# Informe de Simulación con SimGrid

Cristian Bravo
Bruno Bastidas
Silvio Villagra
Daniella Lecanda

**Grupo investigativo Los Catalizadores**

27/03/2025

# Índice

**1. Introducción** **3**

**2. Planes y estrategias previas a la creación del script** **3**

**3. Descripción de la Plataforma** **4**

**4. Definición de Enlaces y Rutas** **5**

**5. Implementación del Script** **6**

**6. Conclusiones y Recomendaciones** **9**

# 1. Introducción

SimGrid es una herramienta para la simulación de sistemas distribuidos que permite diseñar, modelar y probar diferentes configuraciones de máquinas, enlaces de red y aplicaciones distribuidas. Este informe tiene como objetivo mostrar el proceso de creación y el razonamiento detrás de un *script* de simulación en SimGrid, así como la justificación de las decisiones tomadas en cada paso.

En esta simulación, se busca emular un escenario en el que existe un **host principal** (*Horus*) que coordina tareas con otros nodos (*Anubis*, *Himotep* y *Ra*). A lo largo de este documento, se describen los aspectos más relevantes de la plataforma, la infraestructura, los enlaces y la aplicación distribuida.

# 2. Planes y estrategias previas a la creación del script

Antes de proceder con la configuración y escritura del *script* en XML, se definieron objetivos y estrategias clave:

- **Objetivo Principal:** Configurar un ambiente de cómputo distribuido que permita coordinar tareas desde un *host* principal (**Horus**) hacia varios *slaves* (**Anubis**, **Himotep** y **Ra**).
- **Selección de Máquinas:** Se eligieron cuatro máquinas con igual capacidad de cómputo (8 núcleos cada una). La elección de 8 núcleos responde a la necesidad de atender múltiples procesos de forma paralela y cumplir con el requerimiento dado en el taller.
- **Latencias y Ancho de Banda:** Se utilizaron enlaces con latencias diferentes para simular distancias o calidades de red distintas. El *enlace principal* (Enlace 5) es el más rápido (latencia de 1ms) para mejorar la comunicación entre Horus y Anubis.
- **Justificación de la Topología:** Se decidió usar una topología donde **Horus** se conecta directamente a los tres nodos (*Anubis*, *Himotep* y *Ra*) mediante enlaces independientes. Esto facilita la coordinación centralizada.

- **Pasos en la Configuración:**

  1. Determinar hosts y capacidad de cada uno.
  2. Asignar latencias y anchos de banda.
  3. Definir rutas de red.
  4. Especificar la infraestructura en la sección `<clouds>`.
  5. Configurar la aplicación (procesos *Master* y *Slave*).

# 3. Descripción de la Plataforma

La plataforma define la información de cómputo básica de cada nodo. En este caso, se han creado cuatro hosts:

- **Horus (Host principal):** 8 núcleos, potencia de 8095000000 (aprox. 9.5 GFlops).
- **Anubis:** 8 núcleos, potencia de 8095000000.
- **Himotep:** 8 núcleos, potencia de 8095000000.
- **Ra:** 8 núcleos, potencia de 8095000000.

Esta configuración **justifica** la homogeneidad entre los nodos y permite una distribución equilibrada de tareas. La decisión de usar 8 núcleos se basa en la necesidad de procesar múltiples procesos *Slave* de forma simultánea y cumplir con los requisitos de la simulación solicitada.

A continuación, se muestra el bloque de código XML correspondiente a la definición de la *platform* y sus hosts (extracto relevante):

Listing 1: Definición de la plataforma

```
<platform version="3">
    <AS id="AS0" routing="Full">
        <!-- Preguntar por ‘horus_avail.trace‘ -->
        <host id="Horus" core="8" power="8095000000"
              availability_file="horus_avail.trace"
              state_file="horus.state" />
        <host id="Anubis" core="8" power="8095000000"/>
        <host id="Ra" core="8" power="8095000000"/>
```

```
        <host id="Himotep" core="8" power="8095000000"/>

        ...
    </AS>
</platform>
```

# 4. Definición de Enlaces y Rutas

Para la conexión entre los nodos, se definieron cinco enlaces principales con distinto ancho de banda y latencia. Se eligió un ancho de banda de 125000000 (bits/s) y latencias adaptadas al requerimiento. El **Enlace 5**, que une *Horus* y *Anubis*, tiene la latencia más baja (1ms) para priorizar la comunicación entre el *host principal* y uno de los nodos.

- **link1** (Horus - Himotep): latencia de 0.0005 (0.5ms).
- **link2** (Horus - Ra): latencia de 0.0005 (0.5ms).
- **link3** (Himotep - Anubis): latencia de 0.005 (5ms).
- **link4** (Ra - Anubis): latencia de 0.005 (5ms).
- **link5** (Horus - Anubis): latencia de 0.001 (1ms).

La elección de latencias más altas (5ms) entre *Himotep/Anubis* y *Ra/Anubis* obedece a la simulación de enlaces menos fiables o con mayor distancia física.

En el siguiente extracto de código se muestra la definición de los enlaces y las rutas:

Listing 2: Definición de enlaces y rutas

```
<link id="link1" bandwidth="125000000" latency="0.000500"
      bandwidth_file="link1.bw" latency_file="link1.lat"/>
<link id="link2" bandwidth="125000000" latency="0.000500"/>
<link id="link3" bandwidth="125000000" latency="0.005000"/>
<link id="link4" bandwidth="125000000" latency="0.005000"/>
<link id="link5" bandwidth="125000000" latency="0.001"/>

<route src="Horus" dst="Anubis">
```

```
    <link_ctn id="link5"/>
</route>
<route src="Anubis" dst="Horus">
    <link_ctn id="link5"/>
</route>

<route src="Horus" dst="Himotep">
    <link_ctn id="link1"/>
</route>
<route src="Horus" dst="Ra">
    <link_ctn id="link2"/>
</route>
<route src="Ra" dst="Anubis">
    <link_ctn id="link4"/>
</route>
<route src="Himotep" dst="Anubis">
    <link_ctn id="link3"/>
</route>
```

Como se observa, **link5** es bidireccional para la conexión *Horus-Anubis*, mientras que los demás enlaces están definidos como rutas unidireccionales específicas.

# 5. Implementación del Script

La implementación se divide en:

1. **Sección** `<clouds>`: Configura la infraestructura en la nube (*myCloud*), el almacenamiento (*myStorage*) y las máquinas virtuales (VMs) que se levantarán en cada host.

2. **Sección** `<processes>`: Define los procesos *Master* y *Slave* que se ejecutarán sobre cada host.

## Infraestructura

En el bloque siguiente, se muestra cómo se define un **almacenamiento** y un **componente de cómputo** que controla las imágenes (*myImage*) y las instancias (*vm1*, *vm2*, *vm3*). Se usa la configuración con `type="XL"` para asignar 8 núcleos a cada VM, cumpliendo el requisito de usar 8 núcleos.

Listing 3: Definición de nubes

```
<clouds version="1">
    <cloud id="myCloud">
        <storage id="myStorage"
                engine="org.simgrid.schiaas.engine.storage.rise.Rise
                    ">
            <config controller="Horus"/>
        </storage>

        <compute engine="org.simgrid.schiaas.engine.compute.rise.
            Rice">
            <config controller="Horus" image_storage="myStorage"
                    image_caching="PRE" boot_delay="10"/>

            <instance_type id="XS" core="1" memory="2000" disk="40000
                "/>
            <instance_type id="S" core="2" memory="4000" disk="80000"
                />
            <instance_type id="M" core="4" memory="8000" disk="160000
                "/>
            <instance_type id="L" core="6" memory="12000" disk="
                240000"/>
            <instance_type id="XL" core="8" memory="16000" disk="
                320000"/>

            <image id="myImage" size="1073741824"/>

            <host id="Anubis"/>
            <host id="Himotep"/>
            <host id="Ra"/>

            <instance id="vm1" host="Anubis"
                     type="XL" image="myImage"/>
            <instance id="vm2" host="Himotep"
                     type="XL" image="myImage"/>
            <instance id="vm3" host="Ra"
                     type="XL" image="myImage"/>
        </compute>
    </cloud>
</clouds>
```

## Aplicación

Para la aplicación, se definió un **proceso Master** en `Horus` y múltiples procesos *Slave* distribuidos en `Anubis`, `Himotep` y `Ra`. Cada uno de estos *Slaves* realiza un trabajo en paralelo conforme a las tareas enviadas por el *Master*.

Listing 4: Definición de procesos

```
<processes>
    <!-- Proceso Master en el Host principal (Horus) -->
    <process host="Horus" function="cloud.schiaas.Master">
        <!--
          Ejemplo de argumentos:
            (1) nmero de tasks: 80
            (2) tamao de tareas: 5e10
            (3) lmite de iteraciones: 1000000
            (4) ncleos a utilizar: 8
        -->
        <argument value="80"/>
        <argument value="5e10"/>
        <argument value="1000000"/>
        <argument value="8"/>
    </process>

    <!-- 8 procesos Slave en Anubis, uno por cada ncleo -->
    <!-- Se repite 8 veces: <process host="Anubis" function="cloud.
        schiaas.Slave"/> -->
    <process host="Anubis" function="cloud.schiaas.Slave"/>
    <process host="Anubis" function="cloud.schiaas.Slave"/>
    <!-- ... (total 8 procesos) ... -->

    <!-- 8 procesos Slave en Himotep -->
    <!-- Se repite 8 veces: <process host="Himotep" function="cloud.
        schiaas.Slave"/> -->
    <process host="Himotep" function="cloud.schiaas.Slave"/>
    <process host="Himotep" function="cloud.schiaas.Slave"/>
    <!-- ... (total 8 procesos) ... -->
```

```
    <!-- 8 procesos Slave en Ra -->
    <!-- Se repite 8 veces: <process host="Ra" function="cloud.
        schiaas.Slave"/> -->
    <process host="Ra" function="cloud.schiaas.Slave"/>
    <process host="Ra" function="cloud.schiaas.Slave"/>
    <!-- ... (total 8 procesos) ... -->

</processes>
```

**Justificación:** Este esquema permite a **Horus** centralizar la distribución de tareas hacia los demás *Slaves*, aprovechando la potencia de cómputo de cada nodo con 8 núcleos.

# 6. Conclusiones y Recomendaciones

- Se demostró la creación de un escenario de simulación con **SimGrid** que integra múltiples hosts, enlaces y una aplicación distribuida.
- La estrategia de configurar un **host principal (Horus)** con enlaces directos y de latencia baja a los demás nodos (*Anubis, Himotep, Ra*) facilita la administración de tareas y la prueba de distintos escenarios de red.
- Mantener la homogeneidad en el número de núcleos y en la potencia de cómputo en cada host resulta útil para simplificar la planificación de tareas y el balance de carga.
- Para posibles mejoras, se podrían variar más las latencias y anchos de banda para simular redes heterogéneas y estudiar su impacto en el rendimiento de la aplicación.
- La incorporación de trazas de disponibilidad (`.trace`) y estados (`.state`) brinda mayor realismo al permitir emular fallos o mantenimiento de nodos.