Simulación y Análisis de Sistemas de Reparación

Eduardo Brito Labrada
Grupo: C-311
Telegram: @eblabrada

1 Introducción

1.1 Descripción del Proyecto

El propósito de este proyecto es simular el comportamiento de un sistema de reparación que requiere mantener operativas n máquinas en todo momento. Para mitigar los efectos de posibles fallos, se cuenta con s máquinas adicionales que actúan como repuestos (spares). El sistema está diseñado de manera que, cada vez que una máquina falla, se sustituye inmediatamente por una máquina de repuesto disponible. La máquina que falló se envía a un taller de reparación donde un único reparador trabaja en ellas de forma secuencial, una a una.

El sistema se considera operativo mientras haya al menos una máquina de repuesto disponible para reemplazar las máquinas que fallen. El sistema *colapsa* o *falla completamente* en el momento en que una máquina en uso falla y no hay repuestos disponibles para su sustitución inmediata.

Inicialmente, hay $n+s$ máquinas funcionales: n de ellas se ponen en operación y s se mantienen como repuestos. El objetivo de la simulación es estimar el valor esperado del tiempo T hasta que el sistema colapsa, es decir, cuando el número de máquinas dañadas supera al número de repuestos disponibles.

Este ejemplo es tomado del libro "Simulation, Fifth Edition by Sheldon M. Ross", ejemplo 7.7.

2 Detalles de Implementación

La implementación del sistema de simulación se realizó mediante un enfoque de simulación basada en eventos discretos, utilizando una cola de prioridad para manejar eventos futuros y una lógica centrada en el ciclo de fallos y reparaciones de componentes.

Se define la clase `RepairSystem`, la cual recibe como parámetros el número de componentes operativos (`n_operating`), repuestos disponibles (`n_spares`), una función generadora de tiempo hasta fallo, y una función generadora de tiempo de reparación. También puede capturar el estado del sistema a lo largo del tiempo, si se activa el parámetro `capture_states`.

Listing 1: Constructor de la clase RepairSystem

```
class RepairSystem:
    def __init__(self, n_operating, n_spares, time_to_breakdown,
        time_to_repair, capture_states=False):
```

El sistema maneja dos tipos de eventos: `BREAKDOWN` (fallo de componente) y `REPAIR_COMPLETE` (finaliza la reparación de un componente). Estos eventos se almacenan en una cola de prioridad, de manera que siempre se procesa el evento más próximo en el tiempo.

Cada componente comienza operando, y se programan sus respectivos tiempos hasta fallo al inicio. Cuando un componente falla, si aún hay repuestos disponibles, se sustituye de inmediato y se programa un nuevo fallo. El componente dañado pasa a la cola de reparación. Si no hay reparación en curso, esta comienza inmediatamente. Si ya hay una reparación en proceso, el componente espera en la cola de reparación interna.

Listing 2: Clase Event y manejo de eventos

```
class EventType(Enum):
    BREAKDOWN = "breakdown"
    REPAIR_COMPLETE = "repair_complete"

class Event:
    def __init__(self, time: float, event_type: EventType):
        self.time = time
        self.event_type = event_type
    def __lt__(self, other):
        return self.time < other.time
```

El método principal `run()` ejecuta la simulación hasta que el número de componentes dañados supera el número de repuestos. En ese momento se considera que el sistema ha fallado por completo, y se devuelve el tiempo en que ocurre este evento crítico.

Listing 3: Ejecución del sistema

```
def run(self):
    self.schedule_initial_events()
    while True:
        event = heapq.heappop(self.event_queue)
        self.current_time = event.time
        if event.event_type == EventType.BREAKDOWN:
            self.handle_breakdown()
        else:
            self.handle_repair_complete()
        if self.down > self.n_spares:
            break
    return self.current_time
```

Se definieron las siguientes clases clave para representar el modelo:

- `ModelConfig`: encapsula la configuración del sistema con el número de servidores en operación y repuestos, las funciones generadoras de tiempos de fallo y reparación, y una etiqueta descriptiva.
- `CrashTimeAnalysis`: almacena los resultados de simulaciones, incluyendo promedio, desviación estándar, error y tamaño de muestra, y permite una representación legible de estos.

Las funciones generadoras de tiempos de fallo y reparación se modelan como variables aleatorias exponenciales con tasas $\lambda = 1/100$ (fallo) y $\mu = 1/20$ (reparación), respectivamente.

Listing 4: Definición de configuración del modelo

```
class ModelConfig:
    def __init__(self, n_operating, n_spares, breakdown_gen, repair_gen,
        label):
        self.n_operating = n_operating
```

```
self.n_spares = n_spares
self.breakdown_gen = breakdown_gen
self.repair_gen = repair_gen
self.label = label
```

2.1 Simulación

La simulación avanza generando tiempos de fallos y reparaciones para los servidores en operación y en repuesto, siguiendo un esquema de reemplazo inmediato cuando un servidor falla. El proceso termina cuando el sistema no tiene servidores operativos disponibles, momento en que se registra el tiempo total hasta el fallo.

Se realizaron múltiples corridas para obtener distribuciones estadísticas del tiempo hasta el fallo total, con un criterio de parada basado en la precisión del intervalo de confianza del 95%.

3 Resultados y Experimentos

Se realizaron simulaciones para diferentes configuraciones de servidores:

- Modelo 1: 5 servidores en operación y 2 repuestos.
- Modelo 2: 5 servidores en operación y 3 repuestos.

La simulación generó distribuciones de tiempos hasta fallo total que permitieron comparar la robustez del sistema según el número de repuestos disponibles.

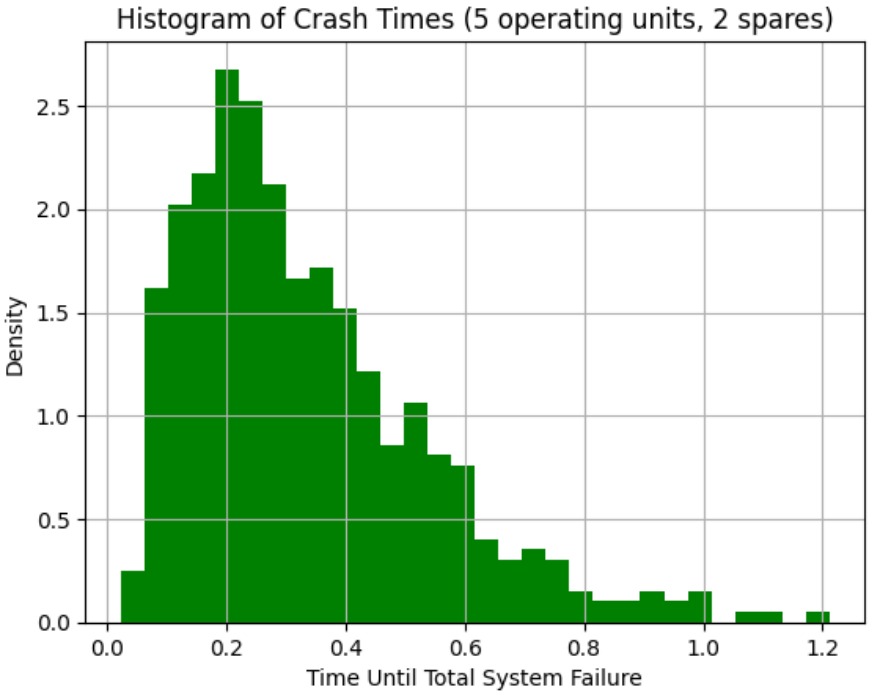

Figure 1: Distribución de tiempos hasta fallo total para Modelo 1 (5 operativos, 2 repuestos).

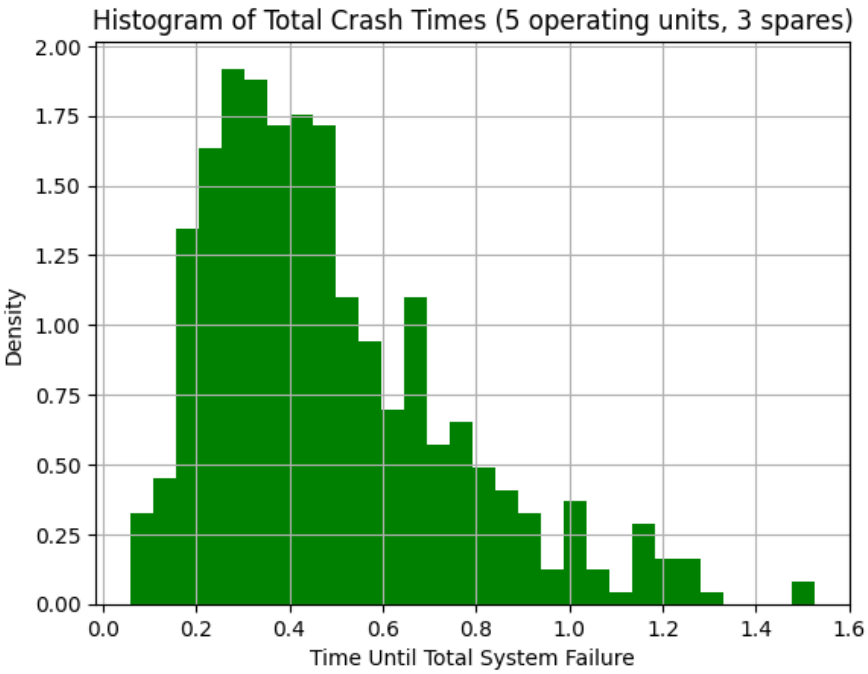

Figure 2: Distribución de tiempos hasta fallo total para Modelo 2 (5 operativos, 3 repuestos).

3.1 Análisis Estadístico y Validación

Se planteó la hipótesis:

H_0 : No hay diferencia significativa en el tiempo medio hasta fallo entre los modelos

H_1 : Existe diferencia significativa en el tiempo medio hasta fallo

Se utilizó un test Z para diferencia de medias con nivel de significancia 5%, aplicando el siguiente estadístico:

$$z = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}}$$

donde $\overline{X}_i$, S_i, n_i son la media, desviación estándar y tamaño de muestra para cada modelo.

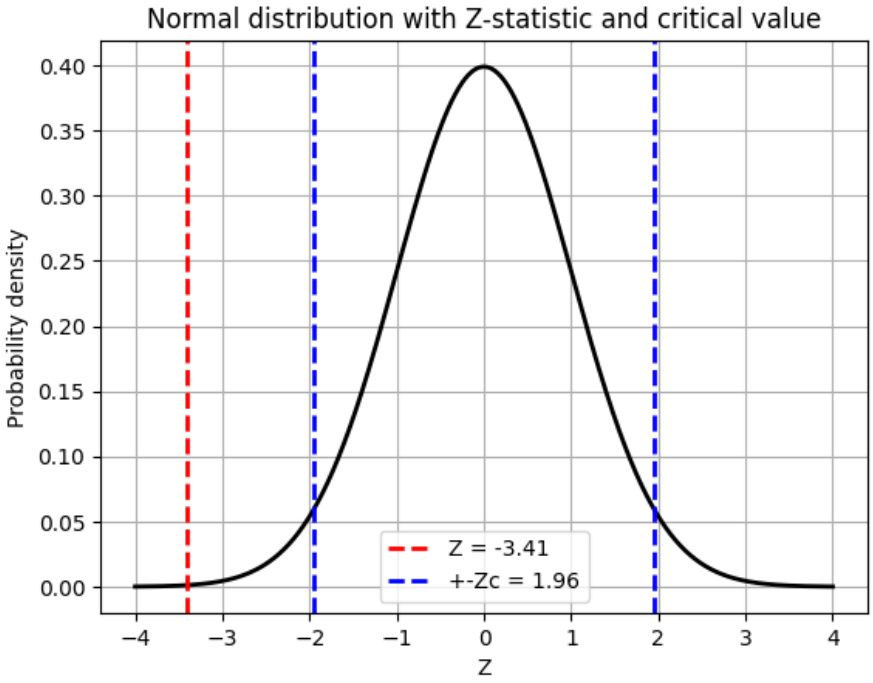

Figure 3: Prueba de hipótesis para diferencia de medias en tiempos hasta fallo, con intervalo crítico en azul y estadístico en rojo (fuera del intervalo).

Dado que el estadístico de prueba se encuentra fuera del intervalo crítico, se rechaza la hipótesis nula. Por lo tanto, existe evidencia estadística para afirmar que hay una diferencia significativa en el tiempo medio hasta fallo entre los modelos comparados.

Este resultado indica que la configuración con mayor cantidad de servidores de repuesto impacta significativamente el tiempo hasta el fallo total del sistema bajo las condiciones simuladas.

Finalmente, se repitió el proceso para otro par modelos, lo cual validó también la hipótesis inicial:

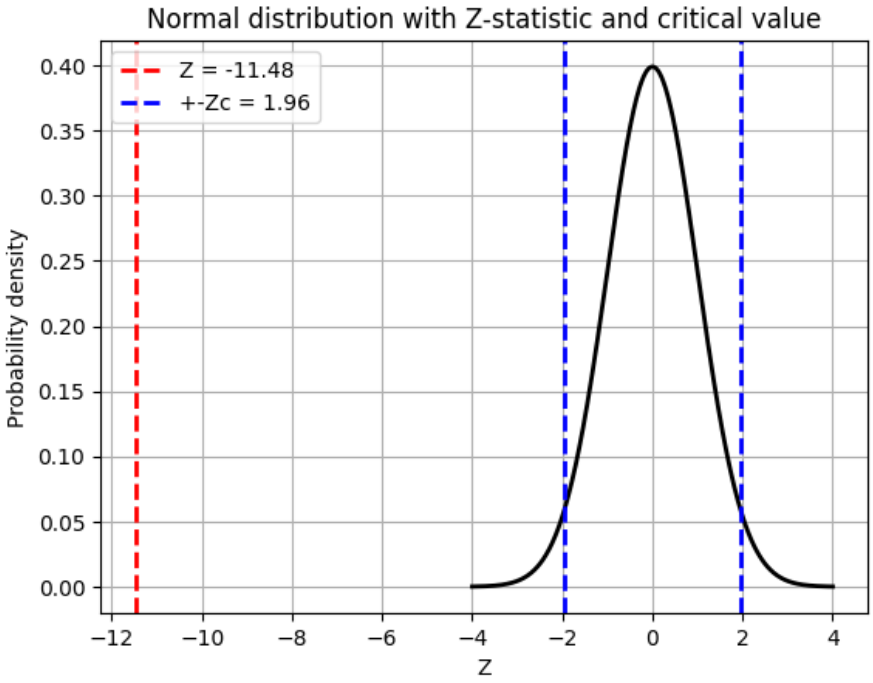

Figure 4: Prueba de hipótesis para diferencia de medias en tiempos hasta fallo de 6 operativos y 1 repuesto vs. 6 operativos y 4 repuestos, con intervalo crítico en azul y estadístico en rojo (fuera del intervalo).

3.2 Criterio de Parada

Se usó un criterio basado en intervalos de confianza para asegurar precisión estadística:

$$2 \times z_{\alpha/2} \times \frac{S}{\sqrt{n}} \leq \text{error aceptado}$$

donde $z_{\alpha/2}$ es el valor crítico para 95%, S la desviación estándar de la muestra y n el número de simulaciones. Se corrieron simulaciones hasta cumplir este criterio.