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

13-07-2015

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

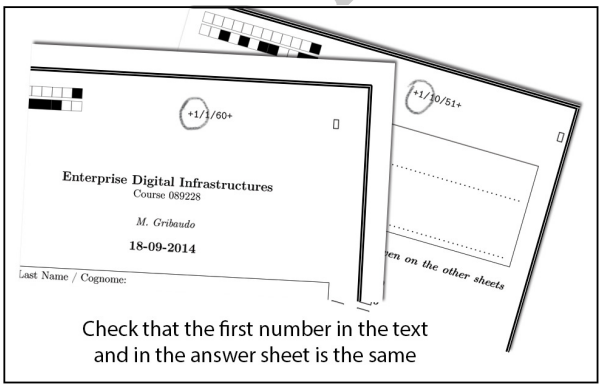
Students must use pen (black or blue) to mark answers (no pencil).
Students are permitted to use a non-programmable calculator.

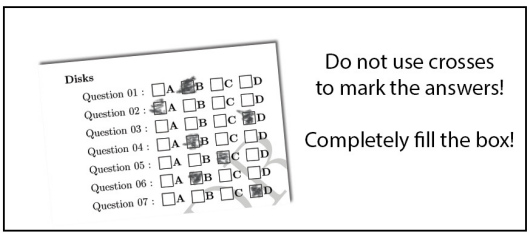
Students are NOT permitted to copy anyone else's answers, pass notes amongst themselves, or engage in other forms of misconduct at any time during the exam.

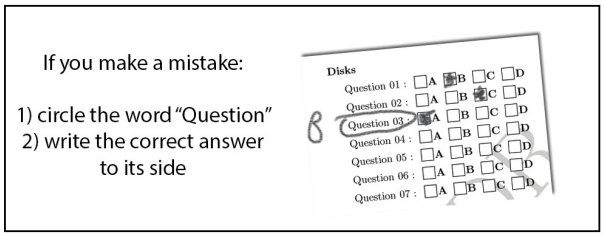
Students are NOT permitted to use mobile phones and similar connected devices.

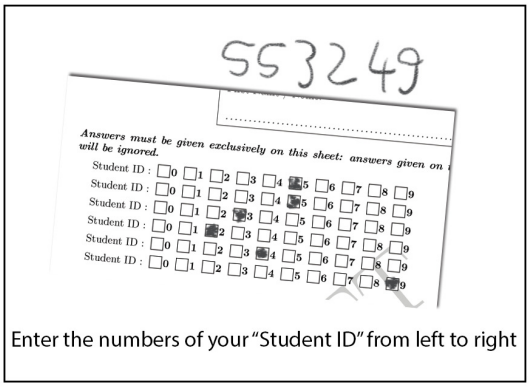
Scores: correct answers +1 point, unanswered questions 0 points, wrong answers -0.333 points.

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









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

Disks

An HDD has a rotational speed of 6000 RPM, an average seek time of 6 ms, a controller overhead of 0.2 ms, and a transfer rate of 50 MB/s. Block size is 4 KB.

Question 1 Which is the average latency of the disk?

A 6 ms B 5 ms C 10 ms D 3 ms

SOLUTION:
$T_{lat} = (60000/6000)/2 = 5\ ms$.

Question 2 Which is the transfer time for a block?

A 0.078125 ms B 0.390625 ms C 0.019531 ms D 0.78125 ms

SOLUTION:
$T_{tr} = 4/(50 \cdot 1024) \cdot 1000 = 0.078125\ ms$.

Question 3 Which is the average access time for a block?

A 11.078125 ms B 11.278125 ms C 11.219531 ms D 16.27812 ms

SOLUTION:
$T_{Acc} = T_{seek} + T_{lat} + T_{ctrl} + T_{tr} = 6 + 5 + 0.078125 + 0.2 = 11.278125\ ms$.

Question 4 How long will it take to transfer a file of 1 MB without locality?

A 2.836 s B 4.167 s C 2.8722 s D 2.8872 s

SOLUTION:
$T_{1MB} = 1024/4 \cdot T_{Acc} = 256 \cdot 11.278125 = 2.8872s$

Question 5 How long will it take to transfer a file of 10 MB with a 93.75% locality (without taking into account the controller overhead)?

A 121.59 s B 1.96 s C 12.841 s D 49.83 s

SOLUTION:
$T_{10MB} = 10 \cdot 1024/4\,((1 - 0.9375) \cdot (T_{seek} + T_{lat}) + T_{tr}) = 2560(0.0625 \cdot 11 + 0.078125) = 1.96\ s$.

Question 6 What is the write amplification factor?

A The number of disk accessed caused by a write operation of a single cluster on an SSD
B The number of disk accessed caused by a write operation of a single block on an SSD
C The number of write accesses caused by a read operation of a single page on an SSD
D The number of write accesses caused by a read operation of a single block on an SSD

SOLUTION:
The number of disk accessed caused by a write operation of a single cluster on an SSD

Performance evaluation

Consider a system composed by three stations: the *CPU* that has a visit ratio $v_{CPU} = 1.2$ and an average service time of 5ms; the *disk*, characterized by a throughput of 120job/s, and a demand of 3.5ms; and the *network* whose demand is 2ms, and throughput is 60job/s. The system throughput is 120job/s, and the response time when there are $N = 24$ jobs in the system is $R = 50$ms.

Question 7 The *demand* of the *CPU* is:

A $D_{CPU} = 2.4$ms B $D_{CPU} = 6$ms C $D_{CPU} = 5$ms D $D_{CPU} = 4.1667$ms

SOLUTION:
$D_{CPU} = S_{CPU} \cdot v_{CPU} = 5 \cdot 1.2 = 6$ms.

Question 8 The *visits* to the *disk* are:

A $v_{disk} = 0.42$ B $v_{disk} = 34.268$ C $v_{disk} = 0.3843$ D $v_{disk} = 1$

SOLUTION:
$v_{disk} = \frac{X_{disk}}{X} = 120/120 = 1$.

Question 9 The *average service time* of the *network* is:

A $S_{net} = 2$ms B $S_{net} = 6.6667$ms C $S_{net} = 4$ms D $S_{net} = 1.3333$ms

SOLUTION:
$S_{net} = \frac{D_{net}}{v_{net}} = \frac{D_{net}}{\frac{X_{net}}{X}} = 2/(60/120) = 4$ms.

Question 10 The *think time* is:

A $Z = 150$ms B $Z = 250$ms C $Z = 350$ms D $Z = 0$ms

SOLUTION:
$R = N/X - Z$, $Z = N/X - R$, $Z = 24/0.12 - 50 = 150$ms.

Question 11 The average *response time* of system when $N = 48$ is:

A $R = 400$ms
B $138\text{ms} \leq R \leq 552$ms
C $R = 250$ms
D $R = 150$ms

SOLUTION:
Since we do not know the throughput for $N = 48$, we cannot apply Little's law to compute the response time exactly. We can only compute its asymptotic bounds: $138\text{ms} \leq R \leq 552$ms.

Question 12 **Reserve** - *Do not answer unless explicitly stated during the exam*
The *asymptotic bounds of the throughput* of system when $N = 48$ are:

A $68.376 \leq X \leq 166.667$job/s
B $56.388 \leq X \leq 148.607$job/s
C $X \leq 166.667$job/s
D $86.957 \leq X \leq 166.667$job/s

SOLUTION:
$68.376 \leq X \leq 166.667$job/s.

Dependability

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

Question 13 Consider a system built by two different components in parallel. Assume for component A: $MTTF_A = 200$ *days* and $MTTR_A = 1$ *days* and for component B: $MTTF_B = 500$ *days* and $MTTR_B = 5$ *days*. The reliability of the system at $t = 10$ *days* is equal to:

A 0.99903 B 0.00342 C 0.95134 D 0.558034

SOLUTION:
$R_A(10) = e^{-10/200} = 0.95123$ $R_B(10) = e^{-10/500} = 0.98019$ $R_{sys} = 1 - (1 - 0.95123)(1 - 0.98019) = 0.99903$

Question 14 The $MTTF$ computed without repair of the previous system is equal to:

A 0.95134 B 700 C 0.00342 D 557.14285

SOLUTION:
$MTTF_{sys} = MTTF_A + MTTF_B - 1/(1/MTTF_A + 1/MTTF_B) = 200 + 500 - 1/(1/200 + 1/500) = 557.14285$

Question 15 Let us now consider a generic component D. Compute the minimum integer value of $MTTF_D$ in order to have at $t = 10$ *days* a reliability $R_D(10) \geq 0.99$.

A 995 B 1023 C 823 D 99

SOLUTION:
$R_D(10) \geq 0.99$ $e^{-10/MTTF_D} \geq 0.99$ $MTTF_D \geq 995$

Consider now the components A and B organized as in Figure 1 and assume they have the characteristics specified in Question 13.

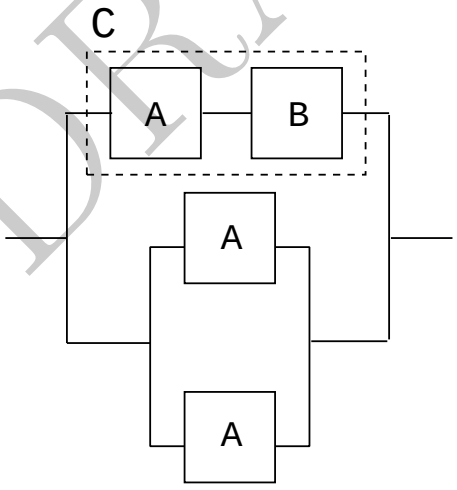


Figure 1: RBD.

Question 16 The MTTF without repair of block C is equal to:.

A 0.9238 B 75.56789 C 142.85714 D 250

SOLUTION:
$MTTF_C = 1/(1/MTTF_A + 1/MTTF_B) = 1/(1/200 + 1/500) = 142.85714$

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

A 0.985159 B 0.99999 C 0.5625 D 0.92

SOLUTION:
$A_A = 200/(200+1) = 0.99502 \quad A_B = 500/505 = 0.99009$
$A_{sys} = 1-(1-A_A A_B)(1-(1-(1-A_A)^2)) = 0.99999$

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 the key benefits for small-medium enterprises when using Infrastructure-as-a-Service instead of traditional housing of servers?

- A Possibility to access applications any-moment any-where
- B Usage of virtualization
- C Reduced hardware investments
- D Reduced long term IT costs

Question 20 Which of the following is the most important technological factor enabling Software-as-a-Service?

- A Adoption of Infrastructure-as-a-Service
- B Widespread diffusion of virtualization
- C Adoption of blade servers
- D Possibility to access Internet any-moment any-where

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 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.

Question 24 **Reserve** - *Do not answer unless explicitly stated during the exam*
Which of the following is one of the goals of the Change Management process in ITIL

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

Solution. Evaluate costs and benefits of fixing bugs

Virtualization

Question 25 Which is the corresponding alternative technology to *Bare-metal*?

A Hosted
B Hardware-assisted
C Para-virtualization
D Micro-kernel

Solution. Hosted

Question 26 Which is the corresponding alternative technology to *Full-virtualization*?

A Hosted
B Micro-kernel
C Hardware assisted
D Para-virtualization

Solution. Para-virtualization

Question 27 Which is the corresponding alternative technology to *Monolitic*?

A Micro-kernel
B Para-virtualization
C Hardware assisted
D Hosted

Solution. Micro-kernel

Question 28 Which is the corresponding alternative technology to *Binary code translation*?

A Hardware assisted
B Micro-kernel
C Hosted
D Para-virtualization

Solution. Hardware assisted

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

A Full virtualization provides better sharing of resources

B Full virtualization provides light-weight virtual machines

C Full virtualization does not allow to run different host operating systems

D Full virtualization provides better insulation between virtual machines

Solution. Full virtualization provides better insulation between virtual machines

Big Data

The following Apache Spark code processes two textual datasets, *apache.access.log* and *countries.txt*, which respectively contains the log records for an online web service and a list of country codes for different server IP addresses. 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, 6 and 15) 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 19, 25 and 31) 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.

```
1  ### imports and custom type definitions
2  from collections import namedtuple
3
4  Row = namedtuple("Row", ["server_ip", "request", "status_code"])
5
6  ### Helper function to parse an input line of the log file
7  def parse(line):
8      lineVect = line.split("\t")
9      return Row(
10         server_ip = lineVect[0],
11         request = lineVect[1],
12         status_code = int(lineVect[2]))
13
14
15 ### Parse the log dataset
16 logPath = "data/apache.access.log"
17 logRawRdd = sc.textFile(logPath)
18 print logRawRdd.take(3)
19 # [u'203.17.79.20\t"GET /images/launch-logo.gif HTTP/1.0"\t200',
20 # u'internet-gw.watson.ibm.com\t"GET /shuttle/missions/51-l/images/850128.GIF HTTP/1.0"\t200',
21 # u'129.237.82.113\t"GET /images/ksclogosmall.gif HTTP/1.0"\t200']
22
23 logRdd = logRawRdd.map(parse)
24 print logRdd.take(3)
25 # [Row(server_ip=u'203.17.79.20', request=u'"GET /images/launch-logo.gif HTTP/1.0"', status_code=200),
26 # Row(server_ip=u'internet-gw.watson.ibm.com', request=u'"GET /shuttle/missions/51-l/images/850128.GIF HTTP/1
27 # Row(server_ip=u'129.237.82.113', request=u'"GET /images/ksclogosmall.gif HTTP/1.0"', status_code=200)]
28
29 nentries = logRdd.<FILL_IN>
30 print "The total number of log entries is %d" % nentries
31 # The total number of log entries is 1000
32
33
34 ### ANALYZE REQUEST PATHS
35 ### Helper function that returns list of directories in the path of request
36 def get_dirs(request_str):
37     return (request_str
38             .split(" ")[1]
39             .split("/")[1:-1])
40 directoriesRdd = logRdd.map(lambda x: get_dirs(x.request))
41 leastRequestedDir = (directoriesRdd
42                      .map(lambda x: (x, 1))
43                      .reduceByKey(lambda x, y: x + y)
44                      .takeOrdered(1))
45 print leastRequestedDir
46 # [(u'1995', 1)]
```

```
47
48 ### ANALYZE FAILED REQUESTS
49 ### Compute the percentage of failures overall
50 errorRdd = logRdd.filter(lambda x: x.status_code == "404")
51 print "The percentage of failures overall is %.3f%%" % (float(errorRdd.count()) / logRdd.count())
52 # The percentage of failures overall is 0.027%
53
54 ### Compute the empirical probability distribution of server failures (i.e. number of times a server fails
55 ### divided by the total number of failures)
56 s = float(errorRdd.count())
57 serverFailureRateRdd = (errorRdd
58                         .map(lambda x: (x.server_ip, 1))
59                         .reduceByKey(lambda a, b: a + b)
60                         .map(lambda x: (x[0], x[1] / s)))
61 print serverFailureRateRdd.collect()
62 # [(u'xyp74p20.ltec.net', 0.037037037037037035), (u'www-c2.proxy.aol.com', 0.037037037037037035),
63 # (u'net.auckland.ac.nz', 0.07407407407407407), (u'internet-gw.watson.ibm.com', 0.8518518518518519)]
64
65 ### COUNTRY ANALYSIS
66 ### parse the country dataset
67 countryPath = "data/countries.txt"
68 countryRawRdd = sc.textFile(countryPath)
69 serverCountriesRdd = countryRawRdd.map(lambda x: parse(x))
70 print serverCountriesRdd.take(5)
71 # [[u'134.75.200.38', u'IT'], [u'xyp74p20.ltec.net', u'US'], [u'192.239.48.10', u'UK'],
72 # [u'198.202.23.100', u'DE'], [u'198.174.185.20', u'JP']]
73
74 ### Compute the top 10 countries with the highest number of failures
75 serverFailuresRatesWithCountry = (serverFailureRateRdd
76                                   .join(serverCountriesRdd)
77                                   .map(lambda x: (x[0], x[1][1], x[1][0])))
78 countryFailuresPerServerRdd = serverFailuresRatesWithCountry.map(lambda x: (x[1],x[2]))
79 countryFailuresTotRdd = countryFailuresPerServerRdd.reduceByKey(lambda a, b: a + b)
80 top10FailingCountries = countryFailuresTotRdd.takeOrdered(<FILL_IN>)
81 print top10FailingCountries
82 # [(u'US', 0.9259259259259259), (u'IT', 0.07407407407407407)]
```

Question 30 Complete line 29.

A nentries = logRdd.collect()

B nentries = logRdd.map(lambda x: 1).reduceByKey(lambda a, b: a + b)

C nentries = logRdd.count()

D nentries = logRdd.count().collect()

SOLUTION:
nentries = logRdd.count()

Question 31 Is line 40 correct?

A No, it should have been *directoriesRdd = logRdd.map(lambda x: get_ dirs(x.request)).collect()*

B No, it should have been *directoriesRdd = logRdd.flatMap(lambda x: get_ dirs(x.request))*

C No, it should have been *directoriesRdd = logRdd.filter(lambda x: x in get_ dirs(x.request))*

D Yes. It is correct.

SOLUTION:
No, it should have been *directoriesRdd = logRdd.flatMap(lambda x: get_ dirs(x.request))*

Question 32 How many errors are there in lines form 50 to 70?

A No errors. B 3 C 2 D 1

SOLUTION:
2 errors. Line 50 should be *errorRdd = logRdd.filter(lambda x: x.status_code == 404)* and line 69 *serverCountriesRdd = countryRawRdd.map(lambda x: x.split("\t")*

Question 33 Is there a more efficient way to compute *serverFailureRateRdd* at line 57?

A
```
avgFailureRateRdd =
(errorRdd
.map(lambda x: (x.server_ip, 1))
.reduceByKey(lambda a, b: a + b)
.map(lambda x: (x[0], x[1] / s.cache())))
```

B
```
avgFailureRateRdd =
(errorRdd
.map(lambda x: (x.server_ip, 1))
.groupByKey(lambda a, b: a + b)
.map(lambda x: (x[0], x[1] / s)))
```

C No, it is a fairly optimal solution

D
```
avgFailureRateRdd =
(errorRdd
.map(lambda x: (x.server_ip, 1))
.reduceByKey(lambda a, b: a + b)
.map(lambda x: (x[0], x[1] / float(errorRdd.count())))
```

SOLUTION:
No, it is a fairly optimal solution (broadcast variables can be used to optimize further, but we didn't cover them during practices)

Question 34 Which line is variable *serverFailuresRatesWithCountry* (line 75) actually computed at?

A 81 B 79 C 75 D None of these lines.

SOLUTION:
None of these lines. *serverFailuresRatesWithCountry* gets computed at line 80 when the first action (takeOrdered) is applied to the RDD, due to the lazy evaluation mechanism of Apache Spark (reduceByKey is a transformation, not an action).

Question 35 Which of these is a possible output for *serverFailuresRatesWithCountry.first()* (line 75)? (Hint: look at previous *print* instructions to understand which is the content of the joined RDDs.)

A (u'net.auckland.ac.nz', 0.07407407407407407, u'IT')

B (u'net.auckland.ac.nz', (0.07407407407407407, u'IT'))

C (u'net.auckland.ac.nz', u'IT', 0.07407407407407407)

D (u'net.auckland.ac.nz', (u'IT', 0.07407407407407407))

SOLUTION:
(u'net.auckland.ac.nz', u'IT', 0.07407407407407407)

Question 36 **Reserve** - *Do not answer unless explicitly stated during the exam*
Complete line 80.

A *10, lambda x: -x[1]*

B *10*

C *10, lambda x: x[1]*

D *10, lambda x: x[0]*

SOLUTION:
10, 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

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

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

$^{\dagger}1$ 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.

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

Performance Evaluation

Question 07 : ☐A ☐B ☐C ☐D
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 **(R) :** ☐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