



TP2 - Simulación

PICCO, MARTÍN ALEJANDRO - 99289

RIPARI, SEBASTIAN DANIEL - 96453

NOCETTI, TOMAS AGUSTIN - 100853

DANERI, ALEJANDRO NICOLÁS -97839

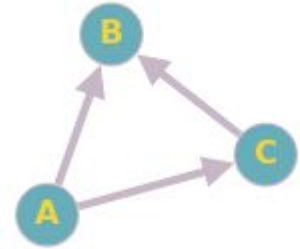


Page Rank

- Comenzando con cualquier vector de probabilidades
- Encontramos el autovector principal de la matriz realizando multiplicaciones de la matriz por el vector de probabilidades
- El resultado converge a la probabilidad final de cada nodo dando como resultado el PageRank

Dead Ends

- No se puede salir de “B”
- El PageRank de B va a ser 1
- Y el de “A” y “C” 0



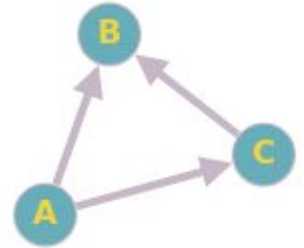
Dead Ends

Convertimos la matriz en estocástica

Hyperlink Matrix H			
	A	B	C
A			
B	1/2		1
C	1/2		

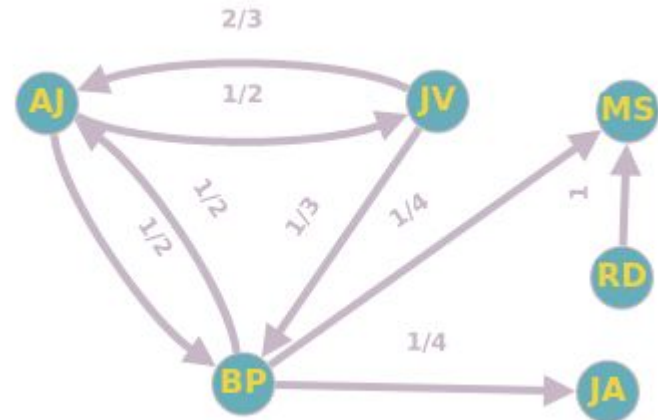
+

Dangling Nodes Matrix A			
	A	B	C
A		1/3	
B		1/3	
C		1/3	



Calculando Page Rank

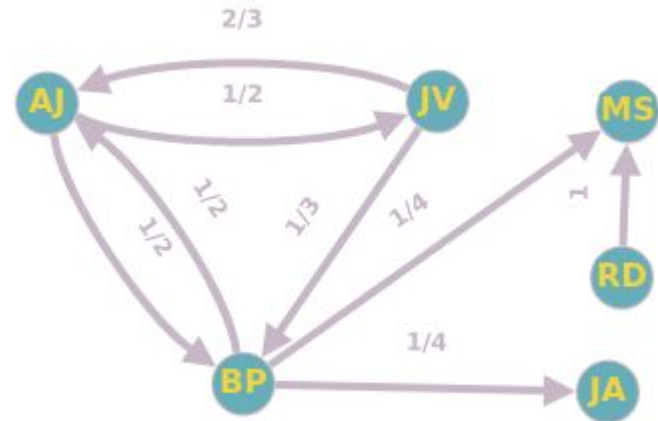
0	0	1	0	0	0
1/5	1/5	1/5	1/5	1/5	1/5
1/5	1/5	1/5	1/5	1/5	1/5
0	0	0	0	2/3	1/3
0	0	0	1/2	0	1/2
0	1/4	1/4	0	1/2	0



Calculando Page Rank (continuación)

Empezamos por cualquier nodo. Por ejemplo

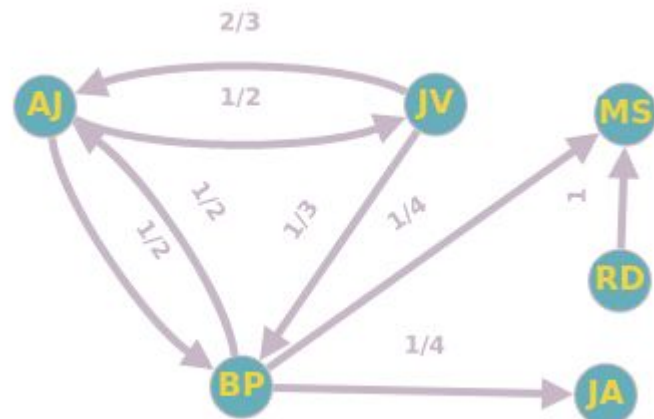
0 0 1 0 0 0



Resultados

Page Rank

angelinajolie	0.28571428571428525
bradpitt	0.24489795918367308
martinscorces	0.14285714285714263
robertdeniro	0.04081632653061218
jenniferanist	0.10204081632653045
jonvoight	0.1836734693877548



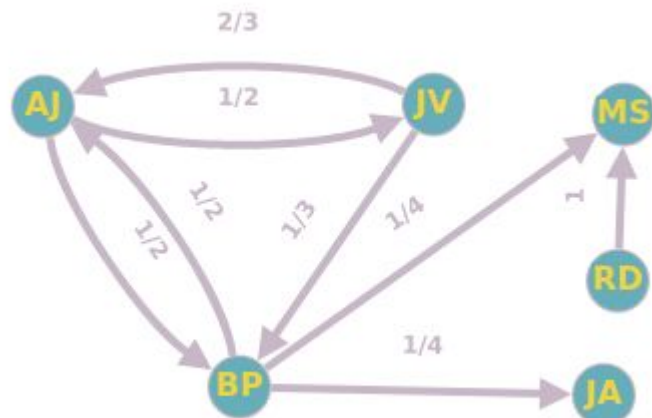


Buscando palabras

```
def calculate_scores(word):  
    scores=dict()  
    for key in list(indexes.keys()):  
        if indexes[key].get(word) is None:  
            # si no encuentra la palabra directamente le pone puntaje 0  
            scores[key] = 0  
        else:  
            # ponderacion basica entre aparicion y page rank  
            scores[key]=0.5* indexes[key].get(word) + 0.5*page_rank[key]
```

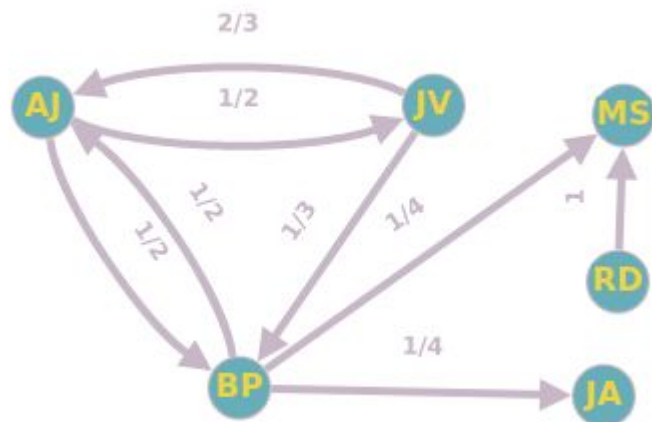

Busco “film”

[('martinscorcese', 3.071428571428571),
(('angelinajolie', 1.6428571428571426),
(('bradpitt', 1.6224489795918364),
(('jonvoight', 0.5918367346938774),
(('jenniferaniston', 0.5510204081632653),
(('robertdeniro', 0.5204081632653061)]



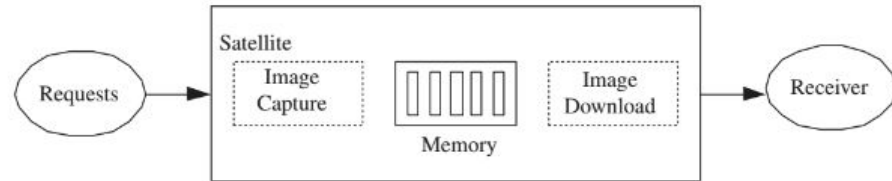
Busco “actor”

[('jonvoight', 2.0918367346938775),
(('robertdeniro', 1.5204081632653061),
(('bradpitt', 1.1224489795918364),
(('martinscorcese', 1.0714285714285714),
(('angelinajolie', 0.6428571428571426),
(('jenniferaniston', 0)]

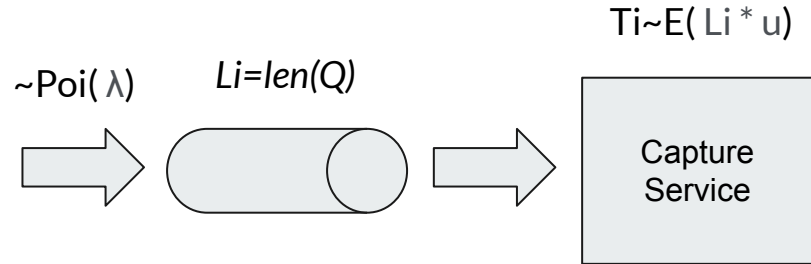
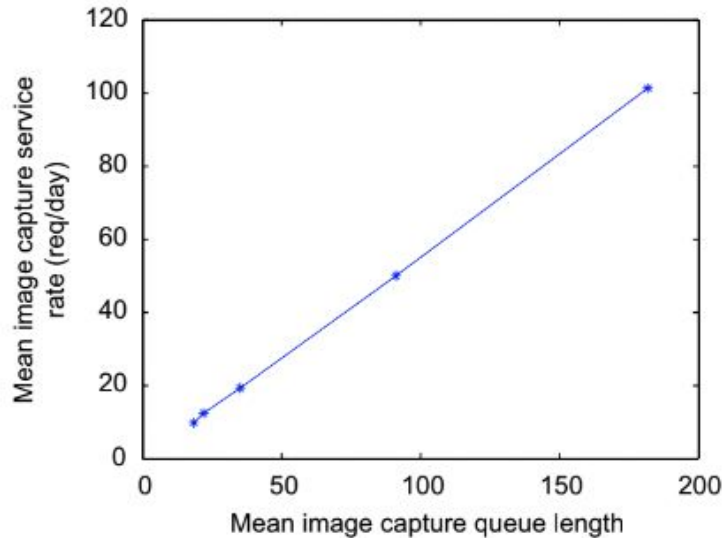


Punto 2 - Paper repaso

- Generar un modelo basado en colas para los servicios provistos por satélites.
- 2 servicios: **captura** y descarga
- Para el servicio de captura:
 - Política FOFS (First Opportunity, First served)
 - Asumimos uniformidad en el `target` de los pedidos.
 - Llegadas con distribución Poisson
 - Asumimos colas infinitas.

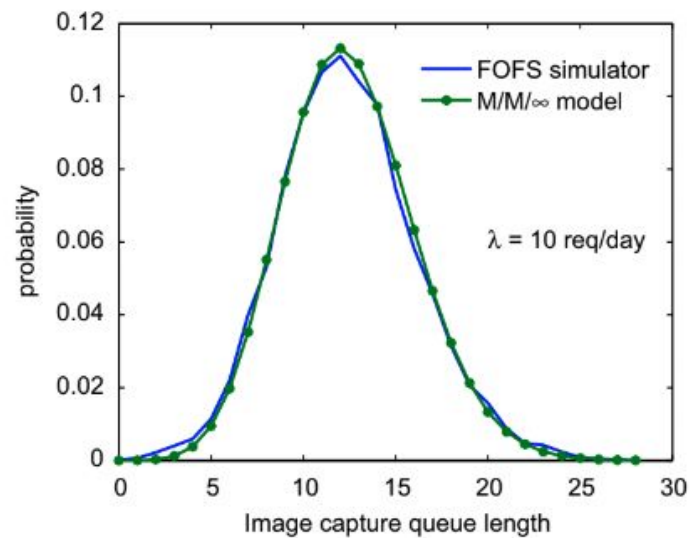
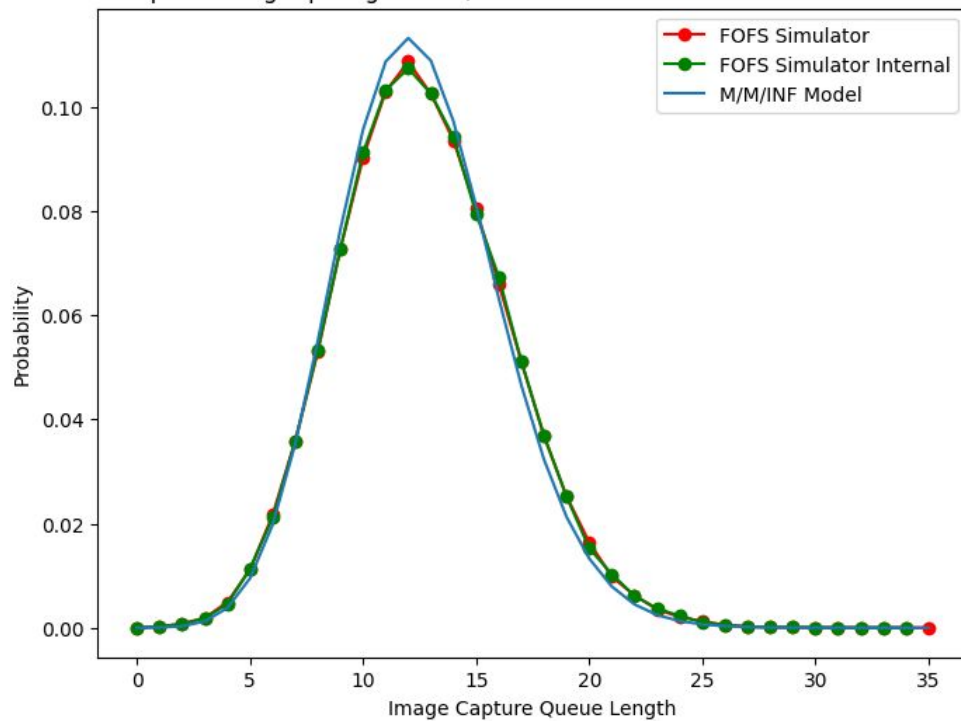


Punto 2 - Modelo propuesto por el paper



$$p_n = \frac{(\lambda/\mu_0)^n e^{-\lambda/\mu_0}}{n!} \quad (n \geq 0).$$

Capture image queuing modal / Arrival MU = 10 - Service Base MU - 0.8



Tamaño promedio de la cola

$$Li = \frac{\lambda}{\mu}$$

```
In [6]: val = REQ_ARRIVAL_MU/PROC_BASE_MU
print('Tamaño de la muestra %d' % N_REQUESTS)

print('Resultado esperado %f' % val)
print('Resultado obtenido %f' % np.average(external_sampling))
```

```
Tamaño de la muestra 100000
Resultado esperado 12.500000
Resultado obtenido 12.657853
```

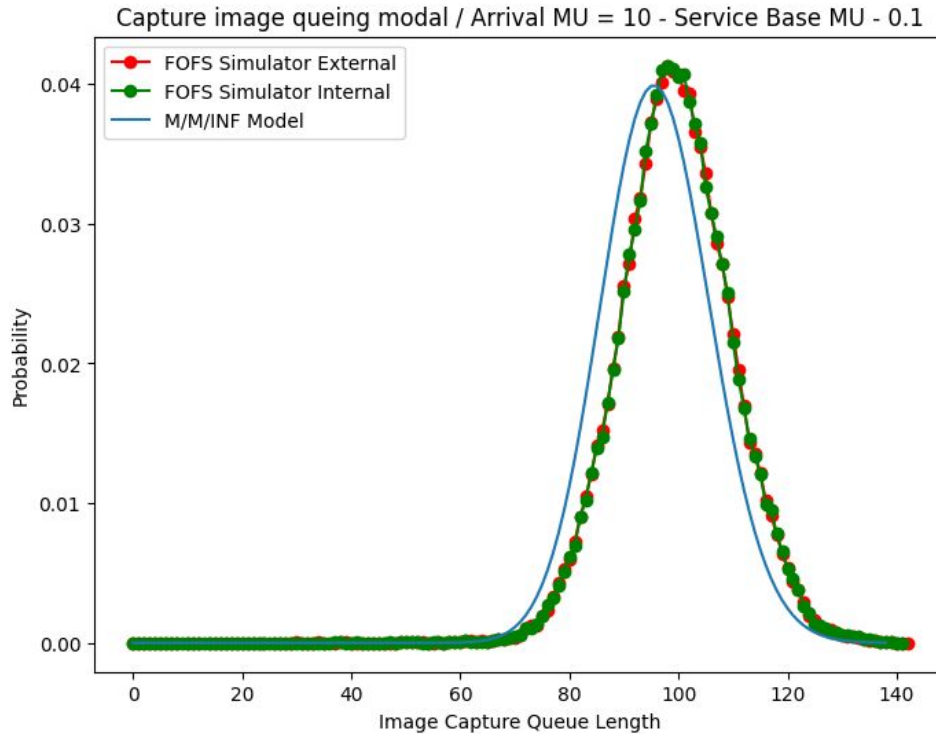
Tiempo promedio de espera

$$W = \frac{Li}{\lambda}$$

```
In [7]: val = REQ_ARRIVAL_MU/PROC_BASE_MU
print('Tamaño de la muestra %d' % N_REQUESTS)

print('Resultado esperado %f' % (1 / PROC_BASE_MU))
print('Resultado obtenido %f' % np.average(processing_times))
```

```
Tamaño de la muestra 100000
Resultado esperado 1.250000
Resultado obtenido 1.362575
```



Tamaño medio de la cola

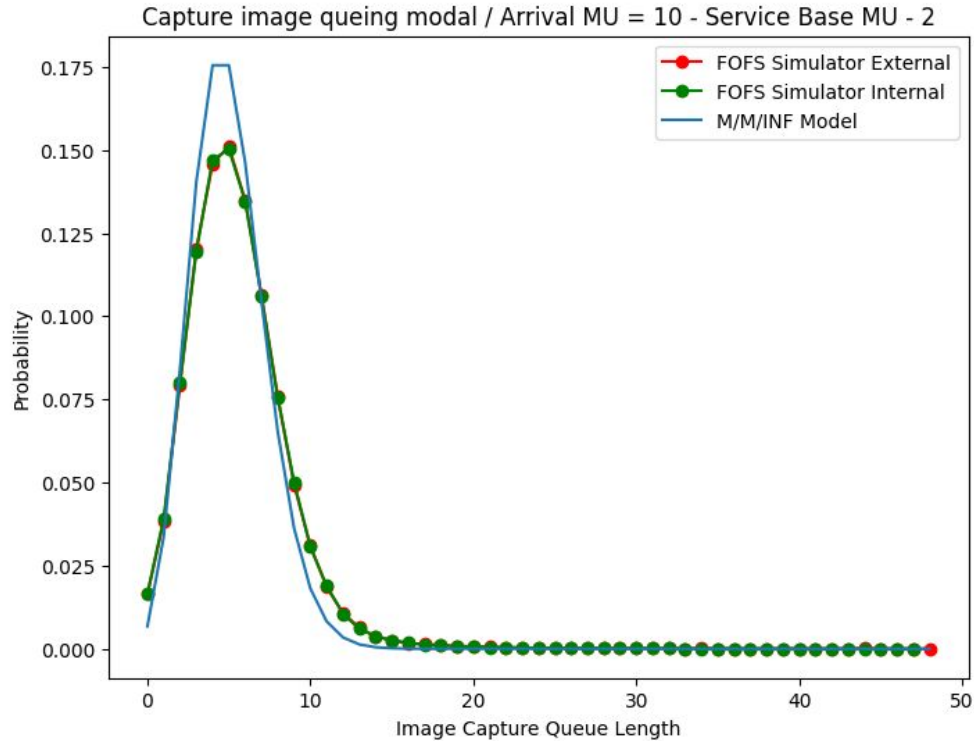
Resultado esperado: 100.000000

Resultado obtenido: 99.482397

Tiempo medio

Resultado esperado de tiempo promedio: 10.000000

Resultado obtenido de tiempo promedio: 10.072087



Tamaño medio de la cola

Tamaño medio de la cola esperado: **5.000000**

Tamaño medio obtenido: **5.515734**

Tiempo medio

Tiempo medio por req esperado: **0.500000**

Tiempo medio por req obtenido: **0.651000**

Punto 2 - Anexo código

```
class Satellite(object):
    """
    The Satellite handles each request with an exponential wait that depends on the number of Queued items.
    """


    def __init__(self, env):
        self.env = env
        self.machine = simpy.Resource(env)
        self.elements_in_queue = []
        self.closed = False

    def process_req(self):
        q_size = len(self.machine.queue)

        self.elements_in_queue.append(q_size)
        req_processing_time = random.exponential(1 / (PROC_BASE_MU * (q_size + 1) ))
        yield self.env.timeout(req_processing_time)

    def get_internal_sample(self):
        return self.elements_in_queue

    def close(self):
        self.closed = True
```

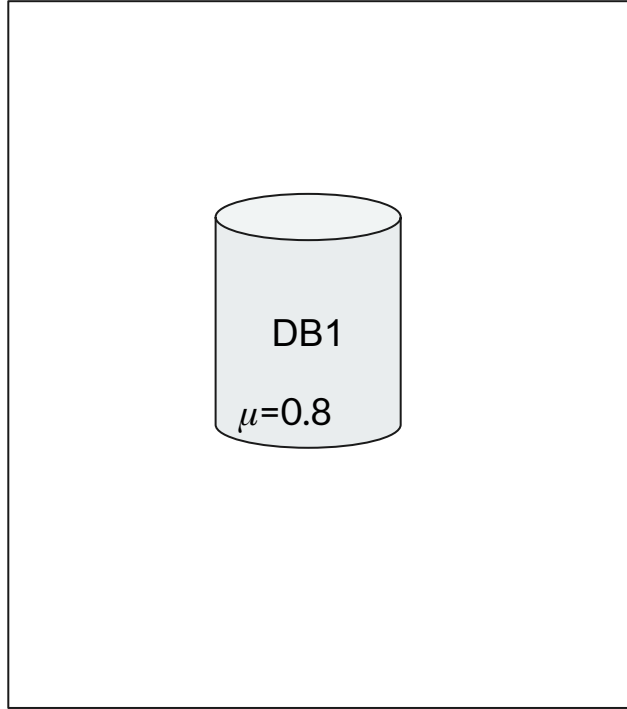


Punto 3 - WebService

M/M/1

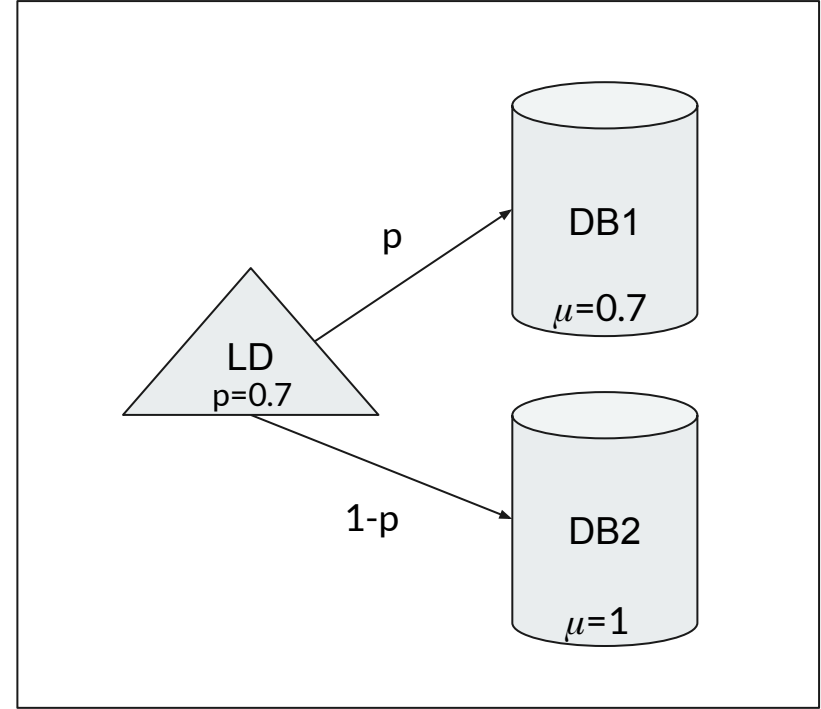
$\lambda=1/4$

M/M/2



SERVER 1

VS



SERVER 2

Implementación con Simpy | Server 1

- $n = 100000$
- Arribos
- Simpy | Timeout
- Numpy | Random

```
def start(self):  
    for i in range(n):  
        self.env.process(self.process_request()) # proceso request  
        yield self.env.timeout(random.exponential(self.media_requests)) # delay tiempo de arribo
```

- Servicio
- Simpy | Resource

```
class BaseDeDatos:  
    def __init__(self, env, mu):  
        self.mu = mu  
        self.resource = simpy.Resource(env, 1)
```

Implementación con Simpy | Server 1

- Simpy | Resource Request
- Seteo de timestamps




```
def process_request(self):  
    request = Request(self.env.now) # set start  
    request.set_espero(False if self.base_de_datos.resource.count == 0 else True)  
  
    with self.base_de_datos.resource.request() as req:  
        results = yield req # delay hasta que servicio se libera  
        request.set_servicio_comienzo(self.env.now) # set comienzo de servicio  
        yield self.env.timeout(random.exponential(self.base_de_datos.media)) # delay tiempo de servicio  
        request.set_servicio_termino(self.env.now) # set termino servicio  
        self.requests.append(request)
```

Implementación con Simpy | Server 2

- Bifurcación
- Numpy | Uniform

```
def start(self):  
    for i in range(n):  
        # bifurcacion a que base de datos voy  
        x = random.uniform(low=0.0, high=1.0, size=None)  
        i = 0 if x < self.p else 1  
        self.env.process(self.process_request(self.bases_de_datos[i])) # proceso request  
        yield self.env.timeout(random.exponential(self.media_requests)) # delay tiempo de arribo
```

Resultados

	Server 1 (1DB)	Server 2 (2DB)
Media tiempo hasta ser atendida	0.194 segundos	0.080 segundos 
Media tiempo total en el sistema	0.989 segundos	0.780 segundos 
Fracción que no espero	0.800	0.897 

Punto 4 - Problema

- Dur. entre arribos $\sim \text{Exp}(1/10 \text{ min})$
- Dur. de retiro $\sim \text{Exp}(1/1.5 \text{ min})$
- Cant. de retiro $\sim U(3, 50)$
- Dur. de depósito $\sim \text{Exp}(1/1.5 \text{ min})$
- Cant. de retiro $\sim U(10, 110)$
- Un único tipo de billete
- 75% retira, 25% deposita

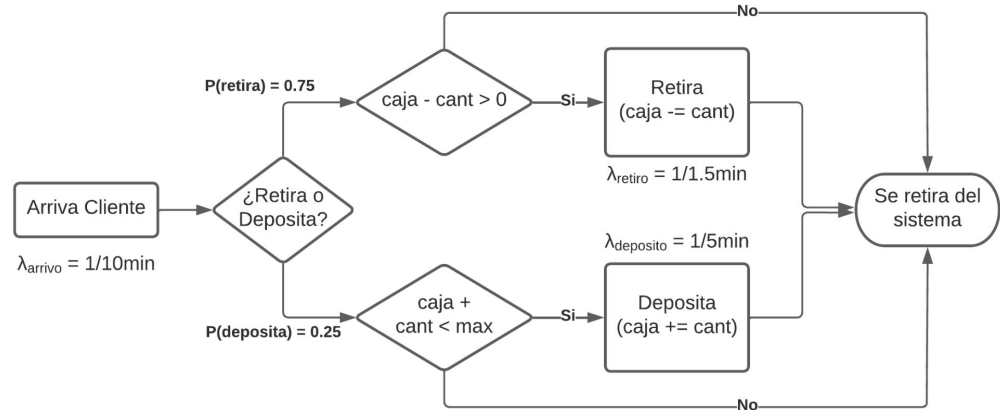


Fig: diagrama de flujo



Punto 4 - Herramientas

- Modelado de la situación y simulación con Simpy.
 - Provee un entorno por el cual se modela la ocurrencia de eventos concretados tras un *delay*
 - Los *delays* de los eventos emitidos “incrementan” el tiempo hacia el futuro
 - Se modelan como eventos el arribo del cliente, el uso del cajero, y la operación a realizar (retiro/depósito)
 - El cajero se modela como un Recurso/Canal de servicio de capacidad 1.
El primer cliente en llegar lo adquiere, y los siguientes esperan a que se libere
- Uso de scipy para simular los tiempos de *delay* de los eventos
- matplotlib para graficar resultados



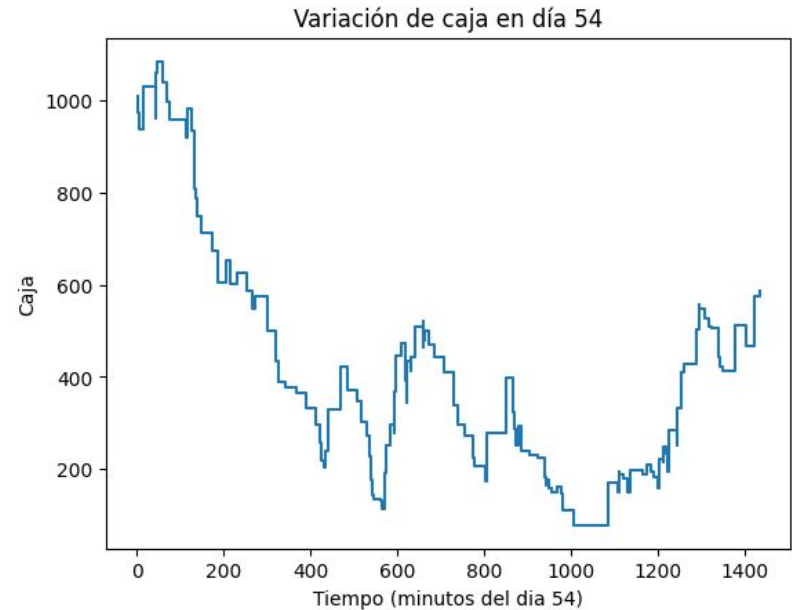
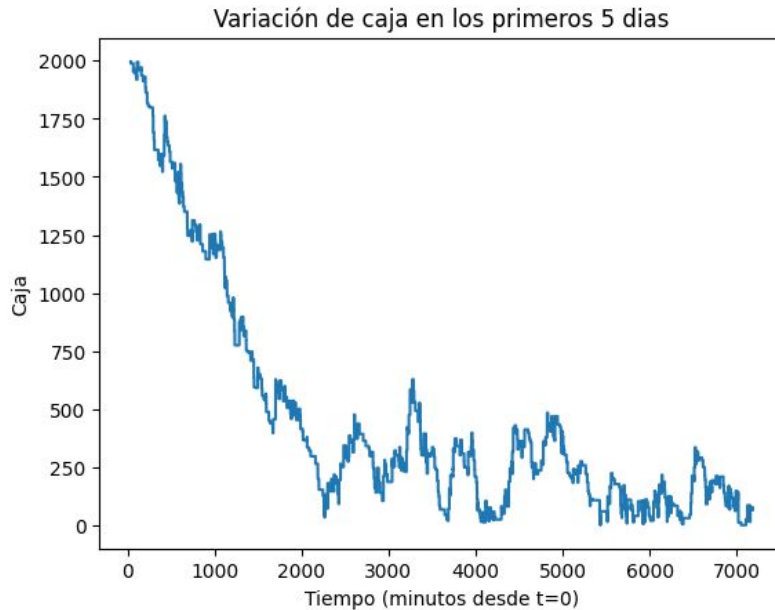
Punto 4 - Pseudocódigo

```
def simulacion:
    generar tiempo de arribo t_arr de dist. Exp(1/10)
    emitir evento: arribo con delay t_arr

    generar random prob_retira
    si prob_retira < 0.75:
        adquirir el recurso, o esperar si está ocupado

    si caja - cant > 0:
        caja -= cant
        generar tiempo de retiro t_ret de dist. Exp(1/1.5)
        emitir evento: retiro con delay t_ret
    si caja + cant <= MAX:
        caja += cant
        generar tiempo de deposito t_dep de dist. Exp(1/5)
        emitir evento: deposito con delay t_dep
```

Punto 4 - Resultados: Variación de Caja





Punto 4 - Resultados: Tiempos y Abandonos

- Parte de los datos recolectados abarcan el tiempo en sistema (cola + operación), y la cantidad de abandonos
- El tiempo promedio de un cliente en el sistema es de **3.36 minutos**
- La tasa de abandonos es de **14.72%**
 - La simulación sugiere que el cambio de cajero resulta en una disminución en la tasa de abandono
 - La tasa de abandono en las simulaciones realizadas ronda los 13%-15%

Punto 4 - Anexo: Código pt. 1

```
def arrivos_clientes(env, atm):
    global caja, tot_arrivos
    caja = CAP_MAX

    nro_cl = 0
    while True:
        # generamos un tpo de arribo exponencial
        tpo_arribo = expon.rvs(scale=MEDIA_ARRIVO, size=1)[0]
        yield env.timeout(tpo_arribo)
        nro_cl += 1

    retira_dinero = uniform.rvs() < PROB_RET

    if retira_dinero:
        env.process(ret(env, atm, nro_cl))
    else:
        env.process(dep(env, atm, nro_cl))

    tot_arrivos = nro_cl
```

```
def ret(env, atm, nro_cl):
    cantidad = randint.rvs(MIN_RET, MAX_RET, size=1)[0]
    condicion = lambda caja, cant: caja - cant >= 0
    operacion = lambda caja, cant: caja - cant
    return operar(env, atm, nro_cl, MEDIA_RET, cantidad, condicion, operacion)

def dep(env, atm, nro_cl):
    cantidad = randint.rvs(MIN_DEP, MAX_DEP, size=1)[0]
    condicion = lambda caja, cant: caja + cant < CAP_MAX
    operacion = lambda caja, cant: caja + cant
    return operar(env, atm, nro_cl, MEDIA_DEP, cantidad, condicion, operacion)
```



Punto 4 - Anexo: Código pt. 2

```
def operar(env, atm, nro_cl, media_op, cantidad, condicion, operacion):
    global caja, tot_abandonos, txs
    with atm.request() as req:
        tpo_entra_queue = env.now
        yield req;
        tpo_sale_queue = env.now
        tpo_espera = tpo_sale_queue - tpo_entra_queue

        tpo_op = expon.rvs(scale=media_op, size=1)[0]

        if condicion(caja, cantidad):
            caja = operacion(caja, cantidad)
            yield env.timeout(tpo_op)

            txs = np.vstack((txs, [caja, env.now, floor(env.now // MINUTOS_DIA)]))
        else:
            # no se da la condición para poder operar, abandona
            tot_abandonos += 1

    tpos_en_sist.append(tpo_espera + tpo_op)
```

¿Preguntas?

Gracias !