

Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Multi-stream application examples

Università degli studi di Roma Tor Vergata
Department of Civil Engineering and Computer Science Engineering

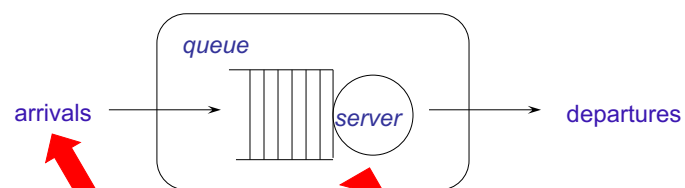
Copyright © Vittoria de Nitto Personè, 2021
<https://creativecommons.org/licenses/by-nc-nd/4.0/>



1

Multi-stream RNG
Case study ssq

Single Server Queue



1. one stream per stochastic component

2. uncoupled processes → no-overlapped streams

Prof. Vittoria de Nitto Personè

2

2

Quando uso il multistream (lo userò sempre) non uso la PutSeed, bensì la PlantSeeds, perchè devo 'piantarli' tutti.

Multi-stream RNG
Case study ssq

ssq2 revisited

- Use rngs functions for arrivals and services

```
double GetArrival(void) {
    static double arrival = START;
    SelectStream(0);
    arrival += Exponential(2.0);
    return (arrival);
}

double GetService(void) {
    SelectStream(2);
    return (Uniform(1.0, 2.0));
}
```

- include "rngs.h" and use PlantSeeds(12345)
(in place of PutSeed(12345))

Prof. Vittoria de Nitto Personè
3

3

arrival and service processes are *uncoupled*

stream 0 for arrivals, stream 1 for services

for 10025 jobs

average interarrival time =	1.99
average wait	3.92
average delay	2.41
average service time	1.50
average # in the node ...	1.96
average # in the queue ..	1.21
utilization	0.75

stream 0 for arrivals, stream 2 for services (or e.g. stream 128 to get more separation)

for 10025 jobs

average interarrival time =	1.99
average wait	3.86
average delay	2.36
average service time	1.50
average # in the node ...	1.93
average # in the queue ..	1.18
utilization	0.75

Theoretical values

\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}
2.00	3.83	2.33	1.50	1.92	1.17	0.75

(in riferimento all'esercizio con Unif(1,2))

Prof. Vittoria de Nitto Personè
4

4

Le cose cambiano, stessi arrivi, distribuzioni etc, ma sto prendendo due pezzi di numeri random diversi!
Ci sono piccole oscillazioni, ma sono normali!

Uncoupling Stochastic Processes

perchè non prenderne una $Unif(0,3)$?
ha un'altra varianza ancora, sarebbe un "altro caso".

Consider changing the service process to

$Uniform(0.0, 1.5) + Uniform(0.0, 1.5)$

La media è sempre la stessa (1.5), variabilità qui più ampia.
(perchè è come fosse un'unica uniforme da 0 a 3, sommando)

- Without uncoupling, arrival process sequence would change!
- With uncoupling, the service process "sees" exactly the same arrival sequence
- Important variance reduction technique

Se voglio vedere l'effetto della variabilità, devo toccare solo lei,
e devo tenere i flussi diversi. Non posso cambiare più cose insieme!

Prof. Vittoria de Nitto Personè

5

5

Theoretical values

\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}
2.00	3.83	2.33	1.50	1.92	1.17	0.75

stream 0 for arrivals, stream 1 for services

for 10025 jobs

average interarrival time = 1.99
average wait = 4.29
average delay = 2.78
average service time = 1.51
average # in the node ... = 2.15
average # in the queue .. = 1.40
utilization = 0.76

nuovi risultati, alcuni sono cambiati di
pochissimo (servizio medio),
altri maggiormente (avg wait).

risultati con uniform(1.0, 2.0)

for 10025 jobs
average interarrival time = 1.99
average wait = 3.92
average delay = 2.41
average service time = 1.50
average # in the node ... = 1.96
average # in the queue .. = 1.21
utilization = 0.75

Prof. Vittoria de Nitto Personè

6

6

ssq with Multiple Job Types

- Consider multiple job types, each with its own arrival and service process
- Two job types:
 - arrivi più frequenti service con stessa media, cambia varianza, in classe 1 maggiore.
 - Class 0: *Exponential(4.0)* interarrivals, *Uniform(1.0, 3.0)* service
 - Classe 1: *Exponential(6.0)* interarrivals, *Uniform(0.0, 4.0)* service
- Use `rngs` to allocate a different stream to each stochastic process

Prof. Vittoria de Nitto Personè

7

7

errore frequente: devo sequenziare istanti di arrivo di due classi. A seconda dell'istante di arrivo che stiamo 'trattando' dobbiamo simulare il successivo. Un errore sarebbe generare TUTTI INSIEME GLI ARRIVI in uno stesso momento.

E' un errore quindi generare subito e tutti insieme istanti di arrivo e tempi di servizio.

Arrival process

```
double GetArrival(int *j)      /* j corrisponde to job type */
{
  const double mean[2] = {4.0, 6.0};
  static double arrival[2] = {START, START};
  static int init = 1;
  double temp;
  if (init) {
    SelectStream(0);
    arrival[0] += Exponential(mean[0]);
    SelectStream(1);
    arrival[1] += Exponential(mean[1]);
    init = 0;
  }

  if (arrival[0] <= arrival[1]) {
    *j = 0;
    temp = arrival[0];
    SelectStream(*j);
    arrival[*j] += Exponential(mean[*j]);
    return (temp);
  }
  else {
    *j = 1;
    temp = arrival[1];
    SelectStream(*j);
    arrival[*j] += Exponential(mean[*j]);
    return (temp);
  }
}
```

li genero via via, infatti ora parto sono dai due START.

trovo quale è "venuto" prima

The first arrival is of class 0!

se sto consumando arrival 0, devo generare il prossimo arrivo.

- streams 0 and 1 are used for interarrival times of class 0 and class 1 jobs respectively

Prof. Vittoria de Nitto Personè

8

8

Ad ogni istante di tempo della simulazione, ho array con i prossimi istanti di arrivo. Come abbiamo detto prima, non li genero tutti, ma solo quando ne sto consumando uno!

Service process

```
double GetService(int j)    j = 0 or 1, è la classe.
{
    const double min[2] = {1.0, 0.0};
    const double max[2] = {3.0, 4.0};
    SelectStream(j + 2); mi sposto sullo stream 2 o 3, a seconda di j
    return (Uniform(min[j], max[j]));
}                        Unif[1,3] or Unif[0,4]
```

- j corrisponde to the job type (0 or 1)
- streams 2 and 3 are used for service times of class 0 and 1 respectively
- All four simulated stochastic processes are uncoupled!
- Any process could be changed without altering the random sequence of others!

Prof. Vittoria de Nitto Personè

9

9

Consistency checks

- The “teorethical” steady-state statistics are

\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}	
2.40	7.92	5.92	2.00	3.30	2.47	0.83	exact analytical results, No simulation!

- obvious consistency checks: $\bar{w} = \bar{d} + \bar{s}$ $\bar{l} = \bar{q} + \bar{x}$
- other consistency checks:
 - both job types have avg service time of 2.0 $\rightarrow \bar{s} = 2.00$
 - arrival rate should be $1/4 + 1/6 = 5/12 \rightarrow \bar{r} = 12/5 = 2.40$
 - \bar{x} should be ratio of arrival to service rates

$$\frac{5/12}{1/2} = 5/6 \approx 0.83$$

Prof. Vittoria de Nitto Personè

10

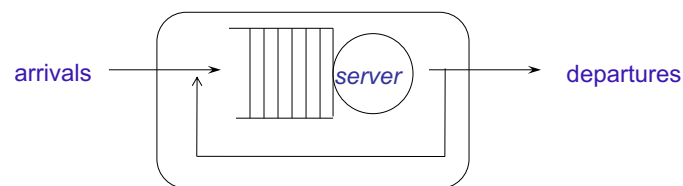
10

Exercises

- Exercises: 3.2.3, 3.2.4, 3.2.7

un job che arriva al sistema potrebbe richiedere 'servizi' ulteriori. Discrete-Event Simulation
Il completamento del servizio non è detto che corrisponda alla partenza. ssq with feedback
Qui una partenza corrisponde al completamento di un numero di servizi richiesti.
C'è quindi completamento singolo e totale.

Single Server Queue with feedback



- If the service a job receives is incomplete or unsatisfactory, the job feeds back
- Completion of service and departure now have different meanings

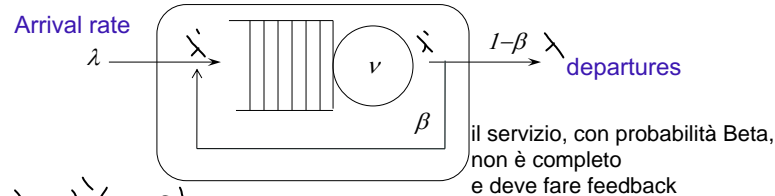
Non so a priori quante volte richiede feedback un job. E' probabilistico.

DE simulation
ssq with feedback

Model Considerations

"ni" quindi $1/\nu$ è tempo di servizio per UN SOLO passaggio.

ν service rate



$$\lambda = \lambda' (1 - \beta)$$

- When feedback occurs the job joins the queue consistent with the queue discipline
- The decision to depart or feed back is random with *feedback probability* β

Come calcolare l'utilizzazione? perchè non è più semplicemente λ/ν $\rho = \frac{\lambda'}{\nu}$

Prof. Vittoria de Nitto Personè

13

13

Discrete-Event Simulation
ssq with feedback

Model Considerations

- Feedback is independent of past history
- In theory, a job may feed back arbitrarily many times
- Jobs from outside the system are merged with jobs from the feedback process
- The steady-state request-for-service rate is larger than λ by the positive additive factor $\beta \bar{x} \nu$
- Note that \bar{x} increases with feedback but $1/\nu$ is the average service time per request

Prof. Vittoria de Nitto Personè

14

14

Flow Balance and Saturation

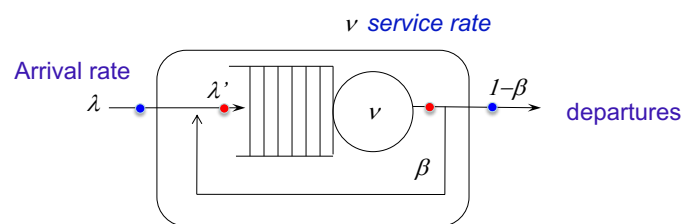
- Jobs flow into the service node at the average rate of λ
- To remain flow balanced jobs must flow out of the service node at the same average rate
- The average rate at which jobs flow out of the service node is $\bar{x}(1-\beta)v$
- Flow balance is achieved when $\lambda = \bar{x}(1-\beta)v$
- Saturation occurs when $\bar{x}=1$ or as $\beta \rightarrow 1 - \lambda/v$

Prof. Vittoria de Nitto Personè

15

15

Coda con feedback



Flow Balance: $\lambda = \lambda' (1-\beta)$
 $\lambda' = \lambda / (1-\beta)$

Saturation: $\rho \rightarrow 1$
 $\lambda / (1-\beta) v \rightarrow 1$
 $\beta \rightarrow 1 - \lambda / v$

Prof. Vittoria de Nitto Personè

16

16

- Feedback is independent of past history
- In theory, a job may feed back arbitrarily many times

```
int GetFeedback(double beta)    /* 0.0 <=  $\beta$  < 1.0 */
{
    SelectStream(2);
    if (Random() < beta)
        return (1);            /* feedback */
    else
        return (0);            /* no feedback */
}
```

Statistical considerations

- Index $i=1, 2, 3, \dots$ counts jobs that enter the service node
 - fed-back jobs are not recounted
- Using this indexing, all job-averaged statistics remain valid
- We must update **delay times**, **wait times** and **service times** for each feedback
- Jobs from outside the system are merged with jobs from the feedback process
- The steady-state request-for-service rate is larger than λ by the positive additive factor $\beta \bar{x} \nu$
- Note that \bar{x} increases with feedback but $1/\nu$ is the average service time per request

Algorithm and Data Structure Considerations

job index	1	2
Arrival/feedback	1	3
service	9	3
departure	10	13

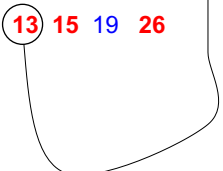
Prof. Vittoria de Nitto Personè

19

19

Algorithm and Data Structure Considerations

job index	1	2	3	4	5	.	6
Arrival/feedback	1	3	4	7	10		14
service	9	3	2	4	7		6
departure	10	13	15	19	26		37



Prof. Vittoria de Nitto Personè

20

20

Algorithm and Data Structure Considerations

job index	1	2	3	4	5	6	7	8	9	...			
Arrival/feedback	1	3	4	7	10	13	14	15	19	24	26	30	...
service	9	3	2	4	7	5	6	3	4	6	3	7	...
departure	10	13	15	19	26	31	37	40	44	50	53	60	...

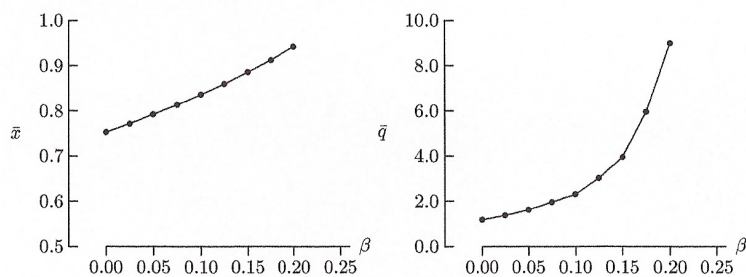
At the computational level, some algorithm and data structure is necessary

21

Theoretical values						
\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}
2.00	3.83	2.33	1.50	1.92	1.17	0.75

Program ssq2 was modified to incorporate immediate feedback

- interarrivals = *Exponential*(2.0)
- service times = *Uniform*(1.0, 2.0)

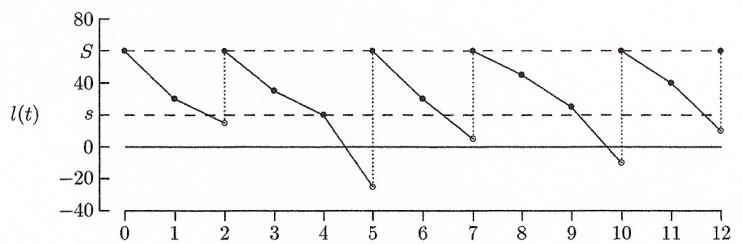


- It appears saturation is achieved as $\beta \rightarrow 0.25$

22

Inventory system with delivery lag

- *delivery lag* (dl) occurs when orders are not delivered immediately
- Lag is assumed to be random and independent of order size
- Without lag, inventory jumps occur only at inventory review times



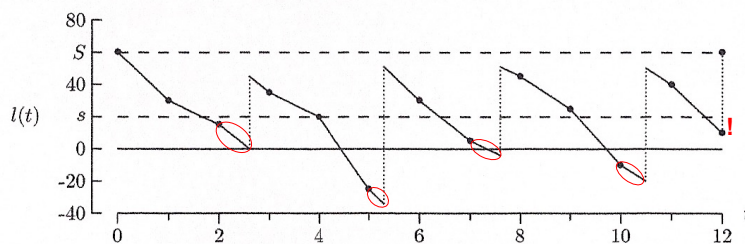
Prof. Vittoria de Nitto Personè

23

23

Inventory system with delivery lag

- With delivery lag, inventory jumps occur at arbitrary times



- The last order is assumed to have no lag
- We assume that orders are delivered before the next inventory review
- With this assumption, there is no change to the specification model

Prof. Vittoria de Nitto Personè

24

24

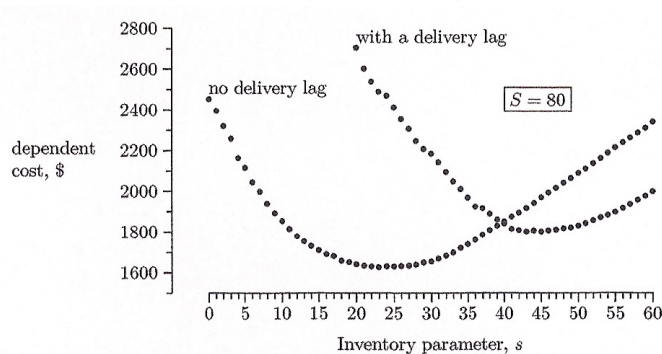
- For the InvSys with delivery lag, $\delta_i = 0.0$ iff no order during i^{th} interval, $0 < \delta_i < 1.0$ otherwise
- The InvSys is *lag-free* iff $\delta_i = 0.0$ for all i
- If (S, s) are fixed then, even with small dl:
 - $\bar{o}, \bar{d}, \bar{u}$ are the same regardless of delivery lag
 - Compared to the lag-free system, \bar{l}_i^+ will decrease
- Compared to the lag-free system, \bar{l}_i^- will increase or remain unchanged

Prof. Vittoria de Nitto Personè

27

27

- Delivery lags are independent *Uniform*(0.0, 1.0) random variates

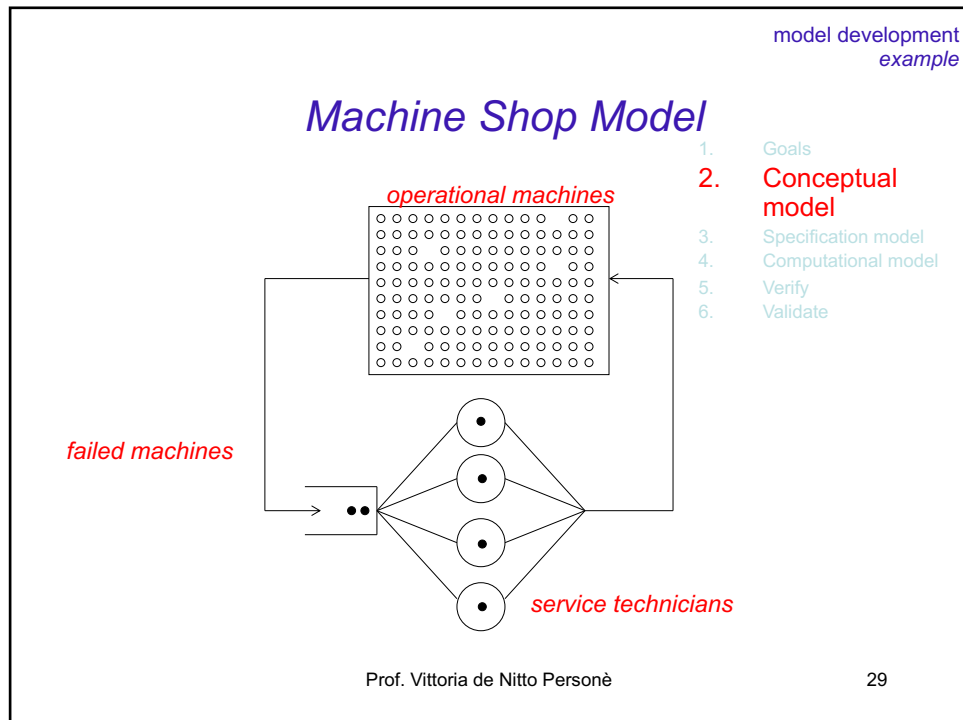


- delivery lag causes \bar{l}_i^+ to decrease and \bar{l}_i^- to increase or remain the same
- with $C_{\text{hold}} = \$25$ and $C_{\text{short}} = \$700$, cause shift up and to the left

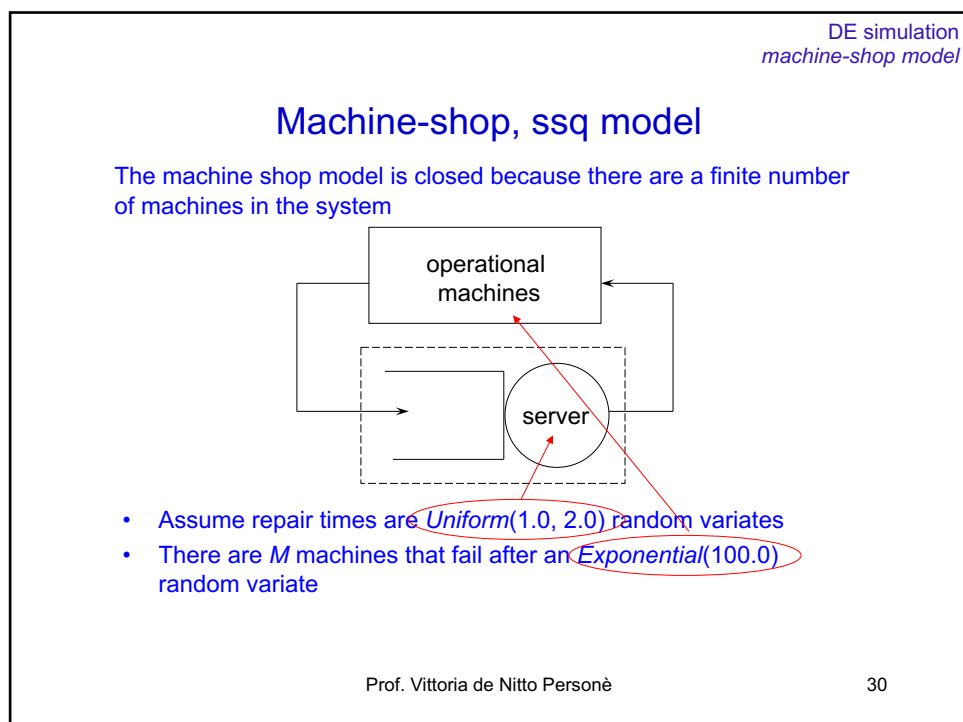
Prof. Vittoria de Nitto Personè

28

28



29



30

Program ssms

- program `ssms` simulates a Single Server Machine-Shop
- the library `rngs` is used to uncouple the random process
- the failure process is defined by the array `failures`
 - a $O(M)$ search is used to find the next failure
 - alternate data structures can be used to increase computational efficiency

Prof. Vittoria de Nitto Personè

31

31

```
double GetFailure(void)
{
    SelectStream(0);
    return (Exponential(100.0));
}

double NextFailure(double failure[], int *m)
{
    int i = 0;
    double t = failure[0];

    *m = i;
    for (i = 1; i < M; i++)
        if (failure[i] < t) {
            t = failure[i];
            *m = i;
        }
    return (t);
}

double GetService(void)
{
    SelectStream(1);
    return (Uniform(1.0, 2.0));
}
```

Prof. Vittoria de Nitto Personè

32

32


```

int main(void)
{
    long    index    = 0;      /* job (machine failure) index */
    double  arrival   = START; /* time of arrival (failure) */
    double  delay;          /* delay in repair queue */
    double  service;        /* service (repair) time */
    double  wait;           /* delay + service */
    double  departure = START; /* time of service completion */
    int     m;              /* machine index 0,1,...(M-1) */
    double  failure[M];      /* list of next failure times */
    struct {                /* sum of ... */
        double wait;        /* wait times */
        double delay;       /* delay times */
        double service;     /* service times */
        double interarrival; /* interarrival times */
    } sum = {0.0, 0.0, 0.0};
}

```

Prof. Vittoria de Nitto Personè

33

33

```

PlantSeeds(123456789);

for (m = 0; m < M; m++)          /* initial failures */
    failure[m] = START + GetFailure();

while (index < LAST) {
    index++;
    arrival = NextFailure(failure, &m);
    if (arrival < departure)
        delay = departure - arrival;
    else
        delay = 0.0;
    service = GetService();
    wait = delay + service;
    departure = arrival + wait;          /* completion of service */
    failure[m] = departure + GetFailure(); /* next failure, machine m */
    sum.wait += wait;
    sum.delay += delay;
    sum.service += service;
}
sum.interarrival = arrival - START;

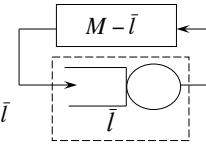
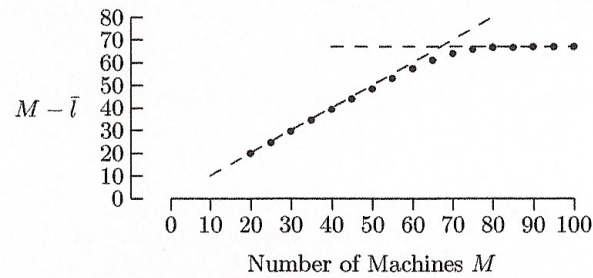
```

Prof. Vittoria de Nitto Personè

34

34

- The time-averaged number of working machines is $M - \bar{l}$



- For small values of M the time-averaged number of operational machines is essentially M
- For large values of M this value is essentially constant at approximately 67

Prof. Vittoria de Nitto Personè

35

35

Exercises

- Exercises: 3.3.2, 3.3.3, 3.3.4, 3.3.7

Prof. Vittoria de Nitto Personè

36

36