Plataforma de Simulación de Rendimiento de Aplicaciones para Desastres de Origen Natural

Una herramienta computacional para predecir el rendimiento de aplicaciones de una plataforma desplegada a nivel nacional

Proyecto FONDEF ID 15 | 20560

Plataforma de Apoyo a la Gestión de Emergencia y Aplicaciones

Plataforma de Simulación de Rendimiento de Aplicaciones para Desastres de Origen Natural
Una herramienta computacional para predecir el rendimiento de aplicaciones de una plataforma desplegada a nivel nacional

Proyecto FONDEF ID 15 | 20560
Plataforma de Apoyo a la Gestión de Emergencia y Aplicaciones

Primera edición: marzo 2021.

Contacto
https://citiaps.usach.cl
citiaps@usach.cl

Tabla de contenidos

1. Contexto y Antecedentes Preliminares

Los desastres naturales son situaciones extraordinarias que afectan a todo mundo [1,2], esto es apreciable desde diversas perspectivas, por ejemplo: vidas humanas perdidas y/o afectadas, pérdidas económicas, gasto público y privado en términos de recuperación y reconstrucción, etc.. Es por estos motivos que cualquier innovación que ayude en cualquiera de las etapas de un desastres natural (prevención, respuesta, recuperación y restauración), se cree será apreciada, y más aún, si es fácil de adoptar y adaptar respecto de la situación en curso.

Es bien sabido que Chile es uno de los países con tendencia a sufrir desastres naturales [3], que a lo largo de la historia cuenta con un gran número de este tipo de situaciones, y que de hecho fue afectado por el terremoto registrado más grande de la historia mundial [4]. Uno de los principales sucesos que afectan a la población en los desastres naturales es la destrucción o inhabilitación de infraestructura, que entorpece o anula las comunicaciones por desplome, caída o saturación de los servicios de telecomunicaciones [5, 6]. Esto fue notorio en el terremoto que afectó a Chile en Febrero del 2010, donde los niveles de insatisfacción de los usuarios respecto de los servicios de telecomunicaciones alcanzó un 57% [7]. Innovaciones como el desarrollo de sistemas computacionales que sean de utilidad en este tipo de situaciones, podrían ser de vital importancia en tareas tales como: la coordinación de equipos de trabajo, en la administración y gestión de los recursos disponibles, en la coordinación de voluntarios en los periodos de respuesta, restauración y recuperación, etc. La dificultad que surge entonces radica en la complejidad en términos de la implementación y despliegue de una plataforma computacional que pueda albergar este tipo de sistemas, los cuales deben ser diseñados para tolerar los niveles de usos por parte de los usuarios, que en situaciones de estrés (ejemplo: un desastre de origen natural) aumenta drásticamente [5, 6].

Es por estos motivos que el principal objetivo de este documento es “obtener una Metodología que permita generar Modelos de Predicción de Rendimiento para aplicaciones desplegadas sobre una infraestructura distribuida y cuya principal misión es ser de utilidad en Situaciones de Desastre de Origen Natural”. La finalidad es apoyar a los equipos de desarrollo de sistemas computacionales, ya sea en las etapas preliminares cuando están realizando en análisis y diseño de una solución, o en etapas posteriores del desarrollo de sistemas, en la mejora de cierto componente, ya que haciendo uso de esta

metodología podrían saber a priori (antes de implementar y desplegar) cual sería el rendimiento de la solución que están diseñando, basado en los niveles de usos que se esperan del sistema.

2. Despliegue de Escenario Experimental

Una de las primeras aristas abordadas en este proceso ha sido el despliegue del escenario experimental, ya que éste nos brinda el marco de trabajo y validación de los modelos de predicción generados, ver la Figura 2.1, donde muestra el escenarios experimental utilizado.

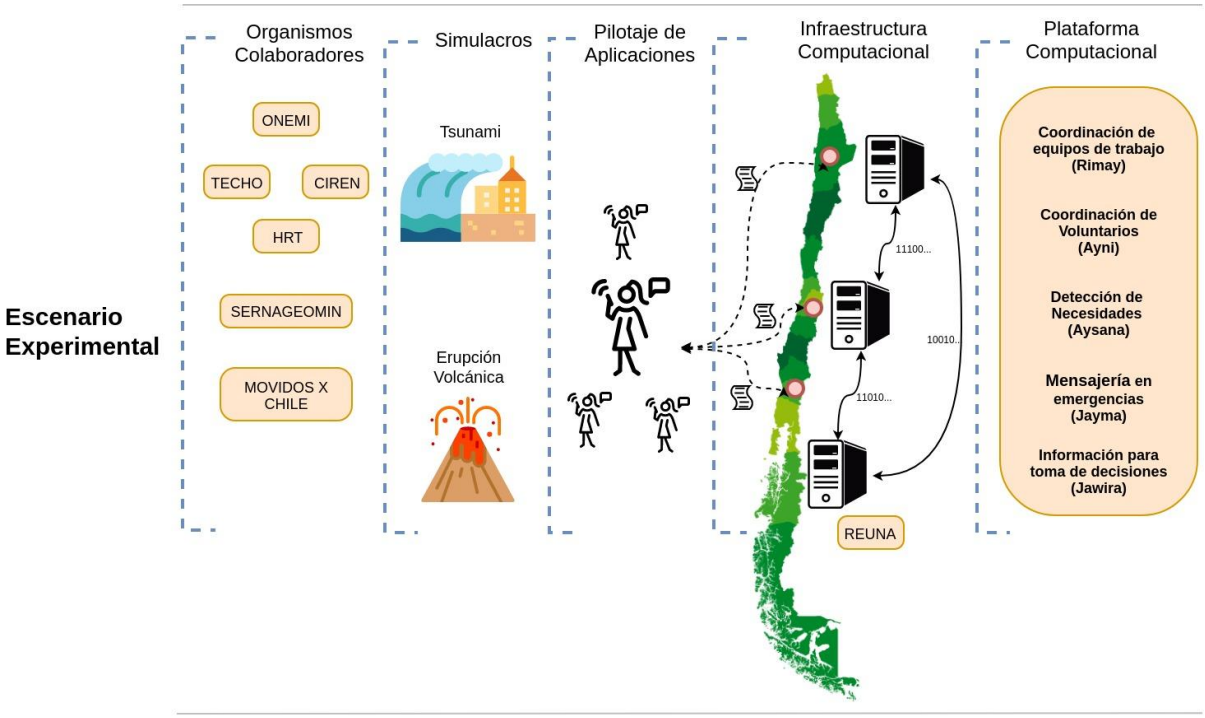


Figura 2.1,Escenario Experimental utilizado en el desarrollo de esta investigación.

2.1. Despliegue y pilotaje de aplicaciones

Para realizar las primeras mediciones de rendimiento computacional se están utilizando sistemas desarrollados por CITIAPS[1], el objetivo de estos sistemas es ser de utilidad en situaciones específicas durante alguna de las etapas de un desastre de origen natural, estos sistemas son: Rimay (coordinación de equipos de trabajo), Ayni (coordinación de voluntarios en zonas de desastre natural) y Jayma (asistencia a víctimas y reporte de estado del usuario). La especificación de las funcionalidades de los sistemas y el pilotaje de los mismos, han sido guiada por expertos que trabajan continuamente en el ámbito desastres de origen natural y en la coordinación de voluntarios. Gracias al soporte brindado por estas entidades se cree que los sistemas presentan funcionalidades válidas y robustas, que definen claramente las acciones a tomar frente a los desastres de origen natural. A continuación se describe el aporte de estas entidades en la definición y pilotaje de los sistemas.

- ONEMI[2] y SERNAGEOMIN[3], organizaciones gubernamentales que han ayudado en la definición del funcionamiento que debería presentar Rimay, Jayma y Ayni. Además, ONEMI ha colaborado con el pilotaje de las diferentes versiones de Rimay, versiones que han surgido de las diversas reuniones de trabajo entre CITIAPS y ONEMI, donde se han definido necesidades especificar por equipos de trabajo. El resultado de estos pilotajes nos ha servido para conocer el comportamiento de los usuarios frente al sistema, y el comportamiento del sistema respecto de los diferentes tasas de arribo de mensajes por parte de los usuario, datos esenciales para la replicación y validación de los modelos de predicción en los que se está trabajando.

- MovidosXChile[4] y TECHO[5], fundaciones de voluntariado sin fines de lucro, estas entidades han colaborado principalmente en la definición de los sistemas Ayni y Jayma, por su amplio conocimiento en el ámbito de la coordinación de voluntarios y gestión de recursos de ayuda humanitaria.

---

[1] "Citiaps." https://citiaps.usach.cl/. Se consultó el 24 marzo de 2021.
[2] "Onemi." https://www.onemi.gov.cl/. Se consultó el 24 marzo de 2021.
[3] "Sernageomin." https://www.sernageomin.cl/. Se consultó el 24 marzo de 2021.
[4] "Movidos x Chile." http://movidosxchile.cl/. Se consultó el 25 marzo de 2021.
[5] "Página de inicio | Chile - Techo." https://www.techo.org/chile/. Se consultó el 25 marzo de 2021.

- REUNA[6], NLHPC[7] entidades sin fines de lucro, encargadas de gestionar y coordinar el uso de recursos computacionales (cluster de computadores y/o Red de alta velocidad) para la investigación académica en Chile, estas entidades están brindado parte de la infraestructura donde se han desplegado las aplicaciones para realizar las diferentes pilotajes de los sistemas, junto a las pruebas de estrés y saturación.

- USACH, Universidad de Santiago de Chile, como institución académica ha facilitado cluster de computadores para realizar pruebas de estrés y saturación de los sistemas.

- CITIAPS, análisis de las necesidades planteadas por los mandantes y creador de las aplicaciones.

2.2. Simulacros de desastres naturales

Se han realizado dos simulacros de desastres de origen natural coordinados con ONEMI y por la intendencia de Iquique. En Iquique se realizó un simulacro de un Tsunami, con la participación de gran parte de la región, en este simulacro se realizó la prueba del sistema Jayma, este fue utilizado por 20 usuario con una traza de unos de 200 operaciones totales, que nos dan una perspectiva del comportamiento de los usuarios de este sistema. El siguiente simulacro fue un Terremoto en Valparaíso. También se probó el sistema Jayma, en conjunto con un sistema complementario, llamado “Traza de Movilidad”, el cual permite rescatar cada 5 minutos la ubicación geográficas de los usuarios, los datos rescatados provienen principalmente de estudiantes de la Universidad de Valparaíso, el sistema Jayma fue probado por 6 usuario, rescatando 20 operaciones de uso, se cree que fue bajo el uso del sistema, ya que al enterarse del simulacro, y de que no se darían todas las clases en la universidad, gran parte de los estudiantes no asistió. Por otro lado la aplicación "Trazas de Movilidad" logró capturar la trayectoria de los 20 participantes cuando se dirigían a los puntos de seguridad.

No se han utilizado Rimay y Ayni en ninguno de los simulacros antes descritos, se espera puedan existir futuros simulacros donde realizar las pruebas. De todas formas,

[6] "Chile - REUNA - RedCLARA." https://www.redclara.net/index.php/es/somos/miembros/chile-reuna. Se consultó el 25 marzo de 2021.
[7] "NLHPC." https://www.nlhpc.cl/. Se consultó el 25 marzo de 2021.

éstos si han sido probado en los pilotajes de expertos y se le han realizado benchmark planificados del uso de los sistemas.

2.3. Simulaciones Basadas en Simulacros

En una primera instancia se crearon benchmark planificados que han sido utilizados en el simulador como trazas sintéticas del usos de las aplicaciones. La finalidad de las trazas sintéticas es poder verificar el funcionamiento correcto del simulador, y de cada uno de sus componentes. Estas pruebas fueron realizadas de forma local (servidores en la Universidad de Santiago de Chile), para luego replicar dichas pruebas en los servidores facilitados por NLHPC y REUNA.

Posteriormente, luego de los pilotajes en terreno, se obtuvieron trazas reales de usuarios de las aplicaciones en situaciones de simulacro de evacuaciones. El siguiente paso es formatear las trazas para adaptar los datos a la entrada del simulador.

3. División de aplicaciones por patrón de diseño

La subdivisión de las aplicaciones por patrón de diseño es una de las instancias más relevantes de la construcción de la metodología de predicción de rendimiento, ya que es en esta sección donde se establecen las características que deben cumplir las aplicaciones a estudiar y cómo deben ser desplegadas en la plataforma computacional.

3.1. Características de las aplicaciones

Las aplicaciones que finalmente se utilizarán en el proceso de conformación de la metodología de predicción de rendimiento son las que obedezcan al patrón de diseño Cliente-Servidor, principalmente ya que nos da un marco de trabajo más generalizado, con lo cual en un futuro se podría trabajar caracterizando las aplicaciones del patrón Streaming. Estas características son:

- Deben ser fáciles de usar y deben ser capaces de tolerar grandes cargas de trabajo, es decir, deben presentar Frontend amigable e intuitivos y Backend robustos con características como escalabilidad, replicación y tolerancia a fallas.

- Los Frontend de uso masivo deben estar principalmente alojados en herramientas de uso cotidiano para los usuarios, es decir, deben ser complementos de sistemas como Facebook, Twitter, Telegram, Gmail, etc., de esta forma el uso de las aplicaciones por parte de los usuarios es más natural y no se salen de su zona de confort.

- Los sistemas deben estar separada en capas (Frontend - backend - Repositorio de Datos), y cada una de estas debe brindar servicios atómicos, es decir, que actúen como caja negra donde reciben datos de entrada, producen una salida y son fácilmente reemplazables, ya que solo basta quitar el componente y poner uno nuevo que se adapte al formato de entrada y salida del servicio.

- Las instancias de comunicación deben ser claramente identificadas gracias a la subdivisión de las aplicaciones en capas, estas deben ser Usuario-Frontend, Frontend-Backend y Backend-Repositorio de Datos. Existen instancias de comunicación adicionales, en las cuales el sistema implementado debe comunicarse con el sistema anfitrión (Facebook, Telegram, Twitter), el canal de comunicación habitual son las APIs (Interfaz de Programación de Aplicaciones) disponibilizadas por los sistemas anfitriones.

- No deben manejar estados en memoria, los estados del flujo de la aplicación deberían ser almacenados en algún repositorio de datos, de esta forma fácilmente se pueden identificar fallos y situaciones anómalas.

- El diseño de las aplicaciones o el despliegue de éstas, deben permitir la escalabilidad horizontal de sus servicios, es decir, si aumenta el número de réplicas de la aplicación, esta pueda atender un mayor número de solicitudes de ejecución.

- Las aplicaciones deben ser tolerantes a fallos, es decir, si un componente se cae o deja de funcionar, se debe levantar una nueva instancia para mantener estable el sistema.

- De existir replicación de componentes, estos deben estar sujetos a algoritmos de balance de carga, que distribuyan el trabajo de forma homogénea o dependiente de los recursos computacionales disponibles.

3.2. Descripción de las aplicaciones

A continuación de hace una breve descripción de cada una de aplicaciones desarrolladas por citiaps y evaluadas en este documento, destacando las principales características y arquitectura interna. Adicionalmente se describe Mongodb como repositorio de datos para la elaboración de los resultados.

- Ayni, aplicación encargada de gestionar las acciones que deben realizar grupos de voluntarios, estos voluntarios son congregados mediante un "BOT" de Telegram, implementado con el API disponibilizado por Telegram. Estos voluntarios se inscriben en la emergencia en curso e informan los requisitos con los que cumplen (físicos, habilidades, conocimiento, etc.), luego pasan por un proceso de selección automático. Finalmente, el voluntario puede participar en las tareas creadas producto de la emergencia. La arquitectura de este sistema está descrita en la Figura 3.1 y se distingue, un Cache, dos Backend y un repositorio de datos.

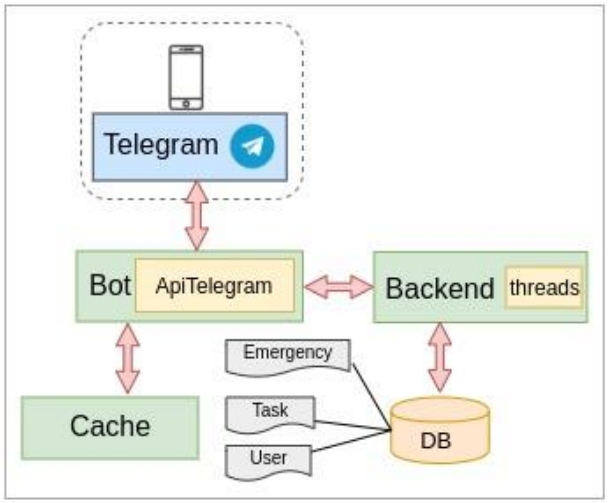


Figura 3.1, Arquitectura del sistema Ayni

- Rimay, aplicación encargada de gestionar la organización de equipos de trabajo. Esta aplicación presenta una arquitectura compuesta por un Frontend, dos backend, un repositorio de datos y un repositorio de archivos. El primer backend denominado “BOT”, implementa el API de Telegram para hacer un sistema embebido dentro de Telegram, el cual se encarga de congregar a los usuarios a la reuniones y notificar las tareas asignadas que deben realizar. El segundo Backend se encarga de la interacción entre el Repositorio de Archivos y el Repositorio de Datos, interacciones basadas en la administración del sistema. Además, presenta un Cache para el manejo de estados del sistema (Ejemplo: Reunión Iniciada). El detalle de esta arquitectura se aprecia en la Figura 3.2.

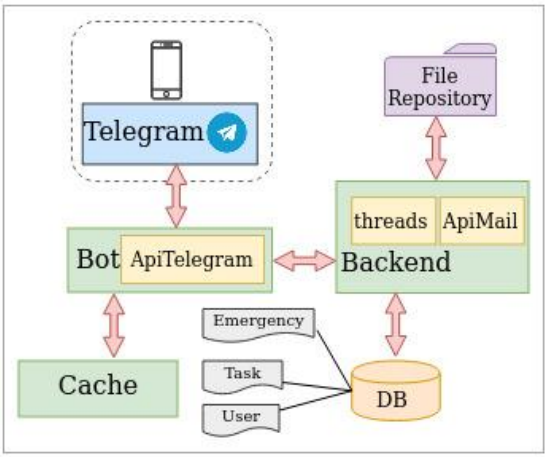


Figura 3.2, Arquitectura del sistema Rimay

- Jayma, aplicación destinada identificar el estado en el que se encuentran los usuario, y dependiendo de este estado se puede gestionar la asistencia a las víctimas, ya que junto al estado en el que se encuentra el usuario, este puede compartir su ubicación, con lo cual, si esta sufriendo por alguna situación, podría ser socorrido. La arquitectura de este sistema presenta un Cache, dos Backend y un repositorio de datos, ver Figura 3.3. Donde el backend que implementa el BOT hace usos de el API de Facebook.

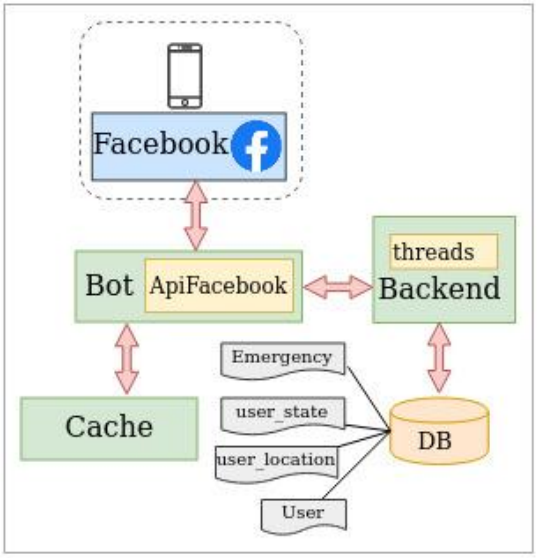


Figura 3.3, Arquitectura del sistema Jayma

- Mongodb, Repositorio de datos o base de datos NoSql, basada en documentos (datos tipo JSON), y colecciones de estos documentos. Tiene la particularidad de ser escalable, replicable y tolerante a fallas. Sus principales componentes son MongoD, MongoS y el ConfigServer, estos componentes y sus conexiones pueden ser vistos en la Figura 3.4.

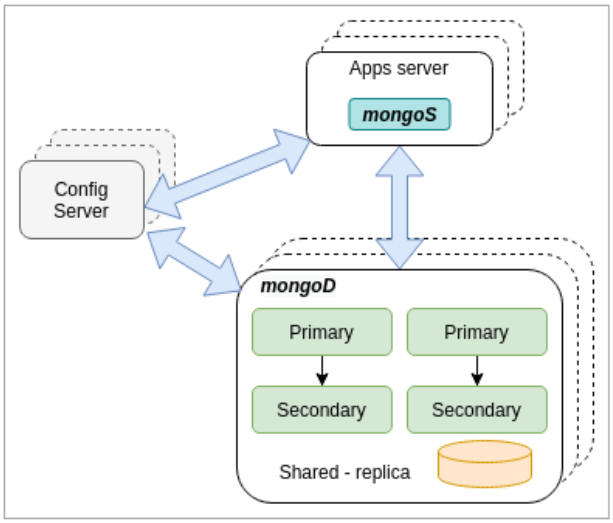


Figura 3.4, Componentes de MongoDB

3.3. Arquitectura de la plataforma

La arquitectura definida para el despliegue de las aplicaciones sobre la plataforma computacional está basada en la tecnología de containers y orquestación de containers. Containers que surgen con el objetivo de realizar el despliegue de aplicaciones independiente del sistema operativo, ya que encapsula las aplicaciones y sus dependencias en módulos portables que corren bajo el mismo núcleo de sistema operativo, evitando la necesidad de crear máquinas virtuales para aislar las aplicaciones. Además, la orquestación de containers permite coordinar de buena forma la escalabilidad horizontal, la replicación y la tolerancia a fallas de las aplicaciones desplegadas mediante containers, ver la Figura 3.5, se puede apreciar el diagrama de arquitectura para el despliegue de las aplicaciones sobre la plataforma computacional.

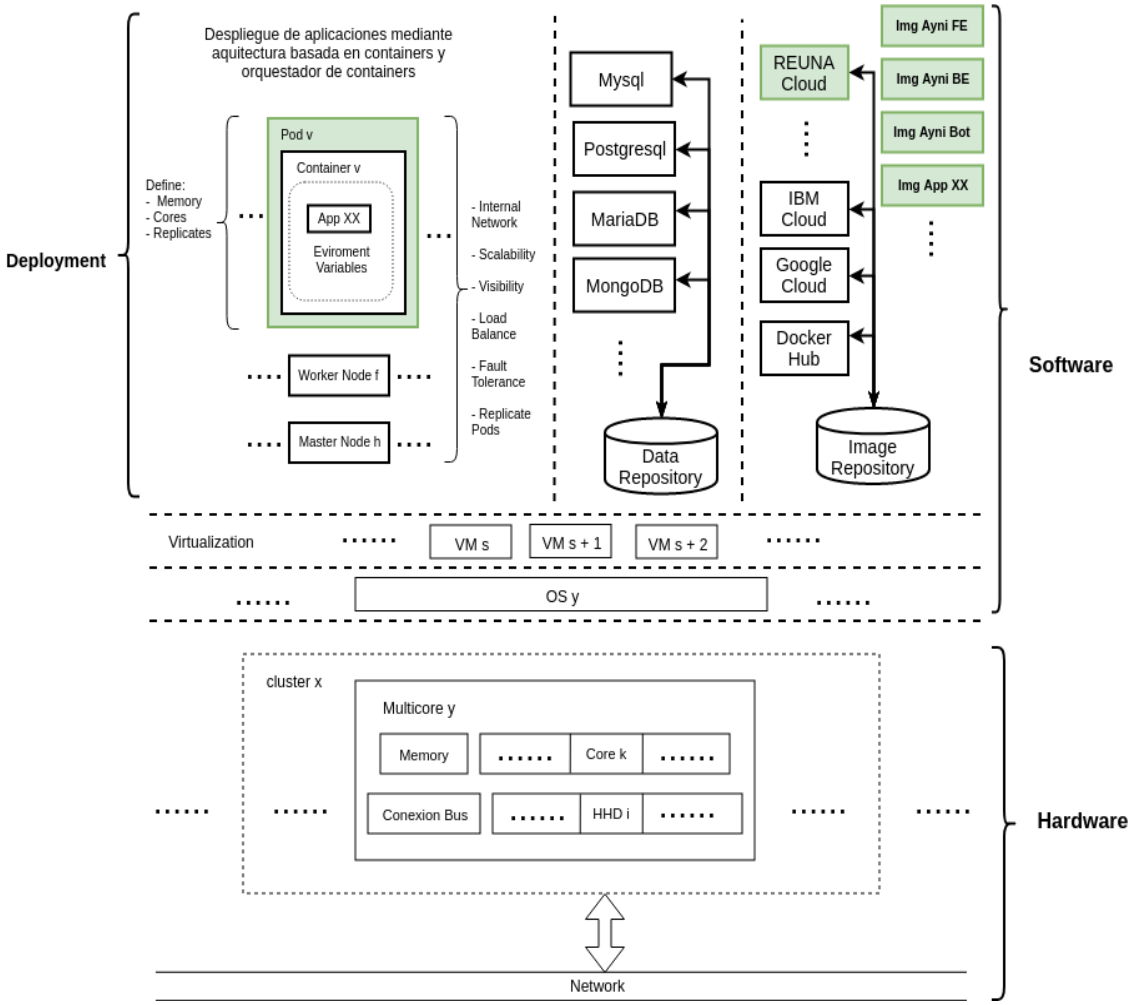


Figura 3.5, Arquitectura de despliegue de la plataforma computacional

Cabe destacar que los containers y el orquestador son tecnologías genéricas, de las cuales existen varias opciones para su implantación, en este caso puntual se ha seleccionado Docker Container para implantar la tecnología de container y Kubernetes para la orquestación de los containers, la principal razón por la cual fueron escogidos es que tienen un mayor respaldo de la comunidad de software libre, mejor documentación y de fácil integración.

4. Selección de características

Basado en la revisión de la literatura, ya sean artículos científicos o documentos técnicos de las tecnologías utilizadas, se han seleccionado un grupo de variables mediantes las cuales se medirá el rendimiento de las aplicaciones en la plataforma y que además cómo medio para validar el simulador, estas son:

- Por Sistema, Servicios de Sistemas y por los componentes de la arquitectura de despliegue (Docker, Pod, WorkerNode, MasterNode, VM, Server)
    - Tasa de arribo
    - Throughput
    - Tamaño de colas
    - Utilización (Core, Memoria, HDD) tiempo de utilización por ventanas
    - Tiempo medio de servicio (esto sirve para identificar los componentes cuellos de botella)
    - SLA, como el porcentaje de las peticiones respondidas bajo una cota de tiempo
    - Cantidad de servidores (nodos) ocupados, ya que la maximizar la utilización de los nodos se traduce en eficiencia energética
    - Latencia de comunicación
    - Tiempo medio de recuperación (tolerancia a fallas)

Todas estas variables serán rescatadas durante el proceso de simulación en ventanas de tiempo x distantes, con dicha información se obtiene el real uso de los recursos obtenidos por el simulador y se estima el rendimiento del sistema.

5. Modelo de predicción por Fase de Comunicación

El punto de partida para determinar los modelos de predicción de rendimiento por fase de comunicación es la separación de las aplicaciones en capas, estableciendo claramente los actores de cada una de estas fases, como ya se ha mencionado estas fases son: Usuario-Frontend, Frontend-Backend, Backend-Repositorio de Datos. Como segunda medida, se identificaron cada uno de los flujos de las aplicaciones estudiadas, para determinar elementos comunes, que han ayudado a estandarizar y/o generalizar los modelos, ver Figura 5.1, que muestra el flujo del caso de uso "/Ayuda" de la aplicación Ayni, donde cualquier usuario del sistema puede revisar las distintas opciones disponibles. Cada aplicación tiene un promedio de 20 flujos, donde cada flujo por lo menos tiene 7 tareas, con lo cual, sumado a las variables que ayudan a determinar la predicción del rendimiento del sistema y los componentes de la arquitectura para el despliegue de los sistemas, conforma un escenario de grandes dimensiones y posibilidades.

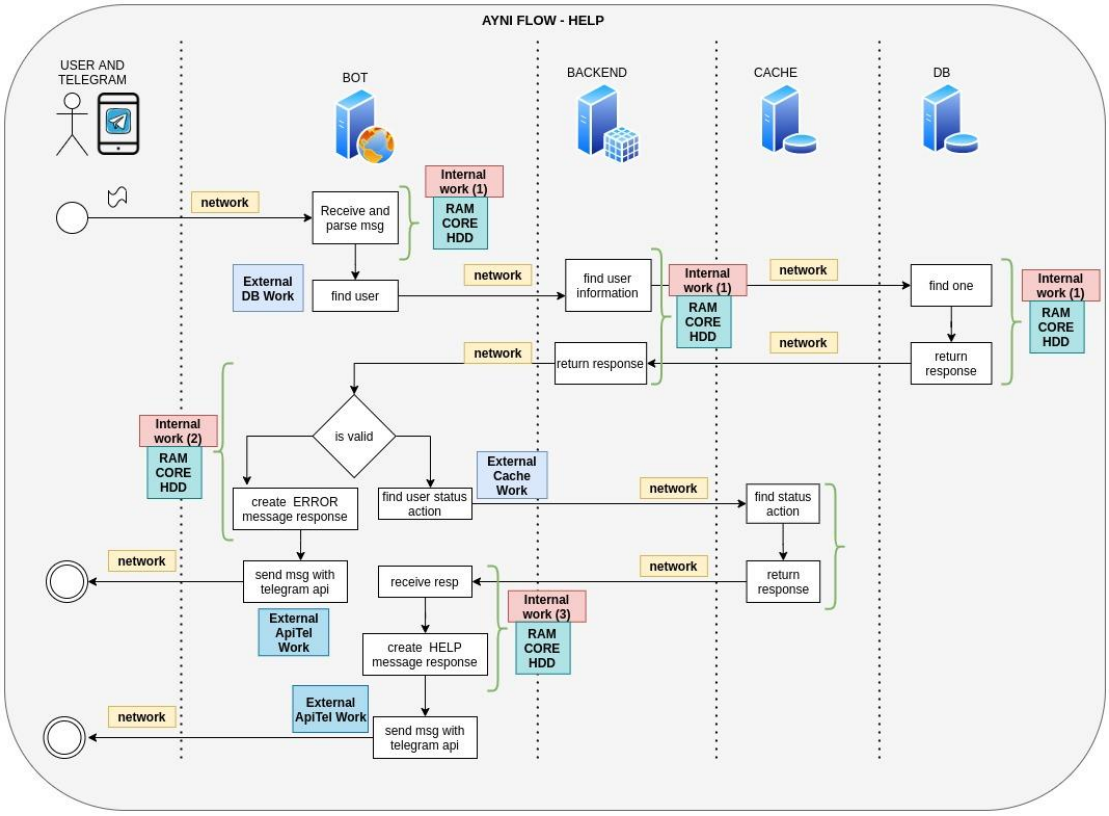


Figura 5.1, Sistema Ayni - Flujo del caso de uso "/Ayuda"

Los elemento comunes que se han identificado tienen directa relación con el componente que el sistema está utilizan para realizar sus tareas, y estos pueden ser de dos categorías: componentes internos, que se refiere a los recursos físicos o lógicos que son parte de la infraestructura donde están desplegados los sistemas (CPU, HDD, MEMORY, CORE, NETWORK), y componentes externos, que se refiere a aplicaciones externas o API que son utilizados para realizar alguna funcionalidad (Ejemplo: API de Telegram). Esto puede ser apreciado claramente en la Figura 5.1, que destaca los componentes utilizados por cada uno de los procesos o tareas de los flujos, además de realizar la separación por fase de comunicación.

La captura de datos para representar fielmente el comportamiento de los usuarios y el uso de los recursos computacionales utilizados por los sistemas ha sido obtenido mediante tres aristas, estas son:

- Pilotaje de aplicaciones, instancia en la que usuarios con experiencia en situaciones de desastre de origen natural han usado los sistemas, redefiniendo funcionalidades y validando el funcionamiento.

- Simulacro de desastres de origen natural, instancia que ha servido para capturar datos de trazas de movilidad y usos de los sistemas. Además, da la idea de cómo se comportaría la masa de usuarios en una situación de desastre de origen natural.

- Benchmark, se han realizado pruebas sintéticas de las aplicaciones, que han permitido analizar el comportamiento del sistema dependiendo de una alta demanda, de esta forma definir cuales son los parámetros de escalabilidad, replicación y tolerancia a fallas necesarios para que los sistemas creados sean de alta disponibilidad. Gracias a lo cual se han determinado elementos externos claves para construir sistemas que cumplan con las características antes mencionadas, estos son: Servidores DNS, Servidores Proxy, Servidores de Balance de Carga y Servidores de Cache.

De forma genérica los modelos de predicción de rendimiento de cada una de las fases de comunicación estará representado por un DAG (Grafo Acíclico Dirigido), cuyo nodos serán los elementos de procesamiento o uso de recursos (internos o externos) de cada aplicación y las aristas representarán el medio por el cual se están conectado dichos nodos, que puede ser un Bus de Conexión interna cuando se trata de aplicaciones

desplegadas en un mismo servidor, o la Red, cuando se trata de comunicación entre aplicaciones desplegadas en distintos servidores, ver la representación en la Figura 5.2, donde muestran la abstracción de modelo de predicción de rendimiento o DAG del flujo "/Ayuda" de la aplicación Ayni, donde las actividades y su conjunto de trabajos representan los vértices, y las flechas azules representan las aristas o instancia de comunicación.

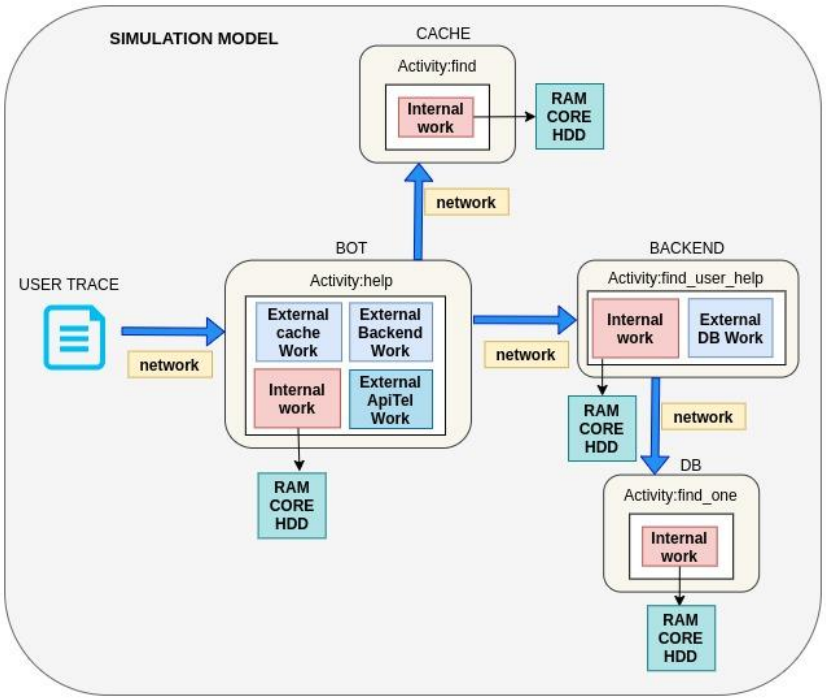


Figura 5.2, GADs por Fase de comunicación de "/Ayuda" - sistema Ayni

5.1. Selección de elementos de procesamiento

En primera instancia se identificaron los componentes genéricos que permiten la alta disponibilidad del sistema, estos son:

- Servidores DNS, aplicación encargada de enrutar las solicitudes de ejecución a los servidores basado en un Dominio Web. Se considera necesario, ya que puede ayudar en la tolerancia a fallas al poder establecer varios servidores destinos con un solo Dominio. Podría ser una aplicación interna del despliegue, como un servicio contratado.

- Servidores Proxy, sistema encargado del enrutamiento de las aplicaciones basado en una ruta Web, permite enmascarar los sistemas que generalmente corren en puerto distinto al 80 (http) o 443 (https), estableciendo una capa de seguridad.

- Servidores de Balance de Carga, aplicación que permite distribuir la carga de trabajo de un sistema. La tecnología utilizada basada en el orquestación de Containers tiene su propio balanceador de carga, el cual es utilizado basado en el número de réplicas establecidas.

- Servidores de Cache, servidor tipo llave-valor que permite entregar respuestas rápidas, sin necesidad de acudir al Backend, esto cuando las consultas realizadas son muy recurrentes.

- Componentes de la arquitectura de despliegue descritos anteriormente.

Para realizar esta selección de elemento se identificaron los componentes por fase de comunicación, los cuales son descritos a continuación:

- Fase de comunicación Usuario-Frontend
    - Traza de arribo de mensajes, la cual debe contener a la aplicación origen (identificador), aplicación destino (identificador), tipo de mensajes y el contenido
    - Latencia de comunicación
    - DAG de Frontend, el Flujo incluye: CPU, MEMORY propios del sistema y API de Mapas, usado como servicio externo.

- Fase de comunicación Frontend-Backend
    - Latencia de comunicación
    - DAG de Backend, el Flujo incluye: CPU, MEMORY, HDD propios del sistema y diferentes alternativas de API-Externo (Telegram, Facebook, Gmail), usados como servicio.

- Fase de comunicación Backend-Repositorio de Datos
    - Latencia de comunicación
    - DAG de Repositorio de Datos, el Flujo incluye: CPU, MEMORY, HDD

6. PLataforma de Simulación

La plataforma de simulación desarrollada tiene como objetivo simular el despliegues de una plataforma de aplicaciones para desastres de origen natural y permitir predecir el rendimiento que estas aplicaciones tendrán, ya sea en momentos de paz (donde no hay una situación extrema) y en los momentos de crisis, donde se sabe que el incremento de uso de aplicaciones crece exponencialmente. Dicha plataforma de aplicaciones está desplegada en un ambiente heterogéneo de clusters de servidores distribuidos a nivel nacional. En el diagrama, de la Figura 6.1, se plasman los componente generales de la plataforma.

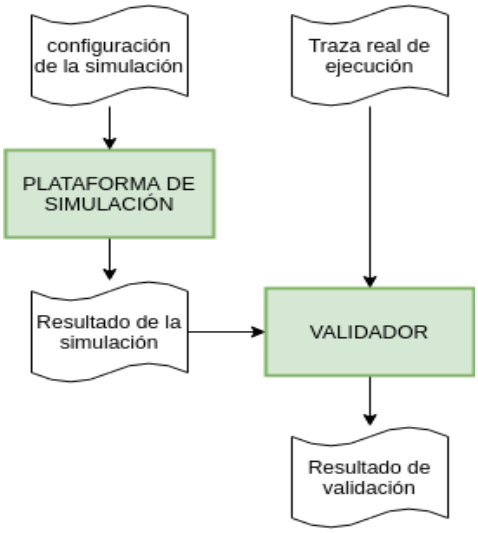


Figura 6.1, componentes generales de la plataforma de simulación

En la Figura 6.2 se realiza una especificación detalladas del modelo presentado en la Figura 6.1, donde se destaca la presencia de los parámetros de entrada al simulador, el simulador y una serie de sistemas que van moldeando los datos resultantes de la simulación hasta poder llegar a gráficos estadísticos y mapas de calor que permiten apreciar el comportamiento y rendimiento de los distintos componentes simulados, como por ejemplo: el uso de CPU, los mensajes recibidos, los mensajes enviados, el flujo de datos por la red, etc. Estas aplicaciones y parámetros se describen a continuación. Cabe destacar que la base de datos utilizada para almacenar los datos es **mongodb**, una base de datos noSql, orientada a documentos y colecciones de documentos, con lo cual permite una gran versatilidad y adaptabilidad en el uso y manejo de los datos resultantes de la plataforma de simulación.

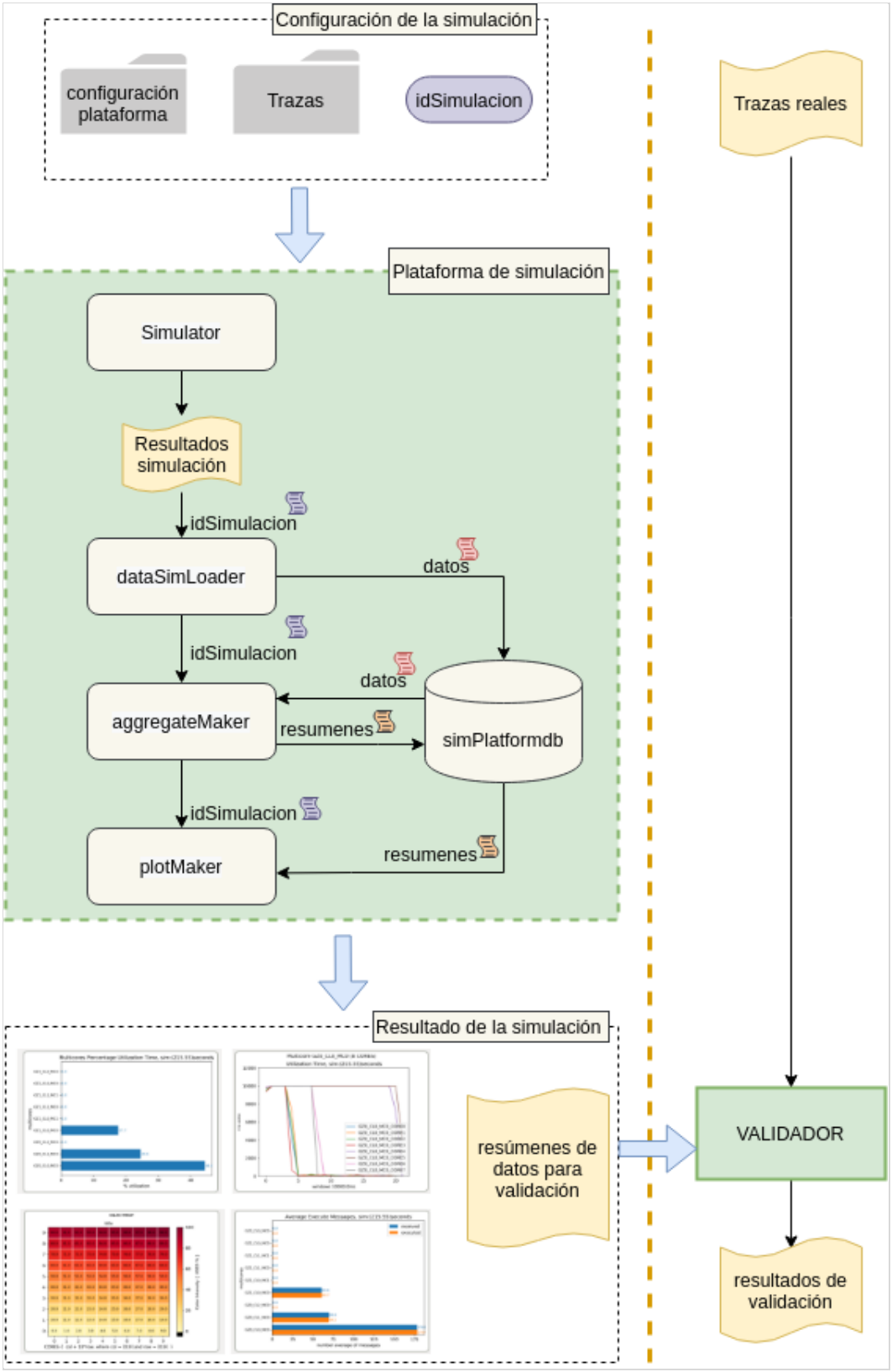


Figura 6.2, Plataforma de simulación

6.1. Parámetros de Entrada

Los parámetros de entrada se describen en tres piezas claves, estos son:

- **idSimulacion**, permite identificar la simulación y sus resultados a través de todo el flujo de la plataforma de simulación. Esto es importante, ya que cómo se ve en la Figura 6.2, la información resultante de las simulaciones es almacenada en una base de datos centralizada, gracias a esto se podrán realizar análisis conjuntos de múltiples simulaciones y sus resultados.

- **Trazas**, representan la información resultante del pilotaje de aplicaciones y reuniones con diferentes expertos, ya sea en despliegue de aplicaciones en clusters de computadores y expertos en situaciones de desastre de origen natural, lo que conforma datos válidos como parámetros de entrada. Las trazas se dividieron en dos tipos, que se describen a continuación y se pueden apreciar en la Figura 6.3. Además, en la Figura 6.4, donde se presenta la ubicación real en la estructura de carpetas de la plataforma de simulación.

    - Las trazas que permiten describir peticiones o uso de las aplicaciones, esto gracias al archivo JSON llamado “peticiones de usuario”. Donde los datos más relevantes a definir son:

        - El destino de la petición, es decir, la instancia de la aplicación que ejecutará la petición.

        - La actividad o acción a realizar sobre la aplicación, ejemplo: comando /Ayuda sobre el bot de Ayni, donde cada actividad está codificada según un conjunto de trabajo específicos por aplicación.

        - El identificador de la petición, con lo cual se le puede hacer el seguimiento completo a dicho mensaje, hasta saber en que CORE de que Servidor se ejecutó la solicitud.

        - El tamaño de petición en bytes

        - TimeHold, que especifica el tiempo en milisegundos en el que se debe lanzar la solicitud de ejecución de dicha petición, ejemplo: si se tienen tres peticiones consecutivas con TimeHold=0 y la cuarta tiene TimeHold=5, esto quiere decir que, las tres primeras peticiones se lanzarán de forma paralela dentro de la plataforma que se está simulando, y que la cuarta petición se lanzará 5 milisegundos después de haber lanzado las tres primeras.

    - las traza de “línea de tiempo”, que permite configurar caídas de componentes específicos de la plataforma computacional (ejemplo: el servidor 1 del Cluster de Arica en la Zona Norte del País se cae luego de 10 minutos luego del inicio el desastre de origen natural), esto según lo expresado por expertos en despliegue de aplicaciones, que comentan situaciones en las que los desastres de origen natural podrían causar problemas sobre equipos de cómputo y Red. Los datos más relevantes son:

        - El Id de componente, el cual puede representar un servidor, un cluster, un programa o una sección de red. En este caso lo ideal es correr la simulación, registrar los identificadores de los componentes que quiere simular con “estado=caída” y luego volver a correr la simulación con esa especificación, esto es así, ya que los identificadores se generan de forma dinámica para los programas.

        - El Status o estado del componente, que es un booleano, que indica caída o no caída.

        - TimeHold, que representa el tiempo en milisegundos cuando debe lanzarse el estado del componente dentro de la simulación en curso.

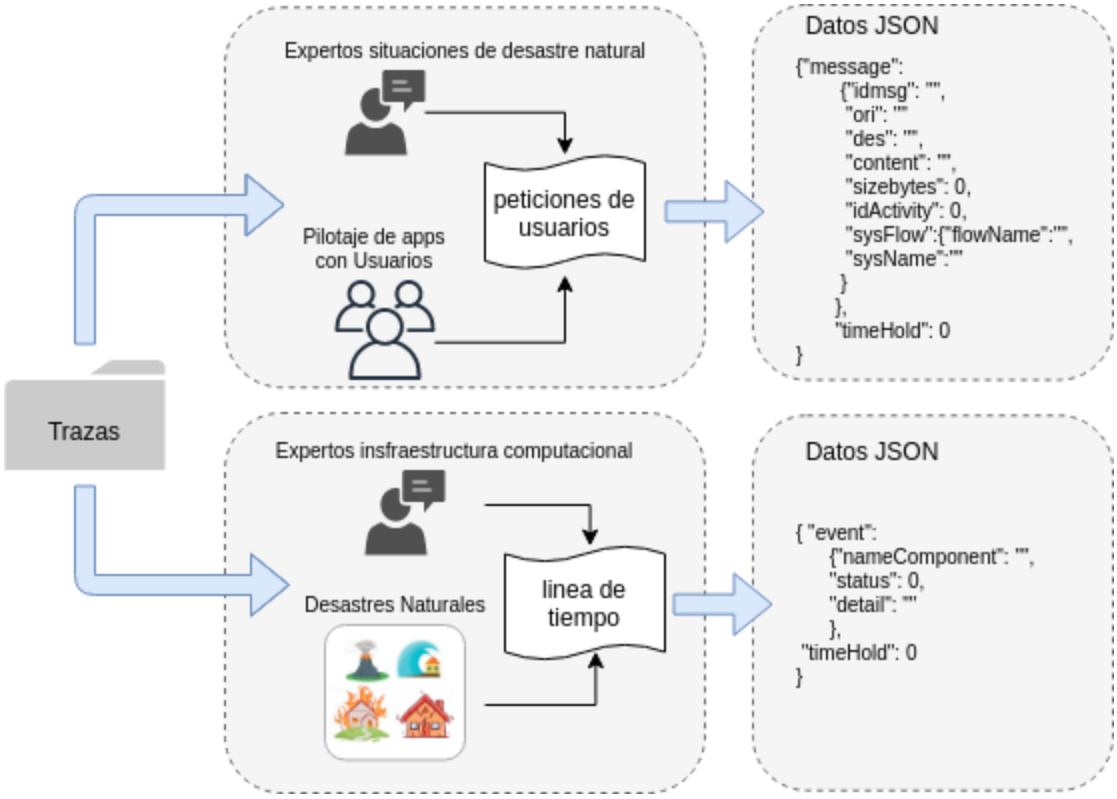


Figura 6.3, Trazas de entrada

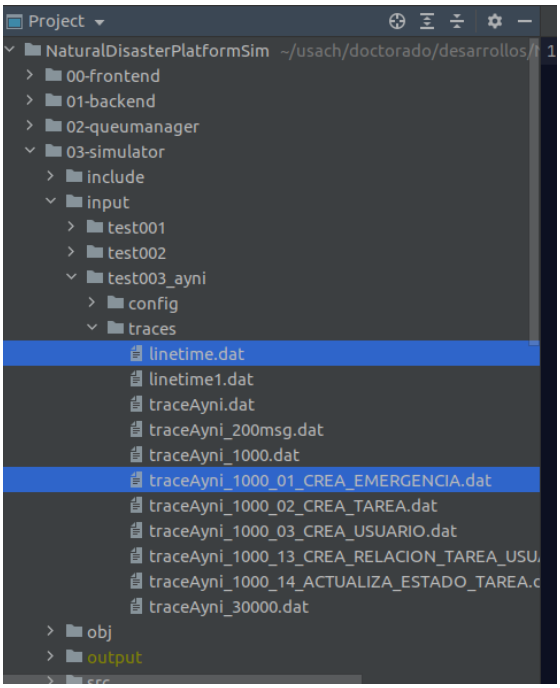


Figura 6.4, archivos con trazas para configurar una simulación

- Configuración Plataforma, este parámetro de entrada se subdivide en cuatro archivos JSON, los cuales describen la plataforma computacional de despliegue y el despliegue de aplicaciones a simular, estos archivos pueden ser vistos en la Figura 6.5. Además, en la Figura 6.6, donde se presenta el resumen de información más relevante almacenada en cada JSON. A continuación se describe cada uno de estos archivos y su contenido.

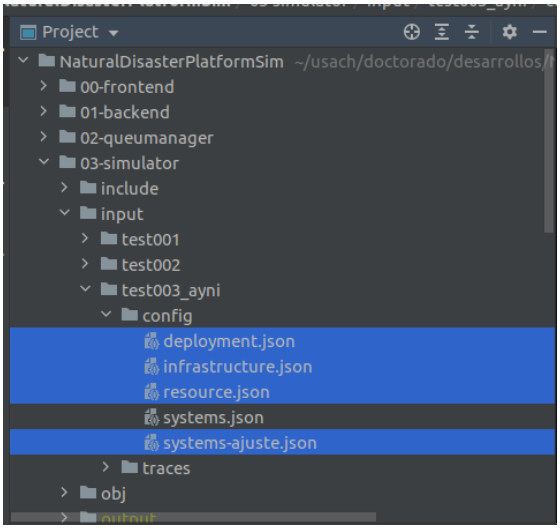


Figura 6.5, Archivos de configuración de la plataforma y el despliegue de aplicaciones

  - El JSON de “Recursos”, permite describir información esencial que será usada de forma genérica dentro del simulador, datos como por ejemplo:

    - El tipo de multicores y de máquinas virtuales que se desplegarán (grande, mediano y pequeño), entregando la especificación de las mismas (RAM, HDD y Cores). Necesarios para el archivo de “Infraestructura”.

    - Puertos prohibidos de usar dinámicamente, ya que son los puertos que se utilizan para las aplicaciones específicas, ejemplo el puerto de mongodb es 27017 y el de Cache es el 6379, que es el puerto utilizado por REDIS.

    - Prefijos de los nombres o identificadores que tendrá cada componente, por ejemplo: GZ-X- que es el prefijo para definir las

diferentes zonas geográficas que contendrán clusters de servidores, es usado al momento de generar dinámicamente los nombre de las zonas geográficas que se especifiquen según el JSON de "Infraestructura" (GZ0 ⇒ Zona Norte de Chile).

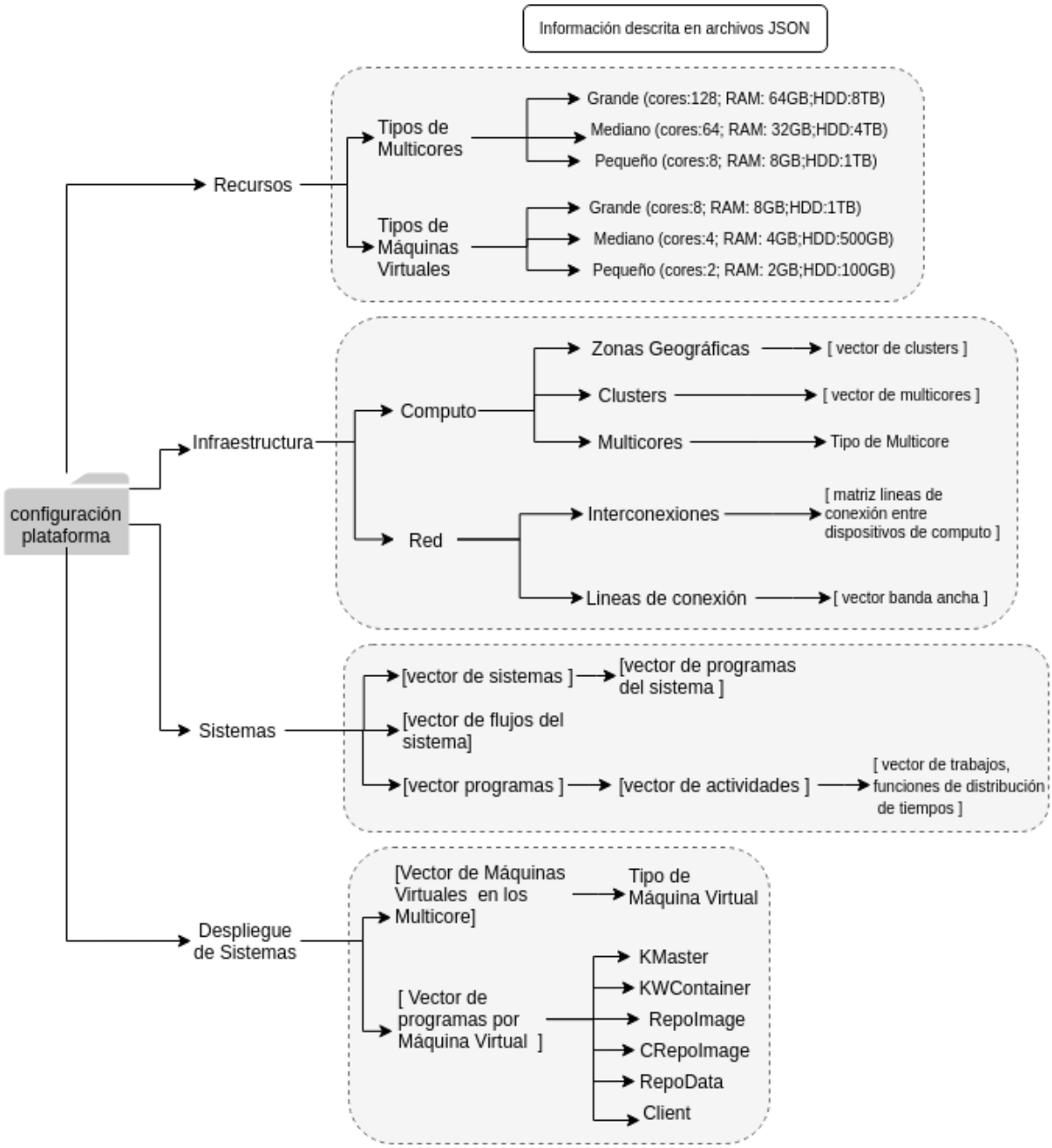


Figura 6.6, Configuración de parámetros de entrada

        - timesToCheckMasters y timesToCheckContainers, que representan los tiempos en milisegundos según los cuales Masters y Workers de kubernetes chequea el estado (caído o en funcionamiento) de los componentes con los que están conectados, de esta forma detectan si hay que levantar o no una nueva instancia. Ejemplo: el Master está orquestando el funcionamiento de un programa que tiene 3 replicas para atender un mayor número de peticiones, luego de un tiempo de ejecución, una de estas replicas se cae (se podría indicar con el archivo “linea de tiempo” de las trazas de entrada), entonces cómo el Master está constantemente monitoreando, detecta esta caída y levanta una nueva instancia del programa en alguno de los servidores sobre los cuales tenga el control de sus acciones, esto lo hace por tolerancia a fallas y con la finalidad de mantener la consistencia del sistema, todo lo descrito anteriormente ocurre dentro de la simulación de la plataforma.

    - El JSON de “Infraestructura”, se divide en dos secciones, la infraestructura de despliegue de cómputo (multicores y sus características, los clusters dónde están estos multicore y las zonas geográficas donde se encuentran estos clusters) y la infraestructura de red, que conectará la infraestructura de cómputo y permitirá el desplazamiento de los mensajes entre componentes. Los principales parámetros de este archivo son:

        - La definición de los servidores y sus IP estáticas.
        - La definición de los clusters de servidores.
        - La definición de las zonas geográficas y sus clusters.
        - La definición de las líneas de conexión y su velocidad de transmisión.
        - La definición de canales primarios y alternativos de conexión en caso de caídas.
        - La interconexión entre los diferentes clusters.

- El JSON “Sistemas”, que especifica el detalle de las aplicaciones a simular, donde se especifican los flujos de cada aplicación y se definen las funciones de distribución estadística que representa los tiempos de ejecución de cada trabajo interno de casa flujo de los sistemas. Algunos de los parámetros más relevantes son:

    - Los sistemas y los programas que los conforman, ejemplo Ayni: está conformado por: un bot, un backend, un cache, una base de datos y se conecta con Telegram mediante su API.

    - El detalle de los flujos de cada programa

    - El detalle de cada actividad de cada flujo de programa

    - El detalle de cada trabajo de cada actividad de cada flujo de programa, junto con las funciones de distribución estadística que representan los tiempos de ejecución en milisegundos.

    - La definición de la interconexión de programas mediante los endpoint, ejemplo: el bot de Ayni para conectarse con el backend de Ayni lo hace a través del endpoint de comunicación, que no es otra cosa más que la IP de la VM y el puerto donde está desplegado el backend de Ayni dentro de esa VM. Además, no hay que olvidar que el medio de comunicación es una representación de la RED, basada en la interconexión de los clusters descrita en el archivo de “Infraestructura”.

- Finalmente, el JSON de “Despliegue de Sistemas”, que describe el detalle de las VMs desplegadas por multicore y el detalle del despliegue de las aplicaciones sobre las máquinas virtuales. Algunos de sus parámetros son:

    - Las VMs desplegadas por multicore y a que tipo pertenecen, ejemplo: pequeña, mediana o grande, todos estos tipos definidos en el archivo “Recursos”.

    - Define el nombre del archivo de traza de peticiones a utilizar

- El despliegue de los programas sobre las VMs desplegadas y sus IP (si corresponde)
- El Masters y los Workers o Container que manejará
- Define qué programa queda desplegado en qué container

6.2. Simulador

El simulador se crea bajo el paradigma de Simulación de Eventos Discretos, según lo cual permite ir registrando de forma secuencial cada uno de los eventos que se están simulando. El simulador está implementado en C++ y en base a los templetes y estructuras provistas por la librería LibCppSim [8], ver Figura 6.7, que muestra los principales componentes del diseño conceptual del simulador. A continuación se describe cada uno de ellos:

- Network, red por la cual viajan los mensajes entre dispositivos de cómputo, define una serie de canales de transferencia dependiendo de su velocidad
- Channel, canal de comunicación que permite la transferencia punto a punto de mensajes
- Cluster, conjunto de uno a más servidores multicore
- Multicore, equipo de cómputo encargado de alojar a los sistemas desplegados en la plataforma computacional
- Core, unidad mínima de procesamiento encargada de estimar el tiempo de procesamiento de cada tarea
- RAM, unidad de almacenamiento temporal en un multicore
- HDD, unidad de almacenamiento definitivo en un multicore

- VirtualMachine, aplicación desplegada dentro de un multicore, simula un ambiente aislado de procesamiento y contempla Cores, Ram, HDD propios

- ProgramDeployment, programas desplegados sobre una máquina virtual, este contempla el ejecutable del programa más la configuración propia del despliegue caracterizado.

- Program, conjunto de actividades específicas, definidas respecto de cada programa a simular (Ejemplo: las actividades definidas para el programa Ayni).

- Activity, conjunto de trabajos específicos definidos para cumplir con una tarea particular (Ejemplo: los trabajos definidos dentro de la actividad “Crea Emergencia” en el programa Ayni).

- Work, mínima unidad de trabajo que permite calcular los recursos usados (Core, Ram, HDD) por el programa desplegado.

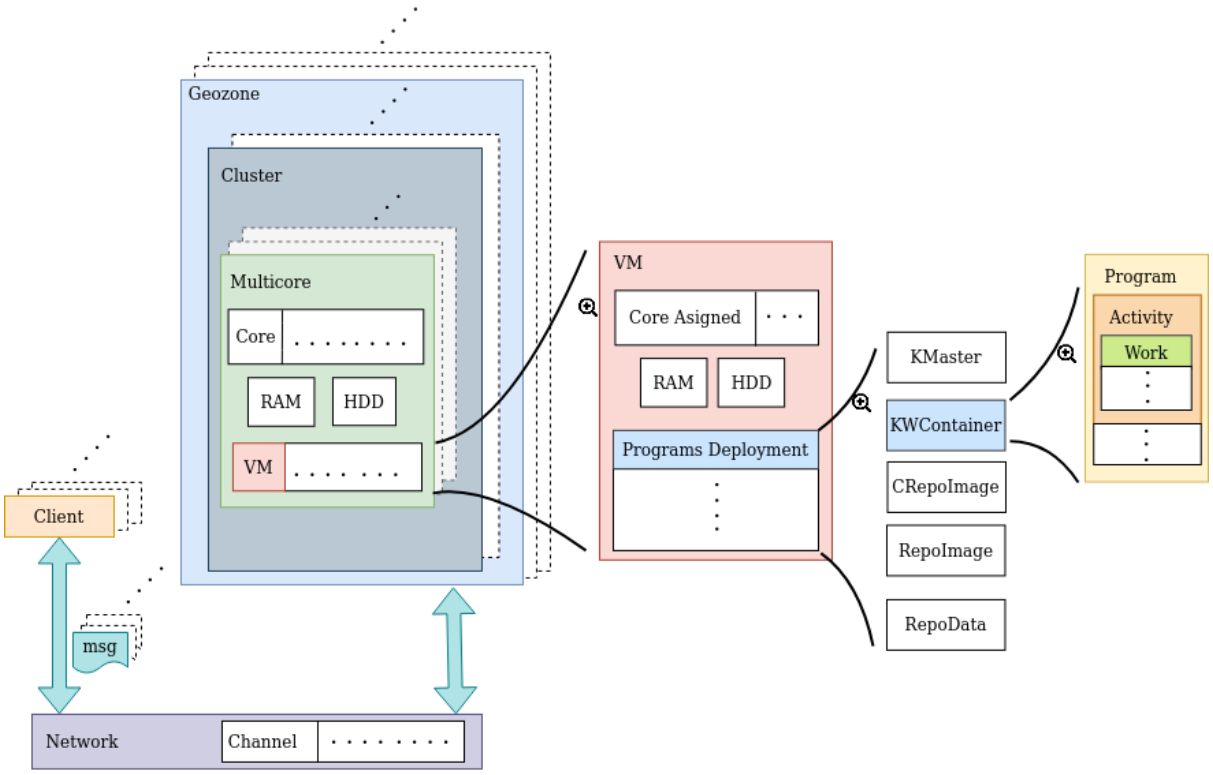


Figura 6.7, principales componentes del simulador

La lógica del funcionamiento del simulador es la siguiente, el cliente envía mensajes por la red al sistema. El cliente se implementa como una corrutina que se activa periódicamente para generar mensajes. La red se simula como un sistema de tubo que produce una demora en el sistema al enviar el mensaje. La demora que se simula depende del tamaño del mensaje y de la capacidad de transmisión de la red. La plataforma se simula como un conjunto de clusters distribuidos geográficamente. Cada cluster tiene un número limitado de multicores con diferentes características como velocidad de procesador, cantidad de memoria, disco duro, etc. Además cada multicore puede ejecutar varias máquinas virtuales (VM). Las VM tienen también recursos limitados en memoria, cantidad de cores y disco duro. Los programas que corren en una VM se despliegan utilizando diferentes componentes tecnológicos como containers. Y finalmente, cada programa es un conjunto de actividades, donde cada actividad posee un grupo de trabajos que debe realizar sobre un Core de un servidor.

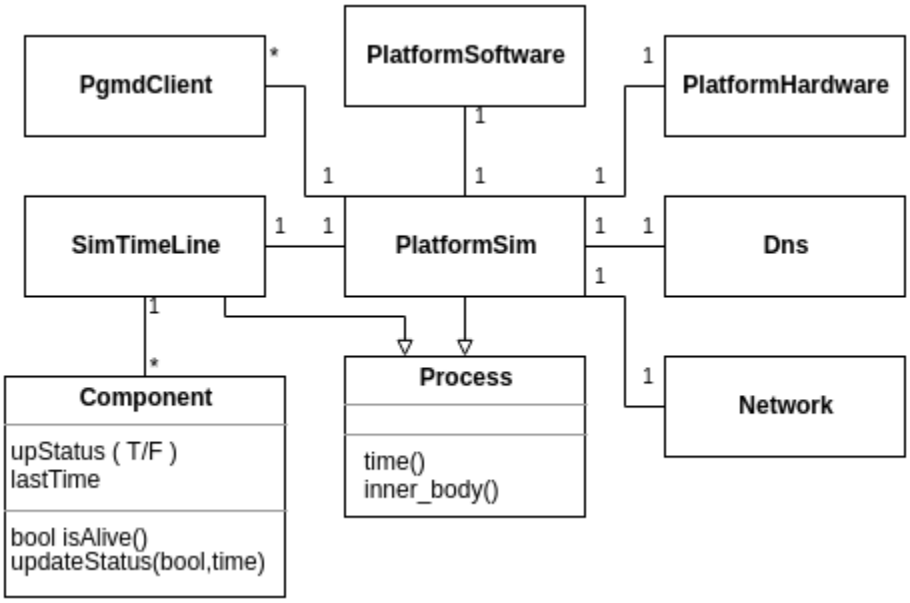


Figura 6.8, diagrama de clases general

La definición lógica general del simulador queda representada por el diagrama de clases, el cual puede ser visto en la Figura 6.8, con las clases nucleares del simulador y donde se destaca la presencia de la clase Process, que es propia de la librería de simulación libcppsim y tiene funciones específicas cómo:

- time(), que permite medir el avance del tiempo de la simulación
- inner_body(), que permite ejecutar las operaciones de hold()

- hold(), que simula los retardos de tiempo en el simulador
- passivate(), que desactiva proceso que no están siendo utilizados
- Entre otras.

De esta definición lógica del simulador surgen sus tres principales aristas que son: Network, PlatformHardware y PlatformSoftware. Estas se describen en los siguientes apartados.

6.2.1. Diagrama de clases de “Network”

En el diagrama de clases de los componentes de Red presentado en la Figura 6.9, se destaca la presencia de dos clases que dan los cimientos de la red, que son la clase “Message”, que permite encapsular los mensajes que viajan por todo el simulador, y la clase “ConnectionLine”, que describe un línea de conexión que une dos multicore con distintas posibilidades de conexión, representadas por el arreglo “BandWidth” (velocidad de conexión, que van ordenadas en forma decreciente en el arreglo) y los estados de estos “Funcionando” o “Caído”, ya que podría caerse un tramo de conexión, y esta podría seguir funcionando conectarse por la siguiente posibilidad en el arreglo (de existir), esta configuración surge luego de reuniones realizadas con expertos en despliegue de infraestructura de Red, REUNA.

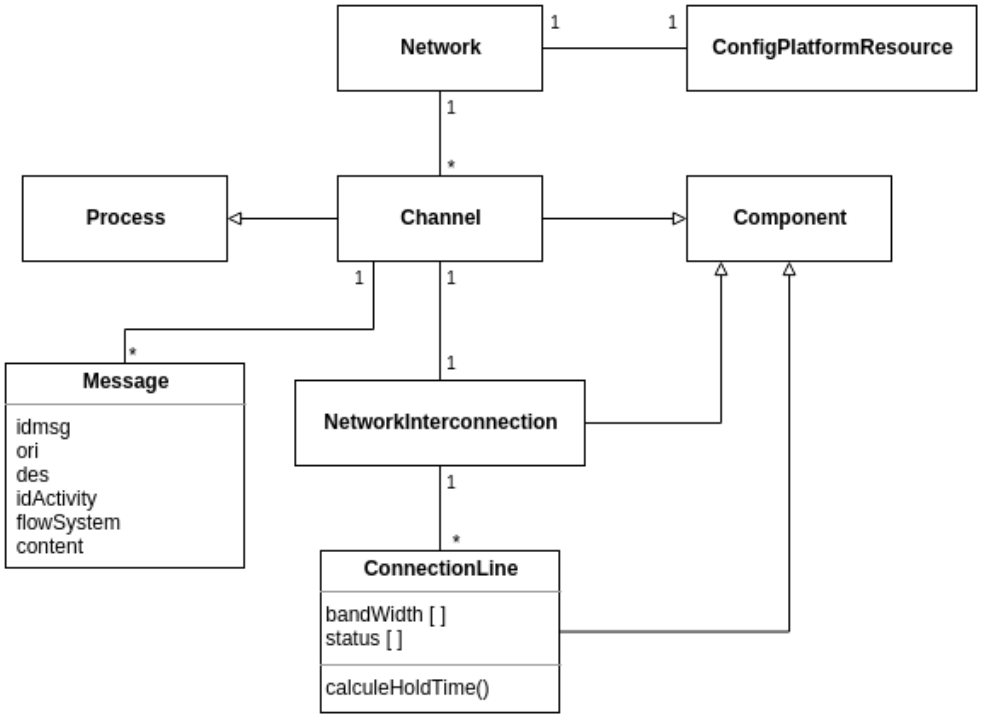


Figura 6.9, Diagrama de clases de componentes de Red

6.2.2. Diagrama de clases de “PlatformHardware”

Este diagrama de clases describe los principales componentes de la infraestructura cómputo implementada por el simulador, ver Figura 6.10 con el detalle del diagrama. Se pueden destacar la clase “Multicore”, como eje central, ya que contiene los Core donde se simula la ejecución de los procesos y contiene las máquinas virtuales, que es donde se despliegan las aplicaciones que se están simulando. Además, se puede destacar la clase “ExecutionRequest”, que son las solicitudes de ejecución que los programas desplegados le hacen llegar a los Core de los Multicore para ser procesadas.

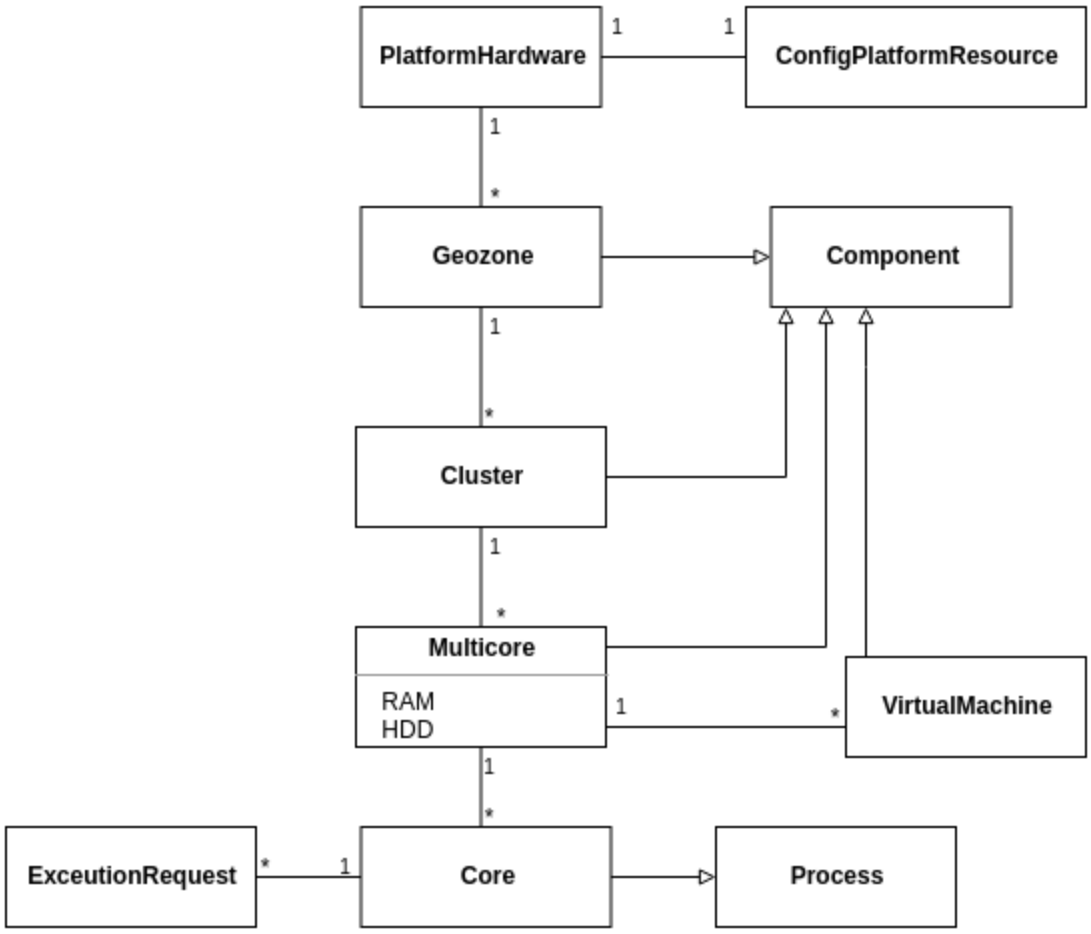


Figura 6.10, Diagrama de clases de los componentes de la infraestructura de cómputo

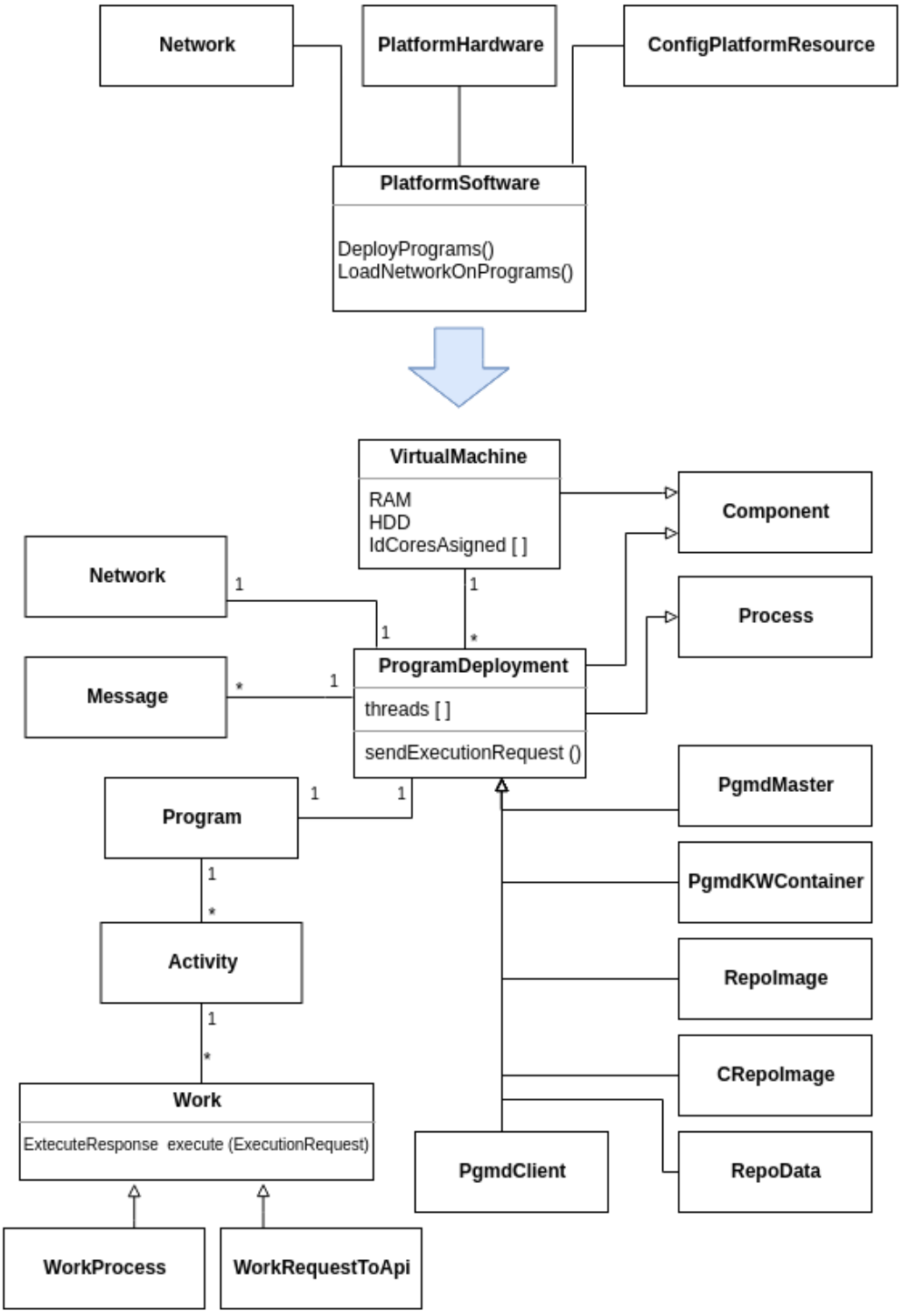


Figura 6.11, Diagrama de clases del despliegue de aplicaciones dentro de la infraestructura cómputo

6.2.3. Diagrama de clases de “PlatformSoftware”

Este diagrama describe el despliegue de aplicaciones dentro de la infraestructura de cómputo, y cómo se puede apreciar el la Figura 6.11 se divide en dos partes. la primera parte, tiene como eje central la clase “PlatformSoftware”, ya que sus tareas principales son realizar el despliegue de las aplicaciones dentro de la infraestructura de cómputo y de asociar sobre cada aplicación desplegada la red de comunicación por donde viajan los mensajes. La segunda parte, detalle cómo las aplicaciones son desplegadas en las VMs, destacando como eje central la clase “ProgramDeployment”, que es la clase base para describir cualquier aplicación que se quiera desplegar, ya que permite enrutar los mensajes o solicitudes a los otros programas de ser necesario, de enrutar las solicitudes de ejecución a los Core asignados a la VM donde está desplegada. También, se destaca la presencia de la clase “Work”, que es la mínima unidad de trabajo que se puede ejecutar en el simulador, esta clase encapsula el tiempo de HOLD o retardo de ejecución en los Cores simulados.

6.2.4. Resultado del simulador

El resultado del simulador es el archivo **“idSimulacion.dta”**, donde cada línea es un JSON listo para ser ingresado a la base de datos de la plataforma de simulación, que cómo ya se ha mencionado en mongodb, ver Figura 6.12 que plasma la base de datos y sus colecciones. Estos JSON se van agregando al archivo a medida que se van disparando los eventos de la plataforma computacional simulada. Los JSON se dividen en dos grupos según el tipo de datos que manejan, estos son:

- “Infrastructure”, que registra cada componente de infraestructura de cómputo o red que se simulen en la plataforma, el objetivo es realizar el análisis posterior de los componentes de la infraestructura utilizada. Además, se permite registrar todos los sistemas que se utilizaran en la plataforma.

- “Simu_data”, que registra los eventos que van pasando en la plataforma simulada a través del tiempo de simulación, donde la información registrada va desde que aplicación recibe un mensaje, hasta el core donde se simula la ejecución de un proceso. Ver Figura 6.15, donde se ve la agrupación de los principales grupos de información registradas por el simulador.

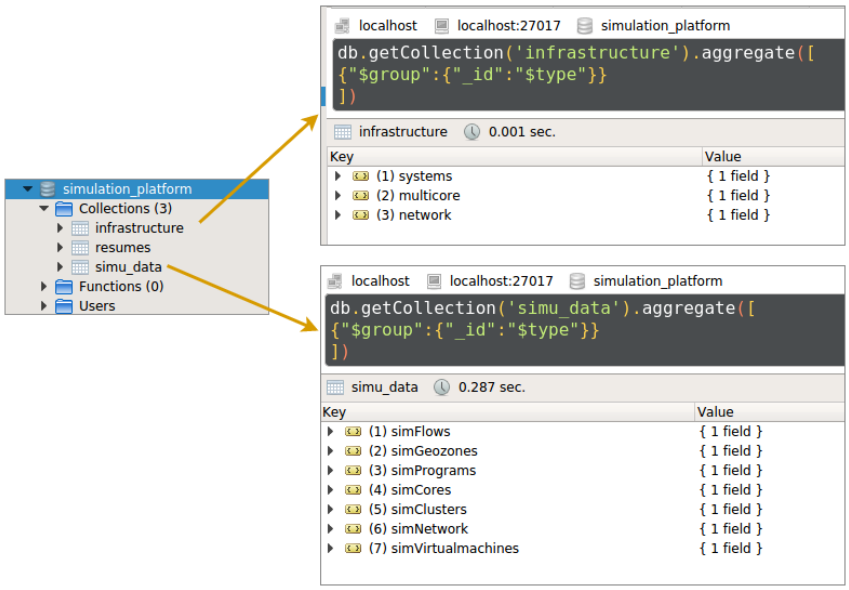


Figura 6.12: Base de datos de la plataforma de simulación

Con la información registrada se pueden analizar distintos escenarios, como por ejemplo: el largo de las colas y la espera promedio por mensajes en un periodo de tiempo determinado y los tiempos de ejecución de un flujo completo, todo este tipo de análisis se realizan con el programa “aggregateMaker”, que se describe en una de las siguientes secciones.

6.3. DataSimLoader

Aplicación desarrolladas en Python 2.7, encargada de tomar la salida del simulador y cargarla a la base de datos simPlatformDB (base de datos NoSql mongodb 4.0), para su posterior procesamiento. Los parametrada a esta aplicación son IdSimulacion y el path del archivo resultante de la simulación. Las colecciones creadas por este programa en la base de datos son:

- simu_data, que contiene todos los registros entregados a travé del tiempo de simulación.

- Infrastructure, que contiene la especificación de la infraestructura completa de la simulación, ya sea de cómputo o red.

6.4. AggegateMaker

Aplicación desarrollada en Python 2.7, encargada de realizar los resúmenes de información, esto lo realiza agrupando la información en ventanas de tiempo, lo que facilita la inspección y posterior visualización en los gráficos computacionales. Los resumes se crean basados los parámetros de entrada que son el **idSimulacion** y el tamaño en milisegundos de la ventana de tiempo que se utilizará para analizar la data obtenida, con esto se quiere decir que, si por ejemplo si se tiene ventana=300000 milisegundos y los datos de resultantes de la simulación están ordenados de forma creciente por tiempo, implicaría que cada 5 minutos sacaría un resumen para saber cómo se comportó la plataforma en cada intervalo de tiempo. Los resúmenes creados permiten analizar los distintos componentes de la plataforma computacional simulada, y sus datos quedan almacenados en la colección “resumes” de la base de datos de la plataforma de simulación. Los resumes tienen relación con:

- Core, saber cuanto tiempo estuvo utilizado el core ejecutando proceso
- Programa, saber cuantos mensajes logró ejecutar
- VM, cuantos programas tiene desplegados y cuánto procesos ejecutó
- Multicore, cuantas VMs tenía desplegadas y cuántos recursos fueron utilizados
- Red, cuantos mensajes viajaron por la red y cuál fue el ancho de banda utilizado
- Entre otros.

6.5. plotMaker

Aplicación desarrollada en Python 2.7, encargada de tomar los resúmenes de información, realizar gráficos estadísticos y mapas de calor por multicore, para representar el uso de los recursos dentro de la plataforma computacional simulada. Estos gráficos los realiza gracias a la librería matplotlib 3.2 de Python. A continuación se presentan una serie de Figuras generadas por este programa.

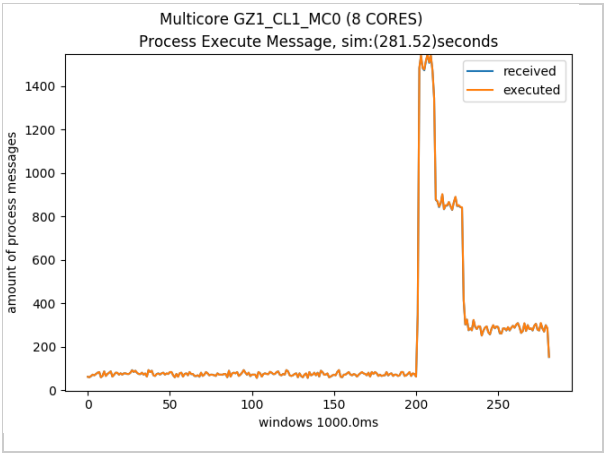


Figura 6.13, Solicitudes de ejecución procesadas durante la simulación.

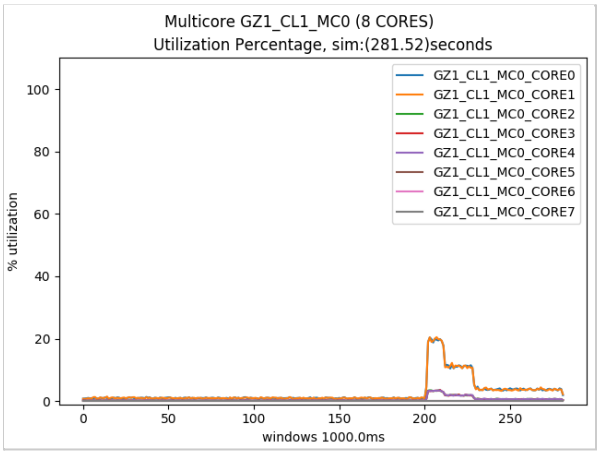


Figura 6.14, Porcentaje de uso de cada Core de un multicore durante toda la simulación.

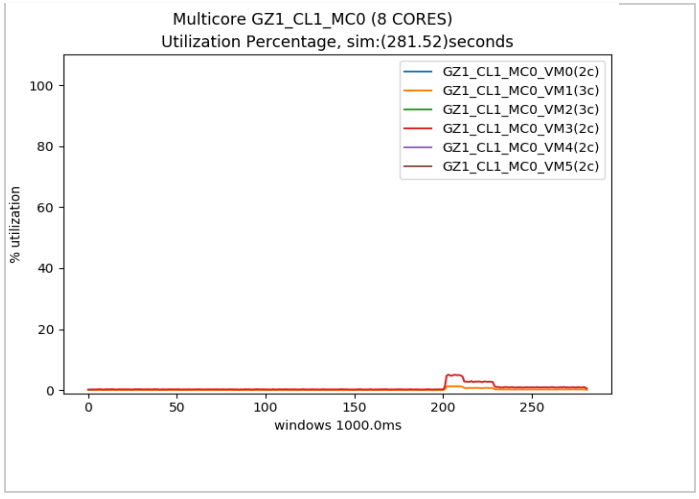


Figura 6.15, Porcentaje de uso de cada VM de un multicore

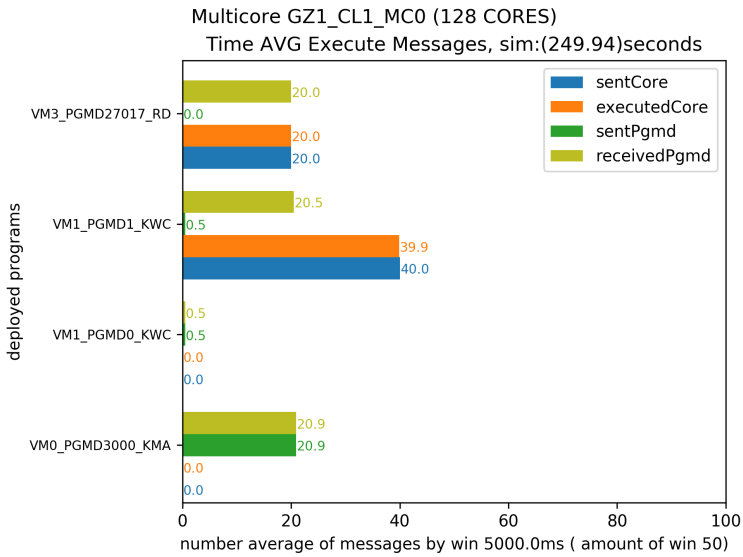


Figura 6.16, Tiempo promedio de ejecución de mensaje por programa desplegado sobre un multicore

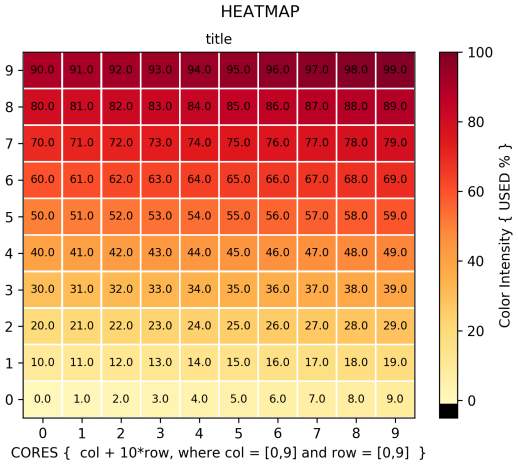


Figura 6.17, Mapa de calor de un multicore de 64 cores, donde la intensidad del color hace referencia al porcentaje de uso de cada core resumiendo toda la simulación

7. Resultado preliminares - Validación

En este apartado se describen los resultados preliminares obtenidos y la forma de validación de los componentes testeados. Se han realizado miles de pruebas sobre cada sistema real, el objetivo era determinar los puntos de saturación de cada componente y determinar la información de configuración para el simulador. Las pruebas consisten en la ejecución de cada uno de los principales flujos de cada uno de los sistemas, en la Tabla 1 y 2, se muestra el resumen de los resultados de estas pruebas sobre Ayni y Rimay, respectivamente, donde el tiempo promedio de las pruebas era entre 2 a 4 minutos y considerando que API-Externa puede ser Telegram o Facebook.

Tabla 1, resumen de resultados sobre sistemas reales sobre Ayni

Nombre del Flujo	Cantidades		Componentes involucrados				
	actividades	Pruebas	bot	backend	db	cache	API externa
Crear emergencia	1	300000		s	s		
Crear tarea	1	300000		s	s		
Crear voluntario	1	300000		s	s		
Ayuda	1	20000	s	s	s	s	s
Cancelar	1	20000	s	s	s	s	s
Inscripción	2	20000	s	s	s	s	s
Desinscripción	2	20000	s	s	s	s	s
Emergencias activas	4	20000	s	s	s	s	s
Mis emergencias activas	4	20000	s	s	s	s	s
Mis tareas activas	2	20000	s	s	s	s	s
Voluntario acepta tarea	1	300000		s	s		
Actualiza estado tarea	1	300000		s	s		

Tabla 2, resumen de resultados sobre sistemas reales sobre Rimay

Nombre del Flujo	Cantidades		Componentes involucrados				
	actividades	Pruebas	bot	backend	db	cache	API externa
Inscripción	3	15000	s	s	s	s	s
Admin acepta usuario	4	15000	s	s	s	s	s
Voluntario mira menú ayuda	1	15000	s	s	s	s	s
Admin crea evento	4	15000	s	s	s	s	s
Admin crea tarea	7	15000	s	s	s	s	s
Voluntario mira status	1	15000	s	s	s	s	s
Voluntario cancela acción	1	15000	s	s	s	s	s
Voluntario acepta tarea	4	15000	s	s	s	s	s
Voluntario termina tarea	4	15000	s	s	s	s	s
Voluntario realiza reporte	7	15000	s	s	s	s	s

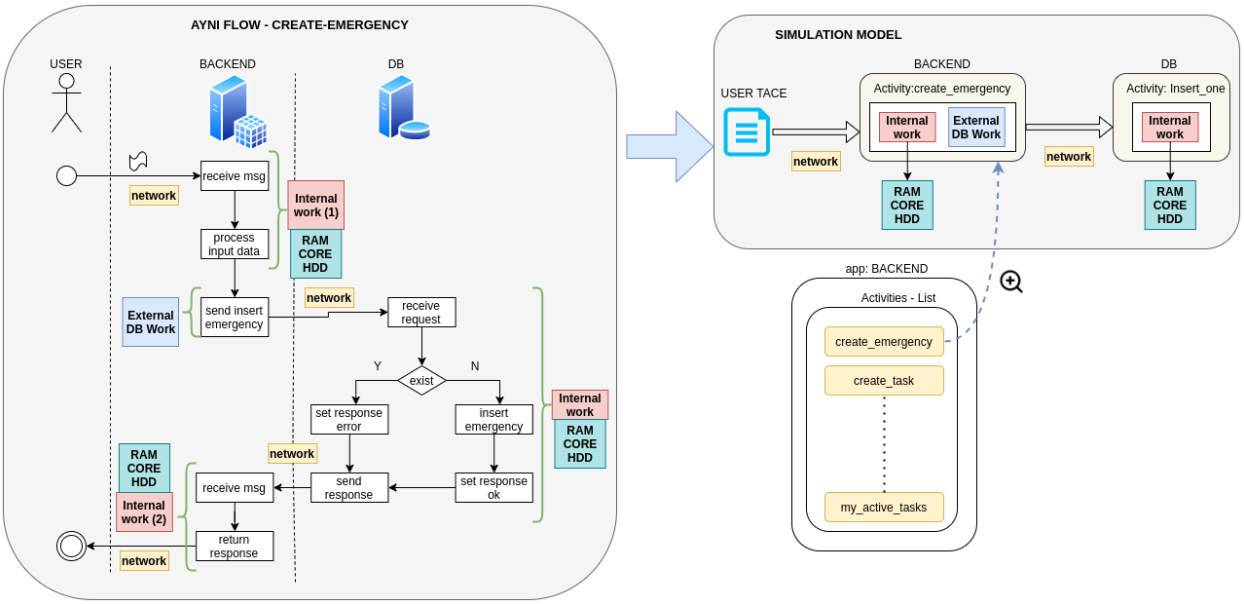
Cómo ya se mencionó, gracias a los valores obtenidos de la ejecución de pruebas anteriormente descrita, se ha logrado realizar la validación de cinco de los flujos de Ayni. La métrica de validación que se está usado es el porcentaje de error basado en la Raíz del Error Cuadrático Medio o RMSE (Root Mean Squared Error), cuya representación matemática se puede apreciar en la Ecuación 1, donde Ys representa el tiempos de simulación en milisegundos y Yr representa el tiempo real de ejecución de cada prueba del componente.

Ecuación 1

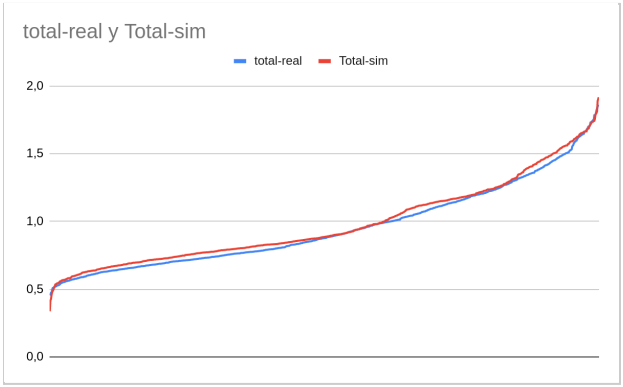
$$\%\, error \;=\; 100 \;*\; \frac{RMSE}{\overline{Yr}} = 100 \;*\; \frac{\sqrt{\frac{\sum_{i}^{n}(Yr - Ys\;\;)^{2}}{n}}}{\overline{Yr}}$$

A continuación presentan el detalle de los resultados:

- Resultados de “Crear emergencia” de Ayni, en la Figura 7.1 se presenta el flujo y su representación con el modelo de simulación interno. Además, la validación que alcanza un 3.4% de error, donde se realizaron 996 simulaciones y considerando en conjunto la operación de backend y base de datos, ver Figura 7.2, que presenta la curva de representación.

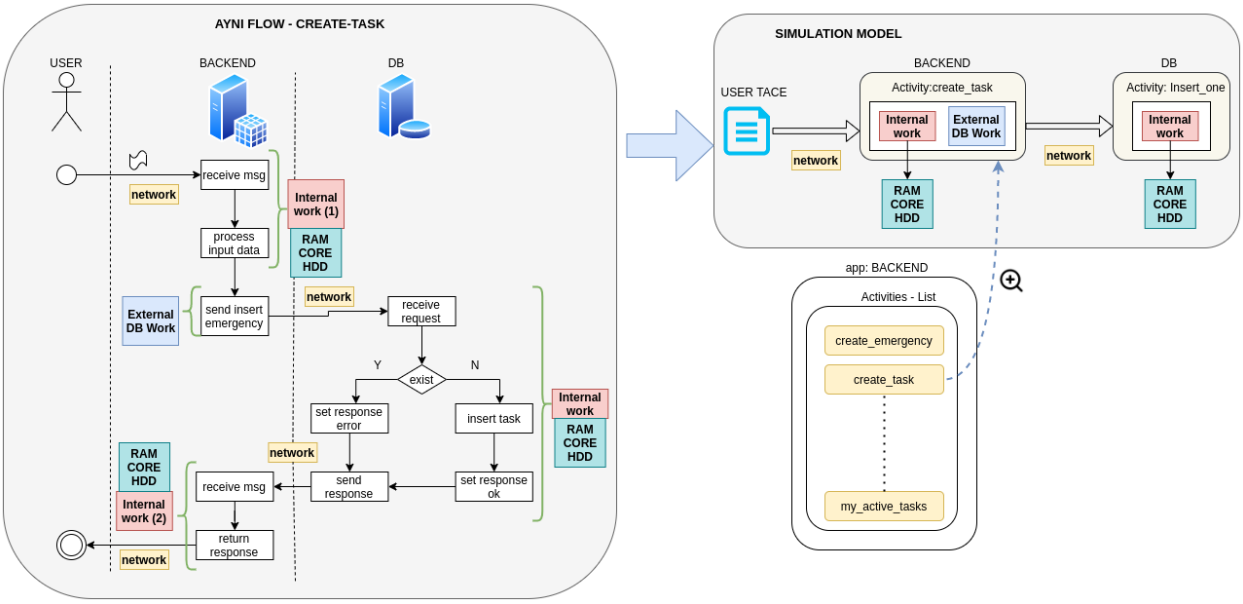


7.1 Flujo y modelo de simulación de “Crea emergencia” de Ayni

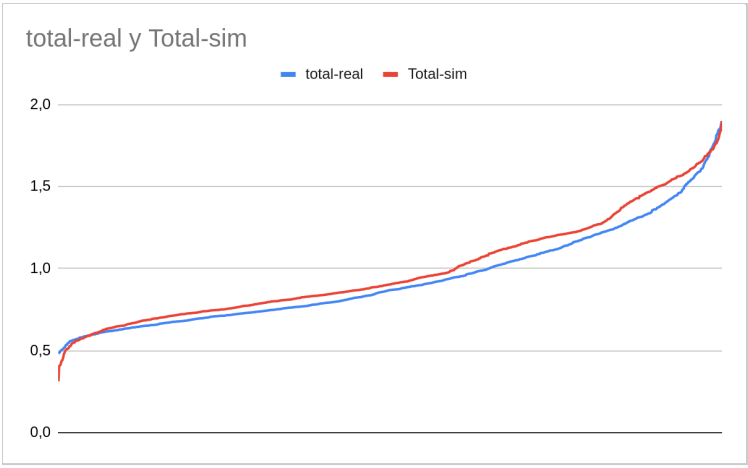


7.2: Curva datos simulación v/s real de la operación “Crea emergencia” de Ayni

- Resultados de “Crear tarea” de Ayni, en la Figura 7.3 se presenta el flujo y su representación con el modelo de simulación interno. Además, la validación que alcanza un 6.7% de error, donde se realizaron 996 simulaciones y considerando en conjunto la operación de backend y base de datos, ver Figura 7.4, que presenta la curva de representación.



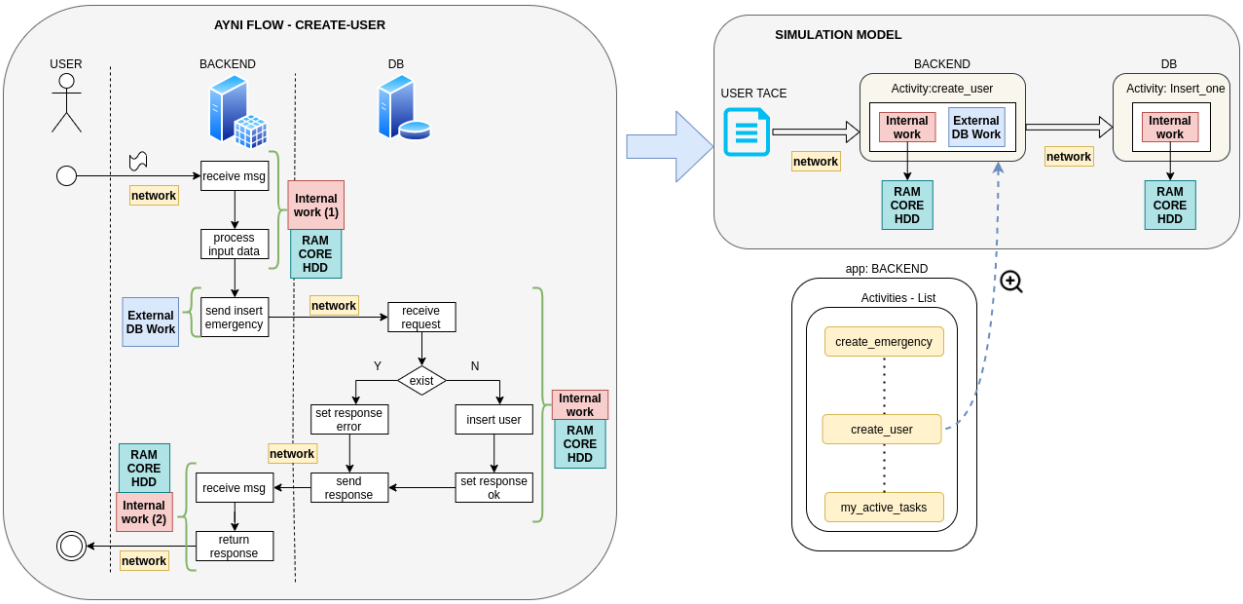
7.3 Flujo y modelo de simulación de “Crea tarea” de Ayni



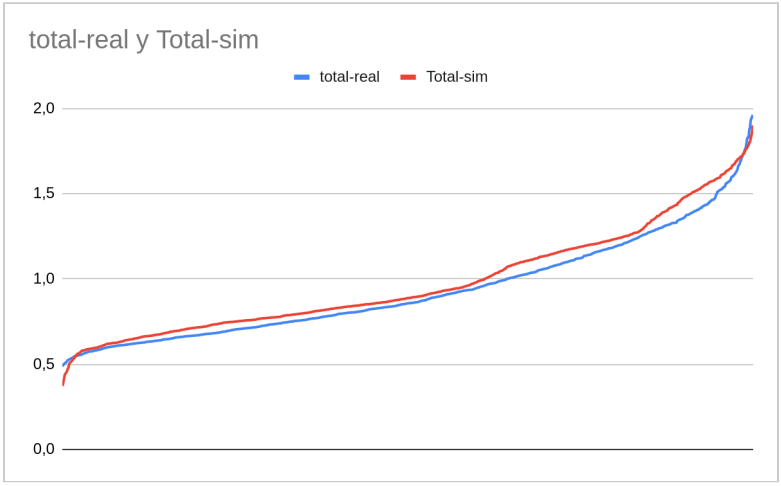
7.4: Curva datos simulación v/s real de la operación “Crea tarea” de Ayni

- Resultados de “Crear voluntario” de Ayni, en la Figura 7.5 se presenta el flujo y su representación con el modelo de simulación interno. Además, la validación que

alcanza un 5.7% de error, donde se realizaron 996 simulaciones y considerando en conjunto la operación de backend y base de datos, ver Figura 7.6, que presenta la curva de representación.



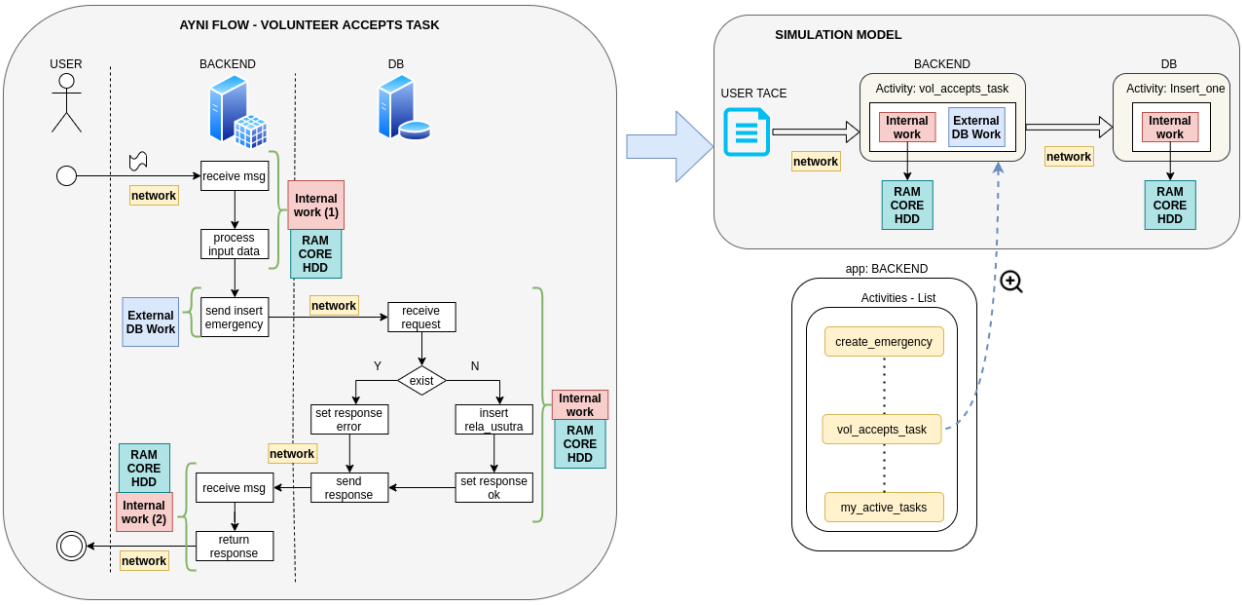
7.5 Flujo y modelo de simulación de “Crea voluntario” de Ayni



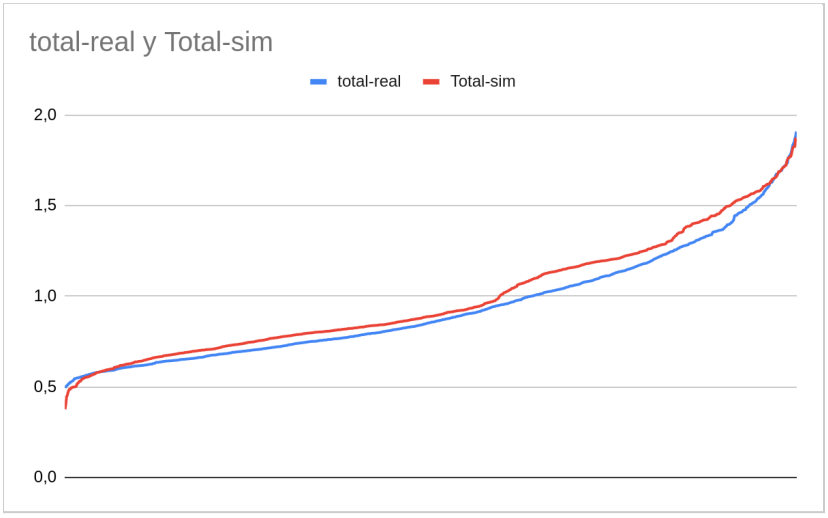
7.6: Curva datos simulación v/s real de la operación “Crea voluntario” de Ayni

- Resultados de “Voluntario acepta tarea” de Ayni, en la Figura 7.7 se presenta el flujo y su representación con el modelo de simulación interno. Además, la validación que alcanza un 6.3% de error, donde se realizaron 996 simulaciones y considerando en

conjunto la operación de backend y base de datos, ver Figura 7.8, que presenta la curva de representación.



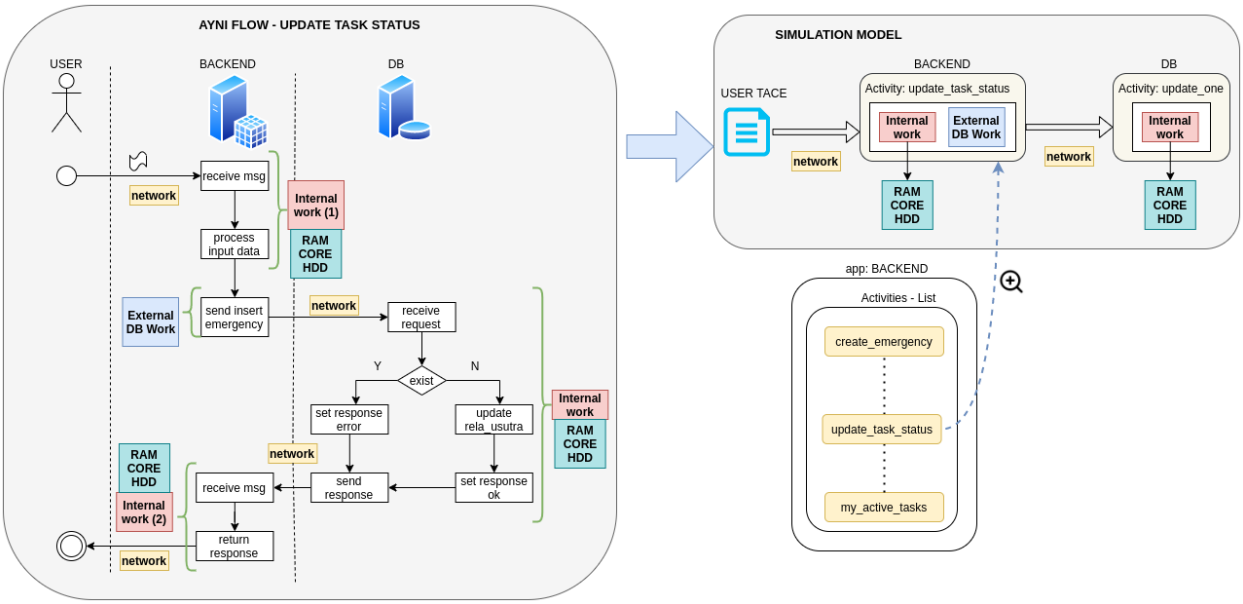
7.7 Flujo y modelo de simulación de “Voluntario acepta tarea” de Ayni



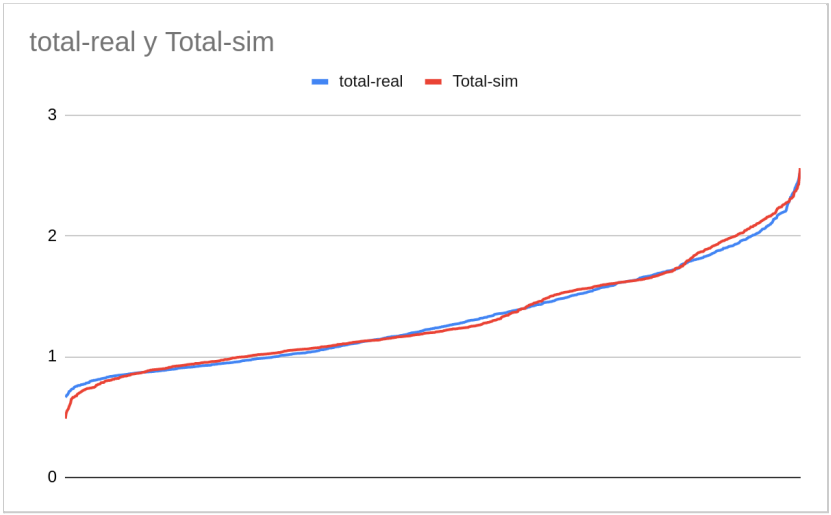
7.8: Curva datos simulación v/s real de la operación “Voluntario acepta tarea” de Ayni

- Resultados de “Actualiza estado tarea” de Ayni, en la Figura 7.9 se presenta el flujo y su representación con el modelo de simulación interno. Además, la validación que

alcanza un 6.3% de error, donde se realizaron 996 simulaciones y considerando en conjunto la operación de backend y base de datos, ver Figura 7.10, que presenta la curva de representación.



7.5 Flujo y modelo de simulación de “Actualiza estado tarea” de Ayni



7.6: Curva datos simulación v/s real de la operación “Actualiza estado tarea” de Ayni

8. Bibliografía

[1] Aon-plc. 2016 annual global climate and catastrophe report. http://thoughtleadership.aonbenfield.com/Documents/20170117-ab-if-annual-climate-catastrophe-report.pdf,2016. Disponible 2021-03-26.

[2] CEPCHILE. Estudio nacional de opinión pública n°33 tercera serie junio-julio 2010, Ministerio de Transportes y Telecomunicaciones, Subsecretaría de Telecomunicaciones, Chile. https://www.cepchile.cl/cep/site/artic/20160304/asocfile/20160304095248/DOC_encCEP_juni-jul2010.pdf, 2010. Disponible 2021-03-26.

[3] D. Guha-Sapir, P. Hoyois, P. Wallemacq, and R. Below. Annual disaster statistical review 2016: The numbers and trends, brussels, 2016. Disponible 2021-03-26.

[4] ITU.Technical report on telecommunications and disaster mitigation, international telecommunication union.https://www.itu.int/en/ITU-T/focusgroups/drnrr/Documents/Technical_report-2013-06.pdf, 2016. Disponible 2021-03-26.

[5] MIC. Maintaining communications capabilities during major natural disasters and other emergency situations, ministry of internal affairs and communications, japan.http://www.soumu.go.jp/main_content/000146938.pdf, 2011. Disponible 2021-03-26.

[6] USGS. 20 largest earthquakes in the world, usgs, science for changing world, usa.https://earthquake.usgs.gov/earthquakes/browse/largest-world.php, 2012. Disponible 2021-03-26.

[7] WorldRiskReport. Analysis and prospects 2017, 2017. . Disponible 2021-03-26.

[8] Marzolla, M. (2004, June). libcppsim: a Simula-like, portable process-oriented simulation library in C++. In Proc. of ESM (Vol. 4, pp. 222-227).