Simulation as a Service with PowerDEVS-Cloud

Ezequiel Pecker-Marcosig[1], Tomás Paradelo[2], Esteban Lanzarotti[1,2], and Rodrigo Castro[1,2]

[1] Instituto de Ciencias de la Computación (ICC, UBA-CONICET), Ciudad Universitaria, Pabellón 0+Infinito, C1428EGA, Buenos Aires, ARGENTINA
[2] Departamento de Computación (DC, FCEyN-UBA), Ciudad Universitaria, Pabellón 0+Infinito, C1428EGA, Buenos Aires, ARGENTINA
emarcosig@dc.uba.ar, tparadelo@dc.uba.ar, elanzarotti@dc.uba.ar, rcastro@dc.uba.ar

Abstract. The Modeling and Simulation as a Service (MSaaS) methodology offers significant advantages, such as immediate availability of tools without requiring prior installation, cross-platform support, ease of model sharing, and the ability to run large-scale simulations on high-performance servers. This work presents *PowerDEVS-Cloud*, a middleware that provides a web-based service for executing simulations of DEVS models using the PowerDEVS simulator and its scripting extension py2PowerDEVS. The proposed architecture consists of a Management Server and multiple Simulation Servers with different computational capabilities, which can run in different setups. This approach enables concurrent simulation execution for multiple users and flexible resource allocation. Throughout three application examples, the capabilities of the system are highlighted, demonstrating its potential to facilitate remote access to advanced simulation tools.

Keywords: Modeling and Simulation as a Service (MSaaS), Workflow , Discrete-Event System Specification (DEVS), PowerDEVS

Simulación como Servicio con PowerDEVS-Cloud

Abstract. La metodología de Modelado y Simulación como Servicio (MSaaS) ofrece ventajas significativas, como la disponibilidad inmediata de herramientas sin necesidad de instalación previa, el soporte multiplataforma, la facilidad para el intercambio de modelos y la posibilidad de ejecutar simulaciones de gran escala en servidores con alta capacidad de cómputo. Este trabajo presenta *PowerDEVS-Cloud*, un middleware que provee un servicio web para la ejecución de simulaciones de modelos DEVS utilizando el simulador PowerDEVS y su extensión

py2PowerDEVS. La arquitectura propuesta se compone de un Servidor de Gestión y múltiples Servidores de Simulación con distintas capacidades de cómputo, que pueden ejecutarse en distintas configuraciones. Este enfoque permite la ejecución concurrente de simulaciones para múltiples usuarios y una asignación flexible de recursos. A través de tres casos de estudio, se destacan las capacidades del sistema y su potencial para facilitar el acceso remoto a herramientas de simulación avanzadas.

Keywords: Modelado y Simulación como Servicio (MSaaS), Workflow, Simulación de Eventos Discretos (DEVS) , PowerDEVS

1 Introducción

Dada su naturaleza interdisciplinaria, la disciplina de Modelado y Simulación (M&S) debe ser accesible para usuarios con perfiles diversos y distintos niveles de experiencia, democratizando así el acceso a las herramientas de simulación. Sin embargo, la instalación y configuración del software de simulación (incluyendo la resolución de sus dependencias) generalmente requiere habilidades técnicas que exceden las de muchos potenciales usuarios. En consecuencia, existe la necesidad de herramientas que estén listas para su utilización desde el primer momento, que ofrezcan una experiencia de usuario accesible y que permitan concentrarse únicamente en la experimentación y el análisis, en lugar de en la configuración del entorno.

Esto resulta particularmente relevante porque la experimentación basada en simulación es inherentemente iterativa: los usuarios ejecutan simulaciones, analizan los resultados, modifican parámetros o escenarios y repiten el proceso. Las herramientas que soportan este flujo de trabajo de manera ágil e interactiva pueden mejorar significativamente la productividad. En este contexto, las herramientas de simulación basadas en la web (WBS) que utilizan servicios en la nube resultan especialmente atractivas, ya que eliminan las barreras de instalación y permiten un uso inmediato.

Por su parte, el formalismo DEVS (Discrete Event System Specification) (Zeigler et al., 2018) ha demostrado ser adecuado para representar un amplio rango de otros formalismos matemáticos (Vangheluwe, 2000) que cubren distintas disciplinas. DEVS soporta la construcción de modelos dinámicos híbridos que combinan comportamientos de tiempo continuo, tiempo discreto y eventos discretos, tanto determinísticos como estocásticos.

El paradigma *as a service* ofrece funcionalidades como servicios disponibles en la nube listos para usar. En particular, bajo los paradigmas de Modelado-y-Simulación-como-Servicio (MSaaS) (Cayirci, 2013; Shahin et al., 2020) y Simulación-como-Servicio (SimaaS) (Wang & Wainer, 2015) un usuario puede acceder a funcionalidades de modelado y/o simulación como servicios *a demanda*.

En este contexto, este trabajo presenta *PowerDEVS-Cloud*, una extensión de la herramienta de simulación PowerDEVS que habilita la ejecución de simulaciones DEVS y experimentación numérica directamente en la nube, sin requerir de la instalación de software adicional. *PowerDEVS-Cloud* es totalmente compatible con el ecosistema existente de herramientas y modelos de PowerDEVS. Las principales contribuciones son: (1) una herramienta de simulación multiplataforma disponible para su uso inmediato; (2) eliminación de dependencias específicas de la plataforma, reduciendo así la barrera de entrada; y (3) una interfaz transparente para la ejecución de simulaciones de modelos complejos y/o de gran tamaño mediante el uso de servidores con mayor poder de cómputo.

La estructura de este trabajo es la siguiente. La Sección 2 introduce conceptos y herramientas que serán utilizados a lo largo del trabajo, incluyendo una revisión de literatura relacionada. La Sección 3 presenta la arquitectura de *PowerDEVS-Cloud*, mientras que la Sección 4 describe los distintos esquemas de ejecución. La Sección 5 describe tres casos de estudio, elegidos para resaltar distintos modos de utilización, destacando la capacidad de intervenir simulaciones y los datos generados de manera interactiva en tiempo de ejecución. Finalmente, la Sección 6 concluye el trabajo.

2 Conceptos Preliminares

2.1 El Formalismo de Especificación de Sistemas de Eventos Discretos (DEVS)

Para el modelado y simulación de sistemas dinámicos híbridos, utilizamos el formalismo de *Especificación de Sistemas de Eventos Discretos* (DEVS) (Zeigler et al., 2018). DEVS puede representar cualquier sistema discreto (de eventos discretos o de tiempo discreto) siempre y cuando produzca un número finito de eventos en un intervalo de tiempo finito. DEVS también permite aproximar sistemas continuos con cualquier nivel de precisión requerido, recurriendo a los métodos numéricos de *sistemas de estados cuantizados* (QSS) (Castro et al., 2024), y expresar el comportamiento estocástico para sistemas híbridos generalizados (Castro et al., 2010).

Un aspecto destacable de DEVS es su estructura modular y jerárquica, basada en modelos estructurales (acoplados) y de comportamiento (atómicos), lo que promueve la construcción incremental de modelos complejos, la reutilización y sustitución de módulos de forma robusta, y la construcción jerárquica de modelos. Esto fomenta la definición de bibliotecas de modelos.

Formalmente, un modelo *Atómico* DEVS (clásico) es un bloque de construcción minimal definido como $M_A = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$, donde X es el conjunto de eventos de entrada, Y es el conjunto de eventos de salida, y S es el conjunto de estados internos. Cuatro funciones dinámicas definen su comportamiento: δ_{int} (función de transición interna, define el comportamiento autónomo), δ_{ext} (función de transición externa, recepción de eventos y define el comportamiento reactivo), λ (función de salida, emisión de eventos) y ta (función de avance de tiempo, sincronización autónoma). Cuando el modelo atómico

alcanza un estado $s \in S$, y en ausencia de eventos de entrada externos, permanecerá en s durante un tiempo $ta(s)$.

Un modelo *Acoplado* DEVS es una red de modelos atómicos y/o acoplados definida como $M_C = \langle X_{self}, Y_{self}, D, \{M_d\}, \{I_d\}, \{Z_{d,j}\}, Select\rangle$, donde $self$ es el propio modelo acoplado, X_{self} e Y_{self} son los conjuntos de valores de entrada y salida, D es el diccionario de componentes en $self$, y M_d ($d \in D$) es el modelo DEVS asociado a d. Para cada $d \in D \cup \{self\}$, I_d es el conjunto de modelos que influencian a d, y para cada $j \in I_d$, $Z_{d,j} : Y_d \to X_j$ es la función de traducción de d a j. Para eventos simultáneos, $Select : 2^D \to D$ es la función de desempate.

2.2 La herramienta de simulación PowerDEVS

El simulador PowerDEVS (Bergero & Kofman, 2011) es una herramienta de simulación DEVS particularmente adecuada para la simulación eficiente de sistemas dinámicos híbridos, dado que es la herramienta de simulación DEVS insignia de la familia de métodos numéricos QSS (Castro et al., 2024). PowerDEVS se destaca por sobre otras herramientas de simulación DEVS por su interfaz gráfica de usuario (GUI) basada en bloques (Van Tendeloo & Vangheluwe, 2017), que mantiene ocultos los aspectos internos de modelos DEVS facilitando su utilización por usuarios no familiarizados con DEVS.

PowerDEVS comprende cuatro programas independientes. **a)** El *Editor de Modelos* es la interfaz gráfica de usuario (GUI) para construir y configurar modelos de simulación utilizando un enfoque de modelado orientado a bloques, en donde los modelos se construyen arrastrando, soltando y conectando bloques funcionales de modelos DEVS. **b)** El *Editor de Atómicos* es un editor de texto estructurado con solapas para describir el comportamiento de los modelos atómicos. **c)** El *Pre-Procesador* genera código C++ a partir de la representación del modelo gráfico, que luego se compila para producir un ejecutable que incluye la especificación del modelo y el motor de simulación. **d)** La *Interfaz de Simulación* es una interfaz gráfica para el manejo del ejecutable, que permite definir parámetros (como el tiempo final de simulación, el número de simulaciones y el modo de ejecución -normal, tiempo real, paso a paso, etc.-). Los modelos y las bibliotecas creadas con la GUI tienen extensión `.pdm` y `.pdl`, respectivamente.

Para la construcción de modelos de gran tamaño, el procedimiento de arrastrar, soltar e interconectar bloques gráficos en el *Editor de Modelos* se vuelve una tarea repetitiva, dificultosa y propensa a errores. Por este motivo, se desarrolló la extensión *py2PowerDEVS* (Pecker-Marcosig et al., 2022) que permite la construcción de modelos de PowerDEVS por medio de scripts de Python.

2.3 Trabajos Relacionados

La *Simulación Basada en Web* (WBS) se refiere a la integración de tecnologías web con el campo de la simulación (Fishwick, 1996). En términos de arquitectura, Byrne et al. (2010) clasifica WBS en tres categorías: (1) *Simulación y Visualización Local* (LSV), (2) *Simulación y Visualización Remota* (RSV), y (3) *Simulación y Visualización Híbrida* (HSV). En LSV la interfaz gráfica y el

motor de simulación coexisten y se ejecutan del lado del cliente, dentro de un navegador web. En RSV, tanto el motor de simulación como la visualización se ejecutan del lado del servidor y son accedidos a través de la web. Finalmente, en HSV, el motor de simulación corre remotamente mientras que la visualización –y en algunos casos la construcción del modelo– se realiza del lado del cliente.

El paradigma RSV se ajusta con el enfoque de Modelado-y-Simulación-como-Servicio (MSaaS) (Shahin et al., 2020) y Simulación-como-Servicio (SimaaS) (Wang & Wainer, 2015), que permiten que un usuario pueda desplegar recursos de simulación en la nube (simuladores, modelos, datos) y acceder a funcionalidades de simulación en la nube como servicios a través de interfaces web (WSs). Cayirci (2013) define al MSaaS como un modelo para proveer servicios de modelado y simulación *a demanda* a través de un proveedor de servicios en la nube (CSP) que mantiene ocultos para los usuarios los detalles y requisitos de la infraestructura, la plataforma y el software subyacentes. Estas arquitecturas permiten por ejemplo el acceso a sistemas de cómputo de alto desempeño (HPC) mediante proveedores de cómputo en la nube para la simulación de sistemas de gran escala (Zehe et al., 2015). Las plataformas como Amazon Web Services (AWS), Microsoft Azure (MA) o Google Cloud Platform (GCP) han puesto recursos de cómputo de alto desempeño al alcance de usuarios en general.

De acuerdo con Wang and Wainer (2015), los WSs relacionados con simulación se agrupan en RESTful y SOAP. Los WSs SOAP presentan desventajas frente a REST: (a) la interacción cliente–servidor está fuertemente acoplada, por lo que cualquier cambio en el servidor requiere modificaciones en el cliente; (b) su payload es de mayor tamaño, y (c) expone detalles de la implementación interna dificultando el desarrollo. REST expone los servicios como recursos accesibles mediante URIs y métodos HTTP estándar (GET, PUT, POST, DELETE), habilitando el desarrollo de *entornos de trabajo distribuidos* a nivel de aplicación.

En el contexto del modelado basado en DEVS, se han propuesto varios enfoques de MSaaS. Aun cuando los primeros simuladores DEVS basados en WSs utilizaban CORBA (Cho et al., 2003) y SOAP (Mittal et al., 2007, 2009), los trabajos más recientes se basan en REST (Al-Zoubi & Wainer, 2013, 2015; Capocchi & Santucci, 2021). Estos servicios fueron luego integrados en distintos flujos de trabajo. Por ejemplo, RISE y CloudRISE fueron integradas con herramientas como CD++ para soportar modelos DEVS y Cell-DEVS (Al-Zoubi & Wainer, 2015; Wang & Wainer, 2015). DEVSimPy (Capocchi & Santucci, 2021) se ha integrado con *frontends* desarrollados como aplicaciones móviles para controlar y visualizar simulaciones ejecutadas de forma remota mediante una API REST (DEVSimPy-REST). Una limitación de estas herramientas es que están restringidas a una única plataforma de ejecución en la nube (en general, un servidor propio).

3 Arquitectura del Sistema

Desde las etapas tempranas de diseño, se adoptó como principio el desacoplamiento entre la interfaz de usuario y la ejecución de las simulaciones.

En consecuencia, la arquitectura del sistema (ver Figura 2) se organiza en dos servidores complementarios: (a) Servidor de Gestión, y (b) Servidor de Simulación. El Servidor de Gestión actúa como el punto de entrada al sistema, gestionando la interacción con el usuario y la redirección de pedidos. Por su parte, el Servidor de Simulación PowerDEVS es el responsable de resolver los pedidos del usuario, conectarse con el Servidor de Gestión y controlar la ejecución de una simulación.

La comunicación de un usuario con ambos servidores es mediante una API basada en REST. A través de esta API, un usuario ve al simulador como una caja negra con la cual interactúa por medio de *endpoints*. Si bien existen diversas herramientas cliente para invocar *requests* REST (como Postman, wget o librerías de Python), en este trabajo se utiliza la herramienta de línea de comandos (CLI) `curl`.

3.1 Servidor de Gestión

El Servidor de Gestión (llamado *Instance Server*) actúa como proxy hacia los Servidores de Simulación. Es la única interfaz que ve el usuario final. Este servidor está escrito en Python y utiliza Flask. El Servidor de Gestión escucha en el puerto 5000 por defecto.

Las instancias de PowerDEVS disponibles para realizar simulaciones se ejecutan en Servidores de Simulación (Sección 3.2). En el caso de haber más de un Servidor de Simulación (*resource pooling*), el Servidor de Gestión mantiene actualizado un registro de simuladores disponibles mediante el mecanismo de *heartbeat*. Cada Servidor de Simulación ejecuta en paralelo el script `register_heartbeat.py`, que envía periódicamente un mensaje al Servidor de Gestión para indicar que está activo y disponible. Si el Servidor de Gestión no recibe un *heartbeat* dentro de un período configurable, descarta esa instancia del registro.

El endpoint `/instances` permite consultar todas las instancias de simulación activas. Luego de la ejecución de una simulación, los resultados quedan almacenados en una carpeta con un `<id>` único en el Servidor de Simulación que contiene los logs del simulador y de los modelos (archivos: `pdevs.log`, `.csv` o `.h5`). Los resultados de todas las simulaciones pueden ser accedidos en cualquier momento (por cualquier usuario) mediante el endpoint `/runs`. Para ejecutar endpoints del Servidor de Simulación, el usuario lo hace de forma transparente a través del Servidor de Gestión mediante el endpoint `/proxy/<instance_id>/<endpoint>`.

El Servidor de Gestión expone además endpoints de administración de la infraestructura, tales como `/create-instance` (crear una nueva instancia EC2), `/start-simulator/<instance_id>` (iniciar `pdserver` en una instancia) y `/check-server/<instance_id>` (verificar que un `pdserver` esté activo).

3.2 Servidor de Simulación

El Servidor de Simulación (llamado `pdserver`) se encarga de manejar los pedidos del Servidor de Gestión, y de gestionar el modelo (e.g. compilación) y

la simulación (e.g. ejecución). Este servidor está implementado en C++. Para simplificar su ejecución en distintas plataformas (incluyendo la satisfacción de dependencias y software adicional), se recurre a la utilización de un servicio de virtualización con Docker. El Servidor de Simulación escucha en el puerto 6000 por defecto, configurable mediante una variable de entorno.

Para asegurarse de que el servicio se está ejecutando correctamente en el servidor y es accesible desde el cliente se cuenta con el endpoint `/hi`. Para la ejecución de un modelo desarrollado con la GUI de PowerDEVS el primer paso es generar el código C++ equivalente del archivo `.pdm`, compilarlo y luego ensamblar el código objeto de los modelos y el simulador mediante el *Pre-Procesador* con el endpoint `/compile-model`.

El procedimiento para la ejecución de modelos desarrollados con py2PowerDEVS es sustancialmente diferente (Pecker-Marcosig et al., 2022). El primer paso es exponer los métodos de los atómicos definidos en C++ para poder ser accedidos desde Python en el módulo `py2pdevs.core`. Luego, las bibliotecas de modelos desarrolladas para py2PowerDEVS en archivos `<my-library>.pdl` deben ser convertidas en módulos `py2pdevs.<my-library>` mediante el *Pre-procesador*. Para realizar estas tareas se utiliza el endpoint `/rebuild`. Luego, para seleccionar el archivo con el modelo a simular se utiliza el endpoint `/set-py2pdevs-model`.

Para la ejecución de simulaciones completas, la plataforma ofrece variantes bloqueantes y no bloqueantes. Para modelos desarrollados con la GUI se cuenta con `/run-model-blocking` y `/run-model-async` , mientras que sus equivalentes para py2PowerDEVS son `/run-py2pdevs-blocking` y `/run-py2pdevs-async`. Esta distinción responde a que la ejecución de modelos complejos y/o de gran escala puede requerir tiempos de cómputo elevados. En este contexto, las variantes no bloqueantes permiten desacoplar la solicitud del cliente del procesamiento de la simulación, evitando mantener la conexión abierta y facilitando una gestión eficiente de recursos. Estos endpoints reciben los parámetros del modelo y del simulador en una estructura tipo JSON y devuelven un identificador numérico de la simulación y los nombres de los archivos generados: log de la simulación y archivos de resultados.

Para recuperar los resultados de una simulación y procesarlos localmente, se cuenta con los endpoints `/get-results` y `/get-py2pdevs` respectivamente, que reciben el identificador de la simulación y el nombre del archivo a descargar.

La Figura 1 ilustra la secuencia completa de *requests* para compilar o seleccionar un modelo, ejecutar la simulación y descargar los resultados para los flujos de la GUI y de py2PowerDEVS, respectivamente. En ambos casos el usuario se comunica únicamente con el Servidor de Gestión, que actúa como *proxy* y reenvía cada operación al Servidor de Simulación correspondiente.

Una ventaja de *PowerDEVS-Cloud*, comparada con otras herramientas DEVS para MSaaS, es que provee endpoints específicos para que un usuario pueda subir sus propios modelos de simulación. El mismo endpoint `/compile-model` puede ser utilizado para subir un archivo `.pdm` creado con la GUI, mientras que `/set-py2pdevs-model` se puede usar para subir un

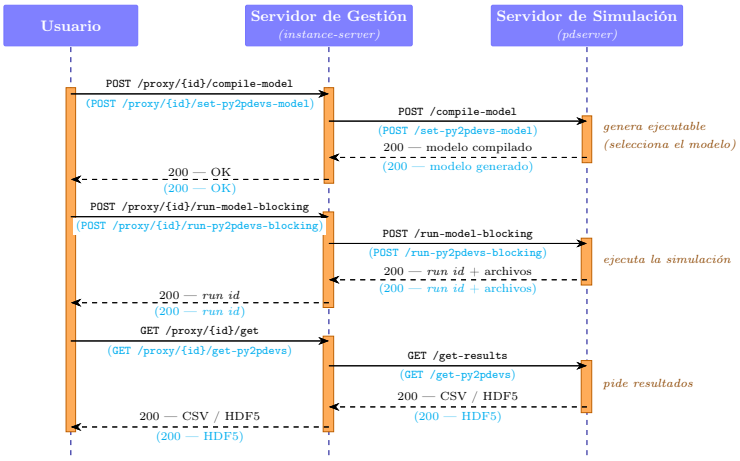

Fig. 1. Flujo de llamadas HTTP para simular un modelo desarrollado con la GUI (py2PowerDEVS), empleando la variante bloqueante `/run-model-blocking` (`/run-py2pdevs-blocking`). El usuario dialoga sólo con el Servidor de Gestión, que reenvía cada *request* al Servidor de Simulación.

modelo de py2PowerDEVS. En el caso de que un usuario provea nuevos modelos atómicos (código C++), debe subirlos al Servidor de Simulación por medio del endpoint `/upload-file` y luego compilarlos para generar el código objeto correspondiente y enlazarlos con la biblioteca de modelos atómicos existentes mediante el endpoint `/compile-atomic`. Para poder ser utilizados en py2PowerDEVS debe utilizarse nuevamente el endpoint `/rebuild`.

La utilidad práctica de estos endpoints quedará evidenciada en los ejemplos presentados en la Sección 5.

4 Simulación Descentralizada e Infraestructura de Simulación

A diferencia de las herramientas DEVS en la nube (mencionadas en la Sección 2.3) *PowerDEVS-Cloud* admite distintas configuraciones para la arquitectura, presentadas en la Sección 3, con los Servidores de Gestión y de Simulación. En particular, ambos pueden ejecutarse localmente en la máquina del usuario, distribuirse entre uno o varios servidores remotos, o apoyarse sobre infraestructura tipo *cloud*. Estas alternativas no son excluyentes y pueden combinarse sin alterar el flujo general de interacción con el sistema.

La conveniencia de cada opción depende principalmente de la capacidad de cómputo requerida, de cómo decida manejarse la infraestructura usada (*self hosted* vs. *cloud*) y de la cantidad de ejecuciones. En todos los casos, la coordinación permanece concentrada en el Servidor de Gestión, mientras que la capacidad de simulación puede ampliarse incorporando nuevos Servidores de Simulación. La Fig. 2 resume estas posibilidades.

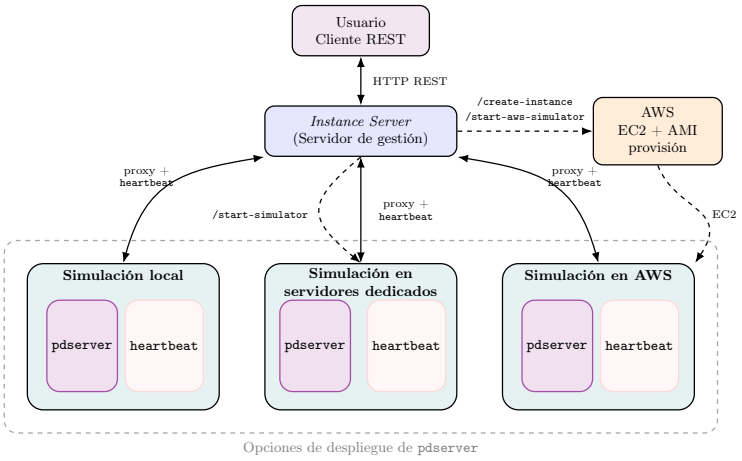

Fig. 2. Arquitectura de despliegue de *PowerDEVS-Cloud*. El usuario interactúa con el Servidor de Gestión, que centraliza la API, administra las distintas opciones de Servidor de Simulación y puede aprovisionar nuevos simuladores sobre AWS.

4.1 Simulación Local

La configuración local constituye la alternativa más directa de lanzamiento, ya que tanto el Servidor de Gestión como el Servidor de Simulación se ejecutan en la misma máquina del usuario. En este escenario, el Servidor de Gestión puede reutilizar un Servidor de Simulación previamente iniciado o lanzar uno o varios simuladores locales encapsulados en contenedores Docker. Para el cliente, esto preserva la misma API REST y el mismo encadenamiento de operaciones que se emplea en despliegues remotos.

Esta modalidad resulta conveniente para desarrollo, depuración y validación de modelos, permite ejecutar el flujo completo sin depender de infraestructura externa y simplifica la transición hacia configuraciones distribuidas, ya que los mismos clientes y scripts pueden reutilizarse sobre servidores remotos o instancias *cloud* con cambios mínimos en la configuración de red. No obstante, esta no es la opción preferida para un usuario general.

4.2 Simulación en Servidores Dedicados

Cuando se dispone de uno o varios servidores dedicados, el Servidor de Gestión puede ejecutarse en un host accesible por red y coordinar varios Servidores de Simulación ubicados en el mismo servidor o distribuidos entre distintos hosts. La disponibilidad de estos simuladores se mantiene mediante el mecanismo de *heartbeat*, que actualiza el conjunto de servidores aptos para recibir pedidos detrás de un único punto de entrada. De esta manera, el usuario interactúa siempre con el mismo servicio de gestión, incluso cuando la capacidad de cómputo esté repartida entre varias máquinas. El endpoint `/start-simulator` del Servidor de

Gestión permite lanzar una nueva instancia de `pdserver` encapsulada en Docker en el mismo servidor, registrándola automáticamente.

Este esquema es cercano a arquitecturas como RISE y CloudRISE (Wang & Wainer, 2015), donde la simulación se ofrece como un servicio remoto accesible a través de interfaces web que facilitan la integración con scripts o clientes externos.

4.3 Simulación en un Servicio de Cloud

Para la ejecución sobre infraestructura *cloud*, la implementación actual integra Amazon Web Services (AWS). En particular, EC2 (*Elastic Compute Cloud*) que provee instancias que cumplen el rol de servidores remotos sobre los cuales puede inicializarse el simulador. En *PowerDEVS-Cloud*, estas instancias se inicializan a partir de una AMI (*Amazon Machine Image*), es decir, una imagen base que contiene el sistema, las dependencias y la configuración inicial necesarias para ejecutar todo el entorno de manera reproducible. El uso de una AMI encapsula los scripts y configuraciones requeridas para poner en servicio un nuevo simulador, evitando repetir manualmente la instalación en cada inicialización. Asimismo, EC2 permite seleccionar el tipo de instancia de acuerdo con la potencia computacional requerida, lo que habilita ampliar la capacidad de simulación según la carga sin modificar la interfaz REST utilizada por los clientes. No se considera aquí el uso de servicios de almacenamiento persistente (como S3 en AWS) ya que, en la versión actual, los resultados se almacenan en la misma instancia donde se corren las simulaciones.

De esta forma, una instancia recién creada puede incorporarse al sistema como un servidor remoto más, sin introducir cambios en la forma de acceder a la instancia. En el Servidor de Gestión, el endpoint `/create-instance` permite crear una nueva instancia de cómputo en AWS, mientras que `/start-aws-simulator/<instance_id>` completa la puesta en marcha del Servidor de Simulación dentro de dicha instancia.

En este sentido, *PowerDEVS-Cloud* permite la posibilidad de desacoplar el acceso al servicio de la infraestructura subyacente. A partir de la interfaz basada en endpoints REST, este mecanismo se puede expandir en el futuro para soportar diferentes proveedores de servicios *cloud* (e.g. MA, GCP). Esto es posible dado que la interfaz de interacción se basa en endpoints REST estándar, por lo que cualquier infraestructura que permita exponer el Servidor de Simulación en una dirección de red accesible es compatible con el sistema, independientemente del proveedor.

4.4 Gestión de Usuarios

La concurrencia entre múltiples usuarios se logra mediante la distribución de las ejecuciones entre distintos Servidores de Simulación: cada proceso de `pdserver` atiende una única simulación activa, por lo que el escalamiento horizontal se obtiene incorporando nuevos simuladores al sistema. *PowerDEVS-Cloud* permite coordinar múltiples instancias de `pdserver` registrados detrás del mismo Servidor de Gestión. Cuando la ejecución se encapsula en Docker, cada simulador se

implementa como un contenedor independiente derivado de una misma imagen base con su propio volmen, lo que facilita la incorporación de nuevas instancias sin necesidad de configuración adicional.

5 Resultados Experimentales

En esta sección se evalúa la API REST desarrollada en la Sección 3 considerando las tres arquitecturas descritas en la Sección 4.

Las tres configuraciones evaluadas son: (1) *Local*, un equipo de escritorio con CPU AMD Ryzen 5 9600X y 32 GB de RAM; (2) *Servidor dedicado* (`networksir`), con CPU Intel Xeon Silver 4208 @ 2.10 GHz y 125 GB de RAM, bajo un esquema de doble virtualización (un contenedor Docker sobre un contenedor LXD); y (3) *AWS EC2*, una instancia de uso general `t3.large` con 2 vCPU y 8 GB de RAM. Se propone como métrica para evaluar la eficiencia de cómputo de cada configuración al tiempo total de resolución (*round-trip* time) de una solicitud HTTP enviada desde el cliente para la ejecución de cada simulación. Sobre cada configuración, se ejecutaron 10 simulaciones de cada caso de estudio (descritos a continuación) y se tomaron valores promedio de la métrica (Tabla 1). Se consideran tres casos de estudio sobre dos modelos: modelo Lotka-Volterra (GUI y py2PowerDEVS), y Red SIR (py2PowerDEVS).

Table 1. Tiempos de ejecución promedio (en segundos) sobre las tres configuraciones. Cada valor es el promedio de 10 ejecuciones, medido como el *round-trip* HTTP completo desde el cliente.

Caso de estudio	Plataforma	Local	Dedicado	AWS
Lotka-Volterra	GUI	0.21 s	0.30 s	0.61 s
Lotka-Volterra	py2PowerDEVS	0.25 s	0.93 s	0.94 s
Red SIR	py2PowerDEVS	34.1 s	365.1 s	111.0 s

5.1 Modelo Poblacional Lotka-Volterra

El modelo Lotka–Volterra (también conocido como modelo depredador-presa) es un modelo poblacional clásico descrito por un sistema de ecuaciones diferenciales ordinarias no lineales de primer orden acopladas (Ecuación (1)).

$$\frac{dx}{dt} = x \cdot (\alpha - \beta \cdot y)\,, \quad \frac{dy}{dt} = -y \cdot (\gamma - \delta \cdot x)\,, \tag{1}$$

donde $x(t)$ representa la población de presas (por ejemplo, conejos), $y(t)$ la población de depredadores (por ejemplo, zorros), y $\frac{dx}{dt}$ y $\frac{dy}{dt}$ denotan sus respectivas tasas de crecimiento y decrecimiento. Los parámetros α, β, γ y δ definen la dinámica de interacción entre ambas especies.

Este modelo fue seleccionado porque es simple pero representativo de una amplia clase de sistemas dinámicos no lineales en diversos dominios como la biología y la economía.

Modelo desarrollado con la GUI Para comenzar, tenemos que cargar el modelo que queremos simular (`lotka-volterra.pdm`) mediante el endpoint `/compile-model`. Este modelo forma parte de la distribución estándar de PowerDEVS y se encuentra en la carpeta: `examples/continuous/lotka_volterra/`.

A continuación, ejecutamos la simulación mediante el endpoint `/run-model-blocking` junto con los parámetros de simulación y del modelo necesarios en formato JSON. En este caso, solo vamos a modificar el tiempo final de simulación (`tf`). La ejecución es bloqueante, y al finalizar devuelve el identificador de la ejecución de la simulación `<id>` y los archivos de resultados y depuración generados. Para descargarlos localmente utilizamos el endpoint `/get-results` especificando el `<id>` y el nombre del archivo `<filename>`.

A continuación se muestra el uso de la API para el caso en que ambos servidores se ejecutan en el servidor dedicado: http://networksir.exp.dc.uba.ar. Para consultar las instancias activas disponibles usamos el endpoint `/instances`.

```
curl -X POST http://networksir.exp.dc.uba.ar:5000/proxy/simulator/compile-
    model?model_path=examples/continuous/lotka-volterra/lotka-volterra.pdm
curl -X POST http://networksir.exp.dc.uba.ar:5000/proxy/simulator/run-model-
    blocking -H "Content-Type: application/json" -d '{"tf":"300"}'
curl -X GET "http://networksir.exp.dc.uba.ar:5000/proxy/simulator/get-results
    ?run=<id>&file=<filename>" -o <filename>
```

Se simularon 300 s de tiempo virtual bajo las tres configuraciones (Tabla 1), con tiempos de ejecución de 0.21 s (*Local*), 0.30 s (*Servidor dedicado*) y 0.61 s (*AWS EC2*). En un modelo de esta escala el cómputo es despreciable y los tiempos quedan dominados por costos fijos (comunicación HTTP e inicio del proceso), por lo que las diferencias reflejan el entorno de acceso más que la eficiencia de cómputo. En este sentido, cabe destacar que la instancia de AWS tiene mayor latencia al estar físicamente ubicada en Estados Unidos.

Modelo desarrollado con py2PowerDEVS Para cargar el modelo que queremos simular (`lotka_volterra.py`) construido con py2PowerDEVS utilizamos el endpoint `/set-py2pdevs-model`.

A continuación, ejecutamos la simulación mediante el endpoint `/run-py2pdevs-blocking`, suministrando los parámetros de simulación y del modelo. Los parámetros del modelo son las poblaciones iniciales de presas (`prey_0`) y depredadores (`pred_0`). Los parámetros de la simulación incluyen: (a) el tiempo final (`tf`), (b) el tiempo de muestreo para el registro (`samplingPeriodForPlots`), (c) el *backend* para el registro (`variable_logging_backend`), y (d) los niveles de registro (`debugLevel` y `logLevel`).

Este último endpoint devuelve el indentificador de la ejecución `<id>` y el nombre del archivo de resultados (`Lotka-Volterra0_0.h5`). Estos valores se utilizan entonces para obtener los resultados mediante el endpoint `/get-py2pdevs`.

```
curl -X POST "http://networksir.exp.dc.uba.ar:5000/proxy/simulator/set-
    py2pdevs-model?model_path=examples/continuous/lotka_volterra/python/
    lotka_volterra.py"
curl -X POST http://networksir.exp.dc.uba.ar:5000/proxy/simulator/run-
    py2pdevs-blocking -H "Content-Type: application/json" -d "{"tf": "100","
```

```
    samplingPeriodForPlots":"0.001","variable_logging_backend":"hdf5","
    debugLevel":"1","logLevel":"1000","prey_0":"0.5","pred_0":"0.5"}"
curl -X GET -d " " "http://networksir.exp.dc.uba.ar:5000/proxy/simulator/get-
    py2pdevs?run=<id>&file=Lotka-Volterra0_0.h5" -o Lotka-Volterra0_0.h5
```

Bajo las mismas tres configuraciones, los tiempos de ejecución fueron 0.25 s (*Local*), 0.93 s (*Servidor dedicado*) y 0.94 s (*AWS EC2*) (Tabla 1). Nuevamente, se trata de un modelo de pequeña escala donde predominan los costos fijos; el orden entre configuraciones se mantiene como en el caso anterior desarrollado con la GUI; sin embargo, en este caso, la diferencia de tiempos es más notoria, siendo este modelo más dependiente de los recursos de cómputo.

5.2 Red de Modelos SIR Interconectados (py2PowerDEVS)

En esta sección se analizará un sistema compuesto por una red de modelos de propagación de enfermedades de tipo SIR (Susceptible-Infectado-Removido) desarrollado en (Pecker-Marcosig et al., 2022).

A diferencia de los modelos de las secciones anteriores, este modelo no forma parte de la distribución estándar de PowerDEVS. En consecuencia, el primer paso consiste en subir el modelo (`multi_SIR.py`) junto con la biblioteca de modelos acoplados (`coronavirus.pdl`) y archivos auxiliares (archivos `csv` con datos reales de casos) al servidor con el endpoint `/upload-file?directory=<remote-folder>&name=<remote-filename>`, adjuntando el archivo local en el cuerpo del *request*. A continuación, para convertir la biblioteca de modelos en un módulo de Python se utiliza el endpoint `/rebuild`.

Dado que se trata de un modelo de gran tamaño (1431 modelos SIR acoplados, con 16 modelos atómicos cada uno, interconectados por 9632 conexiones), el tiempo requerido de la simulación es considerable y amerita utilizar el endpoint `/run-py2pdevs-async`, que es no bloqueante. Este comando devuelve el identificador de la ejecución (`run_id`) que se puede usar para consultar el estado de la simulación con el endpoint `/runs/<run_id>/status`, que puede devolver el estado `running` o `completed`. Dado el número elevado de parámetros del modelo se especifica el archivo de parámetros `multi_SIR.params`.

```
curl -X POST "http://networksir.exp.dc.uba.ar:5000/proxy/simulator/rebuild"
curl -X POST "http://networksir.exp.dc.uba.ar:5000/proxy/simulator/set-
    py2pdevs-model -F "model=@multi_SIR.py"
curl -X POST http://networksir.exp.dc.uba.ar:5000/proxy/simulator/run-
    py2pdevs-async -H "Content-Type: application/json" -d "{"tf": "1000","c":
    "multi_SIR.params"}"
curl -X GET -d " " "http://networksir.exp.dc.uba.ar:5000/proxy/simulator/get-
    py2pdevs?run=<id>&file=multi_SIR_1.h5" -o multi_SIR_1.h5
```

Se simularon 1000 s de tiempo virtual en las tres configuraciones (Tabla 1). A diferencia de los casos de Lotka-Volterra, este modelo de gran escala está dominado por el cómputo y exhibe diferencias marcadas: 34.1 s (*Local*), 365.1 s (*Servidor dedicado*) y 111.0 s (*AWS EC2*).

Estas diferencias se explican por las características de cada entorno de ejecución. Dado que PowerDEVS ejecuta en un único hilo, el factor determinante es el desempeño mono-hilo del procesador. El equipo *Local*, con una CPU de

generación reciente (AMD Ryzen 5 9600X), obtiene el mejor tiempo, seguido por la instancia *AWS EC2* `t3.large`. El *Servidor dedicado*, pese a contar con 125 GB de RAM, resulta el más lento debido a su CPU Intel Xeon Silver 4208, de una generación anterior, y al sobrecosto de la doble virtualización (Docker sobre LXD). Si bien la ejecución *Local* es la más rápida, requiere que el usuario instale y configure el Servidor de Simulación, lo que excede las expectativas de la mayoría de los potenciales usuarios.

6 Conclusiones y Trabajo a Futuro

Este trabajo presentó *PowerDEVS-Cloud*, una plataforma que permite ejecutar simulaciones remotas de modelos de PowerDEVS sin necesidad de instalación de software local. Las capacidades de *PowerDEVS-Cloud* se validaron mediante tres casos de estudio, incluyendo modelos desarrollados con la GUI de PowerDEVS y con py2PowerDEVS. Este trabajo extiende las capacidades de PowerDEVS hacia ejecución en la nube ampliando sus campos de aplicación y haciéndolo más accesible.

Los resultados muestran que, si bien la ejecución local ofrece el mejor desempeño, las arquitecturas remotas permiten ampliar el acceso a la herramienta a potenciales usuarios sin requerimientos técnicos avanzados. En particular, el caso de múltiples modelos SIR interconectados demostró la utilidad de contar con mecanismos de ejecución no bloqueante para simulaciones de larga duración.

Se abren varios caminos para explorar como trabajo a futuro. La API REST desarrollada puede ser integrada en distintos flujos de trabajo sin demasiada complejidad. Una extensión natural es el desarrollo de herramientas de visualización de los resultados. La gestión de múltiples usuarios se realiza bajo la hipótesis de que no existen amenazas de alteración o borrado de resultados. No obstante, la seguridad es un aspecto fundamental en MSaaS (Cayirci, 2013) y será foco de trabajo a futuro. Un paso más adelante sería el desarrollo de un *frontend* visual (basado en la API) que facilite la interacción con la plataforma.

References

Al-Zoubi, K., & Wainer, G. (2013). Rise: A general simulation interoperability middleware container. *Journal of Parallel and Distributed Computing*, *73*(5), 580–594.

Al-Zoubi, K., & Wainer, G. (2015). Distributed simulation of devs and cell-devs models using the rise middleware. *Simulation Modelling Practice and Theory*, *55*, 27–45.

Bergero, F., & Kofman, E. (2011). PowerDEVS: A tool for hybrid system modeling and real-time simulation. *Simulation*, *87*(1-2), 113–132.

Byrne, J., Heavey, C., & Byrne, P. J. (2010). A review of web-based simulation and supporting tools. *Simulation modelling practice and theory*, *18*(3), 253–276.

Capocchi, L., & Santucci, J.-F. (2021). A web-based simulation of discrete-event system of system with the mobile application devsimpy-mob. *SoftwareX*, *13*, 100625.

Castro, R., Kofman, E., & Wainer, G. (2010). A Formal Framework for Stochastic Discrete Event System Specification Modeling and Simulation. *SIMULATION*, *86*(10).

Castro, R., Bergonzi, M., Pecker-Marcosig, E., Fernández, J., & Kofman, E. (2024). Discrete-event simulation of continuous-time systems: Evolution and state of the art of quantized state system methods. *SIMULATION*, *100*(6), 613–638.

Cayirci, E. (2013). Modeling and simulation as a cloud service: A survey. *2013 Winter Simulations Conference (WSC)*, 389–400.

Cho, Y. K., Hu, X., & Zeigler, B. P. (2003). The rtdevs/corba environment for simulation-based design of distributed real-time systems. *SIMULATION*, *79*(4), 197–210.

Fishwick, P. (1996). Web-based simulation: Some personal observations. *Proceedings Winter Simulation Conference*, 772–779.

Mittal, S., Risco-Martín, J. L., & Zeigler, B. P. (2009). Devs/soa: A cross-platform framework for net-centric modeling and simulation in devs unified process. *Simulation*, *85*(7), 419–450.

Mittal, S., Risco-Martín, J. L., & Zeigler, B. P. (2007). Devsml: Automating devs execution over soa towards transparent simulators. *SpringSim (2)*, 287–295.

Pecker-Marcosig, E., Bonaventura, M., Lanzarotti, E., Santi, L., & Castro, R. (2022). Py2powerdevs: Construction and manipulation of large complex structures for powerdevs models via python scripting. *2022 Winter Simulation Conference (WSC)*, 2594–2605.

Shahin, M., Babar, M. A., & Chauhan, M. A. (2020). Architectural design space for modelling and simulation as a service: A review. *Journal of Systems and Software*, *170*, 110752.

Van Tendeloo, Y., & Vangheluwe, H. (2017). An evaluation of devs simulation tools. *Simulation*, *93*, 103–121.

Vangheluwe, H. L. (2000). Devs as a common denominator for multi-formalism hybrid systems modelling. *Proc. of the 2000 IEEE International Symposium on Computer-Aided Control System Design (CACSD)*.

Wang, S., & Wainer, G. (2015). A simulation as a service methodology with application for crowd modeling, simulation and visualization. *Simulation*, *91*(1), 71–95.

Zehe, D., Knoll, A., Cai, W., & Aydt, H. (2015). Semsim cloud service: Large-scale urban systems simulation in the cloud [Special issue on Cloud Simulation]. *Simulation Modelling Practice and Theory*, *58*, 157–171.

Zeigler, B. P., Muzy, A., & Kofman, E. (2018). *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations* (3rd). Academic Press.