducedContext(time_of_day='night', city=63),
# recList=[294, 13, 645, 779]),
# recommendation(userId=995, reducedContext=reducedContext(time_of_day='night', city=86),
# recList=[192, 294, 13, 645]),
# recommendation(userId=639, reducedContext=reducedContext(time_of_day='night', city=20),
# recList=[192, 294, 13, 645])]
```

Question 30 Complete line 54.

A map(lambda x: (x[0],x[1])).groupByKey()

B reduce(lambda a,b: a + b)

C reduceByKey(lambda a,b: a+b)

D groupByKey()

SOLUTION:
reduceByKey(lambda a,b: a+b)

Question 31 What is the effect of lines 25-26?

A Adding a random number to the RDD

B Randomly subsampling the data before with a timestamp before tsSplit

C Removing those items with a timestamp that precedes tsSplit

D Randomly subsampling all the data

SOLUTION:
Randomly subsampling the data before with a timestamp before tsSplit

Question 32 Complete line 63.

A sortByKey(lambda x: -x[1])

B sortByKey(False).map(lambda x: x[0])

C map(lambda x: (x[1],x[0])).sortByKey(False).map(lambda x: x[1])

D takeOrdered(train.map(lambda tr: tr.itemId.distinct().count()),lambda x: -x[1])

SOLUTION:
map(lambda x: (x[1],x[0])).sortByKey(False).map(lambda x: x[1])

Question 33 Complete line 87. Note: the join has to have the user as key.

A .map(lambda x: ((x[0][0],x[0][1]),x[1])).

B .map(lambda x: (x[0][0],x[0][1],x[1])).

C .map(lambda x: (x[0][0],(x[0][1],x[1]))).

D .

SOLUTION:
.map(lambda x: (x[0][0],(x[0][1],x[1])))

Question 34 Which line is variable *parsedDataRdd* (line 23) is (either totally or partially) computed at?

A 23 B 24 C 27 D 29

SOLUTION:
29. take() is an action, while the other lines are transformations.

Question 35 Complete line 92

A userId=x[0], reducedContext=x[1][0], recList=filterPredictions(x[1][1],x[2],topn)

B userId=x[0], reducedContext=x[1][0][0], recList=filterPredictions(x[1][1],x[1][0][1],topn)

C userId=x[0], reducedContext=x[1][0][0], recList=filterPredictions(x[1][0][1],x[1][1],topn)

D userId=x[0][0], reducedContext=x[0][1], recList=filterPredictions(x[1][0],x[1][1],topn)

SOLUTION:
userId=x[0], reducedContext=x[1][0][0], recList=filterPredictions(x[1][0][1],x[1][1],topn)

Question 36 **Reserve** - *Do not answer unless explicitly stated during the exam*
What is the code to print all the "recList" present in "recommendations" as a python list of lists?.

A print recommendations.map(lambda rec: rec.recList).collect()

B print recommendations.collect()

C print recommendations.map(lambda rec: rec.recList)

D print recommendations.flatMap(lambda rec: rec.recList).collect()

SOLUTION:
print recommendations.map(lambda rec: rec.recList).collect()

Disclaimer: this document is intended as a support to the exam. It should not replace the study. No guarantee is given on the correctness of the formulas contained: we assume no responsibility for errors that might occur in the exam due to mistakes in this document. However we did our best to ensure the correctness of the material here included.

Do not take notes on this document.

Table I. PERFORMANCE: VARIABLES

Variable	Definition
T	length of an observation interval
B	Busy time
C	Number of completions
A	Number of arrivals
W	Jobs per service time
N	number of users
U	utilization
Z	average think time of a user
X	system throughput
λ	arrival rate
S	service time
R	system response time
X_k, U_k S_k, R_k	measure for resource k

Table II. PERFORMANCE: RELATIONS

Relations
$\lambda = \frac{A}{T}$
$X = \frac{C}{T}$
$U = \frac{B}{T}$
$S = \frac{B}{C}$
$N = \frac{W}{T}$
$R = \frac{W}{C}$
$X = \lambda$ if stable

Table III. PERFORMANCE: LAWS

Law	Definition
Visits	$V_k = \frac{C_k}{C}$
Demand	$D_k = V_k S_k$
Utilization	$U_k = X_k S_k = X_0 D_k$
Little's	$N_k = X_k R_k$
Response time	$R = \frac{N}{X} - Z$
Forced flow	$X_k = X_0 V_k$
Queue length	$N_k - U_k$
Queue time	$R_k - D_k$

Table IV. PERFORMANCE: BOUNDS

Bounds	Open	Closed
Resp. Time	$R \geq \sum D_k$	$\max(\sum D_k, ND_{\max} - Z) \leq R \leq N \sum D_k$
Throughput	$X \leq \frac{1}{D_{\max}}$	$\frac{N}{N \sum D_k + Z} \leq X \leq \min\left(\frac{1}{D_{\max}}, \frac{N}{\sum D_k + Z}\right)$
N^*		$N^* = \frac{\sum D_k + Z}{D_{\max}}$

Table V. AVAILABILITY: VARIABLES

Variable	Definition
λ	Failure rate
$F(t)$	Failure probability
$R(t)$	Reliability
A	Availability
$MTTF$	Mean Time to Failure
$MTTR$	Mean Time to Repair
AFR	Annualized Failure Rate
P_{on}	# power hours per year

Table VI. AVAILABILITY: RELATIONS

Relations
$\lambda = 1/MTTF$ †
$R(t) = 1 - F(t)$
$R(t) = e^{-\frac{t}{MTTF}}$ †
$R(t) \approx 1 - \frac{t}{MTTF}$ if $t << MTTF$ †
$AFR = e^{-\frac{P_{on}}{MTTF}}$ †
$R_{Ser.}(t) = \prod R_i(t)$
$R_{Par.}(t) = 1 - \prod (1 - R_i(t))$
$MTTF_{Ser.} = \left(\sum \frac{1}{MTTF_i}\right)^{-1}$ †
$MTTF_{Ser.} = \frac{MTTF}{N}$ if *i.i.d.* †
$MTTF_{Par.} = MTTF \sum_{n=1}^{N} \frac{1}{n}$ if *i.i.d.* †
$MTTF_{Par2} = MTTF_1 + MTTF_2 - \frac{MTTF_1 \cdot MTTF_2}{MTTF_1 + MTTF_2}$ †
$A = \frac{MTTF}{MTTF + MTTR}$
$A_{Ser.} = \prod A_i$
$A_{Par.} = 1 - \prod (1 - A_i)$
† Exponential assumption

Table VII. RAID: VARIABLES

Variable	Definition
$MTTF$	Mean Time to Failure
$MTTR$	Mean Time to Repair
$MTTDL$	Mean Time to Data Loss
N	Number of disks in the array
G	Number of disks in a group

Relations

$$MTTDL_{RAID0} = \frac{MTTF}{N}$$
$$MTTDL_{RAID1} = \frac{MTTF^2}{2 \cdot MTTR}$$
$$MTTDL_{RAID1+0} = \frac{MTTF^2}{N \cdot MTTR}$$
$$MTTDL_{RAID0+1} = \frac{2 \cdot MTTF^2}{N^2 \cdot MTTR}$$
$$MTTDL_{RAID5} = \frac{MTTF^2}{N \cdot (N-1) \cdot MTTR}$$
$$MTTDL_{RAID6} = \frac{2 \cdot MTTF^3}{N \cdot (N-1) \cdot (N-2) \cdot MTTR^2}$$
$$MTTDL_{RAID5+0} = \frac{MTTF^2}{N \cdot (G-1) \cdot MTTR}$$
$$MTTDL_{RAID6+0} = \frac{2 \cdot MTTF^3}{N \cdot (G-1) \cdot (G-2) \cdot MTTR^2}$$

$$T_b = T_s + T_l + T_t + T_c$$
$$T_b = (T_s + T_l + T_t)/N + T_c \ ^{\dagger 1}$$
$$T_b = T_{sMax} + 2T_l + T_t/N + T_c \ ^{\dagger 2}$$
$$T_l = \frac{1}{2 \cdot r_r}$$
$$T_t = \frac{B}{r_t}$$
$$T_a = \lceil F/B \rceil \cdot T_b \ ^{\star}$$
$$T_a = \lceil F/B \rceil \cdot [T_t + T_c + (1-l)(T_s + T_l)] \ ^{\circ}$$
$$T_a = \lceil F/B \rceil \cdot [T_c + (T_t + (1-l)(T_s + T_l))/N] \ ^{\circ,\dagger}$$
$$T_a = \lceil F/B \rceil \cdot [T_c + T_t/N + (1-l)(T_{sMax} + 2T_l)] \ ^{\circ}$$

$^{\dagger}1$ for RAID 0, coarse grained
$^{\dagger}2$ for RAID 0, fine grained
* for one file, without locality
$^{\circ}$ for one file, with locality

Table IX. RAID PARITY: VARIABLES

Variable	Definition
D_i	Data on i-th disk ($0 \leq i < N$)
P	Parity data
Q	Second Parity data
g	Parity generator

Table X. RAID 5 AND 6 PARITY P

Parity Computation	$P = \sum_{i=0}^{N-1} D_i$
Parity Update	$P^{new} = P^{old} + D_i^{new} - D_i^{old}$

Table XI. RAID 6 PARITY Q

Parity Computation	$Q = \sum_{i=0}^{N-1} g^i D_i$
Parity Update	$Q^{new} = Q^{old} + g^i (D_i^{new} - D_i^{old})$

Table XII. RAID PARITY: RECONSTRUCTION

Failed	Reconstruction
D_i	$D_i = P - \sum_{j \neq i} D_j$
P	$P = \sum_{i=0}^{N-1} D_i$
Q	$Q = \sum_{i=0}^{N-1} g^i D_i$
D_i and P	$D_i = g^{-i} \left(Q - \sum_{j \neq i} g^j D_j \right)$
D_i and D_k	Solve the system of equations: $\begin{cases} P = D_i + D_j + \sum_{k=0, k \neq i,j}^{N-1} D_k \\ Q = g^i D_i + g^j D_j + \sum_{k=0, k \neq i,j}^{N-1} g^k D_k \end{cases}$

Table XIII. DISKS: VARIABLES

Variable	Definition
T_s	Mean seek time
T_{sMax}	Max seek time
T_t	Mean transfer time (for one block)
B	Block size or Page Size
r_t	Transfer rate
T_l	Rotational latency time
r_r	Rotational speed
T_c	Controller overhead
T_{tP}	Mean transfer time (for one page)
T_{rP}	Mean read time (for one page)
T_b	Mean service time (for one block)
T_a	Mean service time (for one file)
F	File size
l	data locality
N	stripe width

Transformations

The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc (Scala, Java, Python) and pair RDD functions doc (Scala, Java) for details.

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(*func*)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
sample(*withReplacement*, *fraction*, *seed*)	Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator seed.
union(*otherDataset*)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(*otherDataset*)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([*numTasks*])	Return a new dataset that contains the distinct elements of the source dataset.
groupByKey([*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. **Note:** If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using `reduceByKey` or `combineByKey` will yield much better performance. **Note:** By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional `numTasks` argument to set a different number of tasks.
reduceByKey(*func*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) => V. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
aggregateByKey(*zeroValue*)(*seqOp*, *combOp*, [*numTasks*])	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in `groupByKey`, the number of reduce tasks is configurable through an optional second argument.
sortByKey([*ascending*], [*numTasks*])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean `ascending` argument.
join(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through `leftOuterJoin`, `rightOuterJoin`, and `fullOuterJoin`.
cogroup(*otherDataset*, [*numTasks*])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, Iterable<V>, Iterable<W>) tuples. This operation is also called `groupWith`.
cartesian(*otherDataset*)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).
pipe(*command*, *[envVars]*)	Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.
coalesce(*numPartitions*)	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
repartition(*numPartitions*)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
repartitionAndSortWithinPartitions(*partitioner*)	Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. This is more efficient than calling `repartition` and then sorting within each partition because it can push the sorting down into the shuffle machinery.

Actions

The following table lists some of the common actions supported by Spark. Refer to the RDD API doc (Scala, Java, Python) and pair RDD functions doc (Scala, Java) for details.

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset. Note that this is currently not executed in parallel. Instead, the driver program computes all the elements.
takeSample(*withReplacement*, *num*, [*seed*])	Return an array with a random sample of *num* elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(*n*, *[ordering]*)	Return the first *n* elements of the RDD using either their natural order or a custom comparator.
saveAsTextFile(*path*)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.
saveAsSequenceFile(*path*) (Java and Scala)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that either implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
saveAsObjectFile(*path*) (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using `SparkContext.objectFile()`.
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(*func*)	Run a function *func* on each element of the dataset. This is usually done for side effects such as updating an accumulator variable (see below) or interacting with external storage systems.

RDD Persistence

One of the most important capabilities in Spark is *persisting* (or *caching*) a dataset in memory across operations. When you persist an RDD, each node stores any partitions of it that it computes in memory and reuses them in other actions on that dataset (or datasets derived from it). This allows future actions to be much faster (often by more than 10x). Caching is a key tool for iterative algorithms and fast interactive use.

You can mark an RDD to be persisted using the `persist()` or `cache()` methods on it. The first time it is computed in an action, it will be kept in memory on the nodes. Spark's cache is fault-tolerant – if any partition of an RDD is lost, it will automatically be recomputed using the transformations that originally created it.

In addition, each persisted RDD can be stored using a different *storage level*, allowing you, for example, to persist the dataset on disk, persist it in memory but as serialized Java objects (to save space), replicate it across nodes, or store it off-heap in Tachyon. These levels are set by passing a `StorageLevel` object (Scala, Java, Python) to `persist()`. The `cache()` method is a shorthand for using the default storage level, which is `StorageLevel.MEMORY_ONLY` (store deserialized objects in memory). The full set of storage levels is:

Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Store RDD in serialized format in Tachyon. Compared to MEMORY_ONLY_SER, OFF_HEAP reduces garbage collection overhead and allows executors to be smaller and to share a pool of memory, making it attractive in environments with large heaps or multiple concurrent applications. Furthermore, as the RDDs reside in Tachyon, the crash of an executor does not lead to losing the in-memory cache. In this mode, the memory in Tachyon is discardable. Thus, Tachyon does not attempt to reconstruct a block that it evicts from memory.

Note: *In Python, stored objects will always be serialized with the Pickle library, so it does not matter whether you choose a serialized level.*

Spark also automatically persists some intermediate data in shuffle operations (e.g. `reduceByKey`), even without users calling `persist`. This is done to avoid recomputing the entire input if a node fails during the shuffle. We still recommend users call `persist` on the resulting RDD if they plan to reuse it.

Answer sheet:

Last Name / Cognome:

....

First Name / Nome:

....

Answers must be given exclusively on this sheet: answers given on the other sheets will be ignored.

Student ID : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
Student ID : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
Student ID : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
Student ID : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
Student ID : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
Student ID : ☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

Disks

Question 01 : ☐A ☐B ☐C ☐D
Question 02 : ☐A ☐B ☐C ☐D
Question 03 : ☐A ☐B ☐C ☐D
Question 04 : ☐A ☐B ☐C ☐D
Question 05 : ☐A ☐B ☐C ☐D
Question 06 : ☐A ☐B ☐C ☐D
Question 07 **(R)**: ☐A ☐B ☐C ☐D

Performance Evaluation

Question 08 : ☐A ☐B ☐C ☐D
Question 09 : ☐A ☐B ☐C ☐D
Question 10 : ☐A ☐B ☐C ☐D
Question 11 : ☐A ☐B ☐C ☐D
Question 12 : ☐A ☐B ☐C ☐D

Dependability

Question 13 : ☐A ☐B ☐C ☐D
Question 14 : ☐A ☐B ☐C ☐D
Question 15 : ☐A ☐B ☐C ☐D
Question 16 : ☐A ☐B ☐C ☐D
Question 17 : ☐A ☐B ☐C ☐D
Question 18 **(R)**: ☐A ☐B ☐C ☐D

Cloud Computing

Question 19 : ☐A ☐B ☐C ☐D
Question 20 : ☐A ☐B ☐C ☐D

ITIL and DevOps

Question 21 : ☐A ☐B ☐C ☐D
Question 22 : ☐A ☐B ☐C ☐D
Question 23 : ☐A ☐B ☐C ☐D
Question 24 **(R)** : ☐A ☐B ☐C ☐D

Virtualization

Question 25 : ☐A ☐B ☐C ☐D
Question 26 : ☐A ☐B ☐C ☐D
Question 27 : ☐A ☐B ☐C ☐D
Question 28 : ☐A ☐B ☐C ☐D
Question 29 : ☐A ☐B ☐C ☐D

BigData

Question 30 : ☐A ☐B ☐C ☐D
Question 31 : ☐A ☐B ☐C ☐D
Question 32 : ☐A ☐B ☐C ☐D
Question 33 : ☐A ☐B ☐C ☐D
Question 34 : ☐A ☐B ☐C ☐D
Question 35 : ☐A ☐B ☐C ☐D
Question 36 **(R)** : ☐A ☐B ☐C ☐D