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

25-09-2015

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

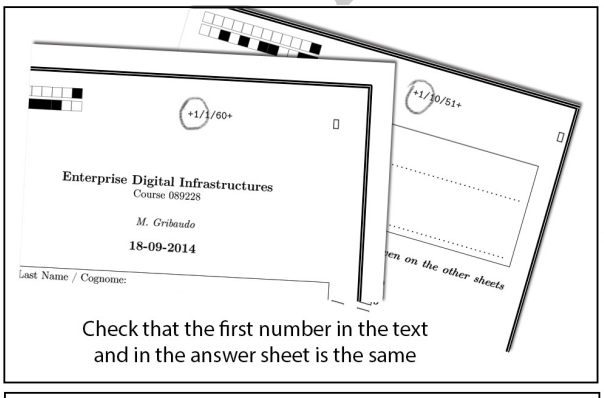
Students must use pen (black or blue) to mark answers (no pencil).
Students are permitted to use a non-programmable calculator.

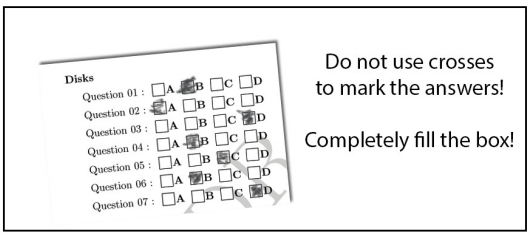
Students are NOT permitted to copy anyone else's answers, pass notes amongst themselves, or engage in other forms of misconduct at any time during the exam.

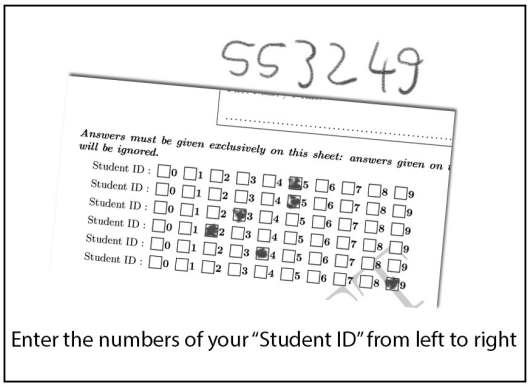
Students are NOT permitted to use mobile phones and similar connected devices.

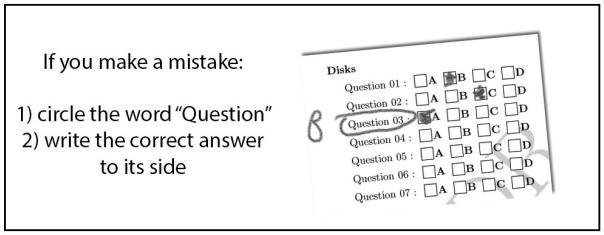
Scores: correct answers +1 point, unanswered questions 0 points, wrong answers -0.333 points.

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









For your examination, preferably print documents compiled from auto-multiple-choice.

Disks

Consider the following set of disks connected in RAID 6, with generator $g = 2$:

Block A1		Block A2		Block A3		Block A4		Block A5		Block A6	
D0	2	D1	3	D2		D3	1	P	7	Q	

Block B1		Block B2		Block B3		Block B4		Block B5		Block B6	
D0	8	D1	3	D2		P	12	Q	18	D3	

Block C1		Block C2		Block C3		Block C4		Block C5		Block C6	
D0	0	D1	0	P	1	Q	8	D2	0	D3	

Figure 1: A RAID 6 configuration.

Question 1 What data is contained in blocks $A3$ and $A6$?

A $D_2 = 13$, $Q = 68$ B $D_2 = 1$, $Q = 20$ C $D_2 = 0$, $Q = 18$ D Cannot be computed

SOLUTION:
$D_2 = 1$, $Q = 20$

Question 2 Data in block $C2$ is changed to $D_1 = 3$. Which are the new values of the parity blocks?

A $P = 4$, $Q = 14$ B $P = 4$, $Q = 20$ C Cannot be computed D $P = 4$, $Q = 11$

SOLUTION:
$P = 4$, $Q = 14$.

After a reconfiguration, the disks will be used in RAID 0. The transfer time of the disks is 128 MB/s, they spin at 10000 RPM, the maximum seek time is 6 ms, and the average is 3 ms. Data is divided into 4 KB blocks.

Question 3 What transfer time required to read a block?

A $0,2441$ ms B $0,03051$ ms C $0,03125$ ms D $0,0038$ ms

SOLUTION:
$4/(128 \cdot 1024) \cdot 1000$ ms $= 0,03051$ ms.

Question 4 Which is the average rotational latency?

A 6 ms B 12 ms C 1 ms D 3 ms

SOLUTION:
$60000/(10000 \cdot 2) = 3$ ms

Question 5 If we consider a locality of $l = 99\%$, which will be the transfer time with required to read 1 GB of data if the disk is formatted using a coarse grained interleaving?

A $5,2654$ s B $3,9547$ s C $5,9321$ s D $6,5762$ s

SOLUTION:
1 GB $\cdot 1024^2/4 \cdot (0,03051 + (3+3) * (1 - 0.99))/1000/6 = 3,9547$ s.

Question 6 If we consider a locality of $l = 99.9\%$, which will be the transfer time with required to read 1 GB of data if the disk is formatted using a fine grained interleaving?

A $4,4791$ s B $3,6926$ s C $2,9062$ s D $5,1457$ s

SOLUTION:
1 GB $\cdot 1024^2/4 \cdot (0,03051/6 + (6+6) * (1-0.999))/1000/6 = 4,4791$ s.

Question 7 **Reserve** - *Do not answer unless explicitly stated during the exam*
If the $MTTR = 1200$ days, and the $MTTF = 600$ days, the $MTTBL$ of the RAID 0 version of the system is:

A 300 days B 100 days C 200 days D 150 days

SOLUTION:
$600/6 = 100$ days.

Performance evaluation

A RAID 5 is composed by a controller and four disks. For a read request when all the disks are working, three out of the four disks are used and the accesses are uniformly distributed among them (hint: you can use this information to guess the visits of the disks). The service time of the controller is $S_{Ctrl} = 10$ ms, and the service time of each disk is $S_{Disk} = 12$ ms.

Question 8 The *demand* of each *Disk* is:

A $D_{Disk} = 4$ ms B $D_{Disk} = 12$ ms C $D_{Disk} = 3$ ms D $D_{Disk} = 9$ ms

SOLUTION:
$v_{Disk} = 3/4$ (since the access generated by a read request are equally distributed among three of the four disks).
$D_{Disk} = S_{Disk} \cdot v_{Disk} = 12\frac{3}{4} = 9$ ms.

Question 9 The *maximum arrival rate* that the RAID can handle for read operation is:

A $\lambda_{max} = 111.11$ req/s. B $\lambda_{max} = 83.33$ req/s. C $\lambda_{max} = 100$ req/s. D $\lambda_{max} = 21.74$ req/s.

SOLUTION:
$\lambda_{max} = \frac{1}{D_{max}} = \frac{1}{0.010} = 100$ req/s.

Question 10 The *minimum response time* that the RAID can offer for read operation is:

A $R_{min} = 46$ ms B $R_{min} = 19$ ms C $R_{min} = 58$ ms D $R_{min} = 22$ ms

SOLUTION:
$R_{min} = \sum D_k = 10 + 9 \cdot 4 = 46$ ms.

When the utilization of the disk is $U_{Disk} = 45\%$, the response time of the system is 160 ms. In this scenario:

Question 11 The *average number of requests in the system* is:

A $N = 10.67$
B $N = 8$
C $N = 9$
D $N \to \infty$

SOLUTION:
$U_{Disk} = X \cdot D_{Disk}$. $X = U_{Disk}/D_{Disk} = 0.45/0.009 = 50$ req/s
$N = X \cdot R = 160 \cdot 0.05 = 8$

Question 12 The *utilization* of the controller is U_{Ctrl} is:

A $U_{Ctrl} = 50\%$
B The controller saturates
C $U_{Ctrl} = 90\%$
D $U_{Ctrl} = 20\%$

SOLUTION:
$U_{Ctrl} = X \cdot D_{Ctrl} = 50 \cdot 0.01 = 50\%$.

Dependability

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

Question 13 Consider two components A and B in a serial configuration as in block C_1 of Figure **??**. They have the following characteristics: $MTTF_A = 500$ *days*, $MTTR_A = 5$ *days*; $MTTF_B = 200$ *days*, $MTTR_B = 2$ *days*. The reliability of sub-system C_1 at $t = 50$ *days* is equal to:

A 0.7046 B 0.8361 C 0.5897 D 0.9789

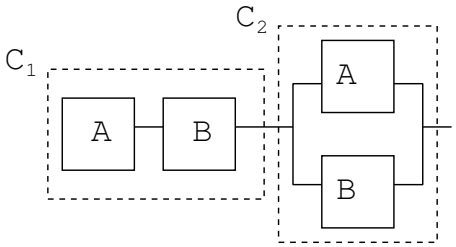


Figure 2: RBD.

SOLUTION:
$R_A(50) = e^{-50/500} = 0.9048$ $R_B(50) = e^{-50/200} = 0.7788$ $R_{C_1} = R_A * R_B = 0.7046$
Question 14 In the previous configuration is required to change component B in order to achieve an MTTF of block C_1, computed without repair, equal to $MTTF_{C1} = 187.5$. The required upgrade is:

A $MTFF_B = 800$ B Impossible C $MTFR_B = 1$ D $MTFF_B = 300$

SOLUTION:
$MTTF_{C_1} = 187.5 = 1/(1/500 + 1/MTTF_B)$ $500MTTF_B/(MTTF_B + 500) = 187.5$ $MTTF_B = 300$
Question 15 Consider the same components of **Question 13** in a parallel configuration as in block C_2 of Figure **??**. The MTTF of C_2computed without repair is equal to:

A 557.1428 B 142.8571 C 350 D 99

SOLUTION:
$MTTF_{C_2} = MTTF_A + MTTF_B - MTTF_A * MTTF_B/(MTTF_A + MTTF_B) = 557.1428$

Question 16 The availability of block C_2 is:

A 0.7892 B 0.9801 C 0.9999 D 0.495

SOLUTION:
$A_A = MTTF_A/(MTTF_A + MTTR_A) = 500/(500 + 5) = 0.9900$
$A_B = MTTF_B/(MTTF_B + MTTR_B) = 200/(200 + 2) = 0.9900$
$A_{A||B} = 1 - (1 - A_A)^2 = 0.9999$
Question 17 The availability of the whole system depicted in Figure **??** is:

A 0.9999 B 0.98 C 0.625 D 0.8

SOLUTION:
$A_{Sys} = 0.9999 * 0.99^2 = 0.98$

Question 18 **Reserve** - *Do not answer unless explicitly stated during the exam*
The approximate value of the reliability of the component A at $t = 150$ is equal to:

A 0.75 B Not applicable. C 0.85467 D 0.25

SOLUTION:
Not applicable since 150«200 is not true.

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] Privacy: cloud services are immune to privacy problems
- [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 Infrastructure-as-a-Service?

- [A] Reduced software costs
- [B] Reduced long term IT costs
- [C] No need to install applications
- [D] Reduced hardware investments

ITIL and DevOps

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

- [A] Increased performance of computer
- [B] Increased mean-time-to-failure
- [C] Reduced time to restore services
- [D] Decreased mean-time-to-repair

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 is the main goal of the Problem Management process?

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

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.

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.

Virtualization

Question 24 Which of the following is a definition of *binary-code translation*?

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

Question 25 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 guarantee to a virtual machine a minimum set of resources, regardless of the loads of other virtual machines running on the same host
- [C] It is possible to run different guest operating systems on the same host
- [D] It is possible to run a virtual machine inside a virtual machine

Question 26 Which of the following statements is correct when talking about *hosted* hypervisor?

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

Question 27 Which is the definition of *bare-metal* hypervisor?

- [A] The system is not virtualized
- [B] The hypervisor is using binary code translation
- [C] The hypervisor is using ring -1 of the processor to provide hardware-assisted virtualization
- [D] The hypervisor is running directly on the hardware

Question 28 How many CPU levels are required to efficiently run a traditional operating system – such as Linux or Windows – within a virtual machine without using software emulation?

- [A] Two
- [B] Five
- [C] Three
- [D] Four

Big Data

The following Apache Spark code is a script to manage exam data. The imaginary exam is composed of 2 exercises. The maximum grade that you can get is 32 (grades greater than 30 are laude). The maximum grade of each exercise is 16. The minimum score to pass the exam is 18. This should be nothing new to you.

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  from math import sqrt
4  import random
5
6  ### creates data with matricola and grade
7  exercisesGradesRow = namedtuple("matricola_exercises_results", ["matricola", "ex_1","ex_2"])
8  matricolaExercisesRdd = (sc.parallelize([exercisesGradesRow(
9                matricola=i+800000,
10               ex_1=random.randint(0,16),
11               ex_2=random.randint(0,16))
12               for i in range(1,1000)]))
13 matricolaExercisesRdd.take(1)
14 # [matricola_exercises_results(matricola=800001, ex_1=6, ex_2=2)]
15
16 ### creates data with matricola and email
17 matricolaEmailRdd = (matricolaExercisesRdd
18                    .map(lambda matricolaGradeRow:
19                         (matricolaGradeRow.matricola,
20                          str(matricolaGradeRow.matricola*10)+"@mail.polimi.it")))
21 matricolaEmailRdd.take(1)
22 # [(800001, '8000010@mail.polimi.it')]
23
24 ### sums the grade in each exercise to obtain the final grade
25 matricolaGradeRow = namedtuple("matricola_and_grade", ["matricola", "grade"])
26 matricolaGradeRdd = (matricolaExercisesRdd.map(lambda ex:
27  matricolaGradeRow(matricola=ex.matricola, grade=ex.ex_1+ex.ex_2)))
28 matricolaGradeRdd.take(1)
29 # [matricola_and_grade(matricola=800001, grade=8)]
30
31 ### creates an RDD with students that have a grade less than 18
32 rejectedStudentsRdd = matricolaGradeRdd.<FILL IN>
33 rejectedStudentsRdd.take(1)
34 # [matricola_and_grade(matricola=800002, grade=5)]
35
36 ### creates a list of rejected student ids
37 rejectedStudentsMatricolaList = (rejectedStudentsRdd.map(lambda matgrade:
38  matgrade.matricola).collect())
39 print rejectedStudentsMatricolaList[:10]
40 # [800002, 800007, 800008, 800009, 800010, 800012, 800014, 800015, 800018, 800020]
41
42 ### computes the average grade of rejected students
43 sumAndCountGradeRejectedStudents = (rejectedStudentsRdd
44                                  .map(lambda matgrade:(matgrade.grade,1))
45                                  .reduce(sumAndCountFun))
```

```
46
47  print "Average rejected students grade: %f" % (
48  sumAndCountGradeRejectedStudents[0]/float(sumAndCountGradeRejectedStudents[1]))
49  # Average rejected students grade: 11.333333
50
51  ### creates an rdd with matricolaGradeRow of students that passed the exam
52  passedStudentsRdd = matricolaGradeRdd.filter(lambda matgrade: matgrade.matricola
53  not in rejectedStudentsMatricolaList)
54  passedStudentsRdd.first()
55  # matricola_and_grade(matricola=800001, grade=24)
56
57  ### produces the emails for rejected students
58  emailRow = namedtuple("email_and_content", ["email", "content"])
59  emailsNotPassed = (matricolaEmailRdd.filter(lambda x: x[0] in rejectedStudentsMatricolaList)
60                     .map(lambda x: emailRow(email=x[1],content="Dear %d,\nWe are sorry to inform
61                      you that you did not pass the exam.\nCheers" % x[0]))
62                    )
63  emailsNotPassed.first()
64  # email_and_content(email='8000020@mail.polimi.it', content='Dear 800002,\nWe are sorry
65  # to inform you that you did not pass the exam.\nCheers')
66
67  ### produces the emails for students that passed the exam
68  def checkIfLaude(grade):
69      if grade > 30:
70          return "30L"
71      return str(grade)
72
73  rdd1 = (passedStudentsRdd.map(lambda matgrade:(matgrade.matricola, matgrade.grade))
74          .join(matricolaEmailRdd))
75  rdd1.cache()
76  emailsPassed = (rdd1
77                  .map(lambda x: emailRow(
78                      email=x[1][1], content="Dear %d,\nWe are happy to inform
79                       you that you passed the exam with grade %s\nCheers" % (
80                       x[0],checkIfLaude(x[1][0])))) )
81  print emailsPassed.first()
82  # email_and_content(email='8002560@mail.polimi.it', content='Dear 800256,\nWe are happy
83  # to inform you that you passed the exam with grade 21\nCheers')
84
85  print rdd1.take(1)
86  ### OUTPUT NOT DISCLOSED
87
88  print "Number of emails to send to students who passed the exam: %d" % emailsPassed.count()
89  # Number of emails to send to students who passed the exam: 432
90
91  ### compute some statistics over students' grades
92  sum = lambda x, y: x + y
93  gradeCount = matricolaGradeRdd.count()
94  gradeSum = matricolaGradeRdd.map(lambda x:x[1]).reduce(sum)
95  mean = float(gradeSum) / gradeCount
96  std = matricolaGradeRdd.map(lambda x:(x[1]-mean)**2).reduce(sqrt(sum)) / gradeCount
97  ordered = <FILL_IN>
98
99  print "mean: %f" % mean
100 print "std: %f" % std
101 print "(1st quartile, median, 3rd quartile): (%d, %d, %d)" % \
102     (ordered[int(.25*gradeCount)], ordered[int(.5*gradeCount)], ordered[int(.75*gradeCount)])
103 # mean: 16.293293
104 # std: 6.957548
105 # (1st quartile, median, 3rd quartile): (11, 16, 21)
```

Question 29 What is the proper definition for sumAndCountFun used by the instruction at line 43?

- [A] sumAndCountFun = lambda x, y: x[0] + y[0], x[1] + y[1]
- [B] sumAndCountFun = lambda x: (x[0][0] + x[0][1], x[1][0] + x[1][1])
- [C] sumAndCountFun = lambda x, y: [x(0) + y(0), x(1) + y(1)]
- [D] sumAndCountFun = lambda x, y: (x[0] + y[0], x[1] + y[1])

SOLUTION:
sumAndCountFun = lambda x, y: (x[0] + y[0], x[1] + y[1])

Question 30 Given the previous outputs, which of the following is a possible output for the instruction rdd1.take(1) at line 85?

- [A] [('802816', ('23', '8028160@mail.polimi.it'))]
- [B] [((23, '8028160@mail.polimi.it'), 802816)]
- [C] [(802816, (23, '8028160@mail.polimi.it'))]
- [D] [(802816, 23, '8028160@mail.polimi.it')]

SOLUTION:
[(802816, (23, '8028160@mail.polimi.it'))]

Question 31 What of the following is a valid and correct Spark instruction to compute the 'ordered' vector (line 97), later used to **compute** and **print** the 3 quartiles (line 102)?

- [A] ordered = sorted(matricolaGradeRdd.map(lambda x: (x[1], x[0])).collect())
- [B] ordered = matricolaGradeRdd.map(lambda x: (x[1], x[0])).sortByKey().map(lambda x:x[0]).collect()
- [C] ordered = matricolaGradeRdd.map(lambda x: (x[1], x[0])).sortByKey().collect()
- [D] ordered = matricolaGradeRdd.sortByKey().map(lambda x:x[1]).collect()

SOLUTION:
ordered = matricolaGradeRdd.map(lambda x: (x[1], x[0])).sortByKey().map(lambda x:x[0]).collect()
NOTE: *sorted(matricolaGradeRdd.map(lambda x: (x[1], x[0])).collect())* and *matricolaGradeRdd.map(lambda x: (x[1], x[0])).sortByKey().collect()* are incorrect because the resulting 'ordered' vector will be made of tuples, raising an error at the print instruction

Question 32 Is the expression used to compute variable 'std' correct (line 96)?

- [A] No, the correct version is std = matricolaGradeRdd.map(lambda x:(x[1]-mean)**2).reduce(lambda x,y:sqrt(x+y)) / gradeCount
- [B] Yes
- [C] No, the correct version is std = sqrt(matricolaGradeRdd.map(lambda x:(x[1]-mean)**2).reduce(sum) / gradeCount))).
- [D] No, the correct version is std = sqrt(matricolaGradeRdd.map(lambda x:(x[1]-mean)**2).reduce(sum)) / gradeCount

SOLUTION:
No, the correct version is std = sqrt(matricolaGradeRdd.map(lambda x:(x[1]-mean)**2).reduce(sum) / gradeCount)

Question 33 Complete line 32 to extract students with grade less than 18. Note: it has to be compliant with the output

A map(lambda matgrade: matgrade.grade).filter(lambda x: x < 18)

B filter(lambda x: x < 18)

C filter(lambda matgrade: matgrade.grade < 18)

D map(lambda matgrade: matgrade < 18)

SOLUTION:
filter(lambda matgrade: matgrade.grade < 18)

Question 34 In which line variable rdd1 (defined at line 73) gets actually computed, even partially?

A 88 B 85 C 73 D 81

SOLUTION:
81

Question 35 **Reserve** - *Do not answer unless explicitly stated during the exam*
Which of the following Spark instructions can be used to retrieve the top 3 student that:

- passed the exam
- with the highest grades in ex_1 (in descending ex_1 grade order)

A
```
matricolaExercisesRdd.map(lambda exg:  (exg.matricola,exg.ex_1,exg.ex_1
+exg.ex_2) ).takeOrdered(20, lambda x:  -x[1])
```

B
```
matricolaExercisesRdd.map(lambda exg:  (exg.matricola,exg.ex_1,exg.ex_1
+exg.ex_2) ).filter(lambda x:  x[2] >= 18).takeOrdered(20, lambda x:
x[2])
```

C
```
matricolaExercisesRdd.map(lambda exg:  (exg.matricola,exg.ex_1,exg.ex_1
+exg.ex_2) ).filter(lambda x:  x[2] >= 18).takeOrdered(3, lambda x:
-x[1])
```

D
```
matricolaExercisesRdd.map(lambda exg:  (exg.matricola,exg.ex_1,exg.ex_1
+exg.ex_2) ).filter(lambda x:  x[2] >= 18).takeOrdered(3, lambda x:
-x[2])
```

SOLUTION:
```
matricolaExercisesRdd.map(lambda exg:  (exg.matricola,exg.ex_1,exg.ex_1 +exg.ex_2)
).filter(lambda x:  x[2] >= 18).takeOrdered(3, lambda x:  -x[1])
```

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

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

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

†1 for RAID 0, coarse grained
†2 for RAID 0, fine grained
⋆ for one file, without locality
° for one file, with locality

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

Virtualization

Question 24 : ☐A ☐B ☐C ☐D
Question 25 : ☐A ☐B ☐C ☐D
Question 26 : ☐A ☐B ☐C ☐D
Question 27 : ☐A ☐B ☐C ☐D
Question 28 : ☐A ☐B ☐C ☐D

BigData

Question 29 : ☐A ☐B ☐C ☐D
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 (R) : ☐A ☐B ☐C ☐D