Plan de Recuperación, Transformación y Resiliencia

Financiado por la Unión Europea NextGenerationEU

uc3m | Universidad Carlos III de Madrid

SORUS

Validación y optimización conjunta de RIS y vRANs

SORUS-RAN A2.3-E2

SEGUNDA VERSIÓN DEL PROTOTIPO PARA VALIDAR LA ORQUESTACIÓN vRAN

Revisión	Autor	Fecha de entrega	Cambios
Versión 01	Jose Ayala Romero, Andrés García Saavedra	10/02/2024	Versión Inicial
Versión 02	Luis Roda Sánchez, Jorge San Martin Gomez	04/03/2024	Revisión, actualización de tablas, resumen ejecutivo y conclusiones

Exención de responsabilidad:

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810

CONTENIDOS

LISTA DE ABREVIATURAS Y ACRÓNIMOS

Tabla 1. Lista de abreviaturas y acrónimos

Abreviatura	Explicación/Definición
BS	Base Station
CAT	Cache Allocation Technology
CP	Control Plane
CQI	Channel Quality Indicator
DL	DownLink
DQN	Deep Q-Network
DT	Digital Twin
FCAPS	Fault, Configuration, Accounting, Performance And Security
FDD	Frequency-Division Duplexing
FEC	Forward Error Correction
GPP	General Purpose Processor
IA	Inteligencia Artificial
IoT	Internet of Things
LLC	Last Level Cache
MCS	Modulation and Coding Scheme
MLP	MultiLayer Perceptron
MSE	Mean Squared Error
NFV	Network Function Virtualization
NN	Neural Network
O-CU	Open – Centralized Unit
O-DU	Open – Distributed Unit
OFDM	Orthogonal Frequency-Division Multiplexing
O-RU	Open – Radio Unit

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810

Abreviatura	Explicación/Definición
PM	Performance Management
PoE	Power-over-Ethernet
RAN	Radio Access Network
RIC	RAN Intelligent Controller
RL	Reinforcement Learning
RN	Relational Network
RRM	Radio Resource Management
RT	Real-Time
RU	Radio Unit
SIRA	Single Instance Resource Allocation
SMO	Service Management and Orchestration
SMT	Simultaneous MultiThreading
SNR	Signal-to-Noise Ratio
SO	Sistema Operativo
UE	User Equipment
UL	UpLink
UP	User Plane
vBS	virtualized Base Station
VNF	Virtualized Network Function
vRAN	virtualized Radio Access Network

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810

LISTA DE FIGURAS

LISTA DE TABLAS

1 RESUMEN EJECUTIVO

Este documento se corresponde con el entregable SORUS-RAN-A2.3-E2. Este trabajo presenta soluciones a dos problemas detectados en vRANs generados por la compartición de recursos. En la primera parte se presenta AIRIC, un sistema para proporcionar reconfiguración dinámica basada en Reinforcement Learning teniendo en cuenta el problema de los *noisy neighbours*. Se ha conseguido reducir la utilización de recursos en un 17% dimensionando correctamente el conjunto de núcleos de computación a lo largo del tiempo. La segunda parte está destinada a describir el framework implementado para solucionar la asignación óptima de memoria caché LLC (L3). Para ello se presenta MemorAI, que, mediante un Digital Twin y un clasificador basado en redes neuronales, permite una asignación de LLC óptima con un ahorro energético entre 0,35kJ y 1 kJ según la estrategia de asignación de vías de caché.

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810

2 INTRODUCCIÓN

En este entregable, partimos de la caracterización experimental en SORUS-RAN-A2.1-E2 y abordamos dos problemas relevantes en la optimización de recursos y consumo energético en redes de acceso radio virtualizadas (vRANs). Esta investigación representa un paso significativo hacia la comprensión más profunda de los desafíos inherentes a la gestión de recursos en entornos virtualizados, donde la eficiencia operativa y el rendimiento son de suma importancia. En particular, nos centramos en dos áreas clave: la asignación de recursos computacionales en vRANs y la optimización de la asignación de caché L3 en servidores de computación compartidos.

En la primera parte del entregable, exploramos en detalle las complejidades asociadas con la optimización de asignación de recursos computacionales en vRANs. Esta tarea es fundamental para garantizar un rendimiento óptimo del sistema, ya que la asignación inadecuada de recursos puede llevar a cuellos de botella, latencia aumentada y una utilización ineficiente de la infraestructura. Nos sumergimos en la difícil tarea de cuantificar dinámicamente los requisitos de CPU para las instancias vBS, considerando una variedad de factores como la demanda de tráfico de red, la relación señal-ruido (SNR) de los enlaces inalámbricos y los esquemas de codificación de modulación (MCS) utilizados. Esta investigación revela la naturaleza compleja y dinámica de los requisitos computacionales en un entorno vRAN, destacando la importancia de enfoques adaptativos y basados en datos para la asignación de recursos[1] [2].

En la segunda parte del entregable, abordamos el problema de la asignación óptima de caché L3 en servidores de computación compartidos. La gestión eficiente de la caché L3 es crucial para optimizar el acceso a los datos y minimizar los cuellos de botella en la memoria, lo que a su vez tiene un impacto directo en el rendimiento del sistema. Nuestra investigación se centra en desarrollar estrategias inteligentes de asignación y gestión de la caché L3, diseñadas para adaptarse dinámicamente a las demandas cambiantes de las cargas de trabajo y optimizar la utilización de los recursos disponibles.

Ambos problemas abordados en este entregable son de suma relevancia en el contexto de las vRANs, donde la gestión eficiente de recursos es esencial para garantizar un desempeño óptimo y una utilización eficiente de la infraestructura. Nuestro enfoque se basa en una comprensión

[1] G. Garcia-Aviles et al., “Nuberu: Reliable RAN virtualization in shared platforms,” in Proceedings of the 27th MobiCom, 2021, pp. 749–761.

[2] J. A. Ayala-Romero et al., “vrAIn: A deep learning approach tailoring computing and radio resources in virtualized RANs,” in Proceedings of the 25th MobiCom, 2019, pp. 1–16.

profunda de las complejidades inherentes a estos sistemas, así como en la aplicación de técnicas avanzadas de optimización y gestión de recursos. Los experimentos realizados en el entorno de SORUS-RAN-A2.1-E2 demuestran empíricamente la efectividad de nuestras propuestas, validando así su relevancia y contribución al campo de las redes de acceso radio virtualizadas.

Sin embargo, abordar los requisitos computacionales de un sistema vRAN no está exento de desafíos. La naturaleza dinámica y heterogénea del tráfico de red, combinada con la variabilidad en las condiciones del canal inalámbrico, presenta desafíos significativos para la asignación de recursos y la gestión de la caché. Además, la contención de recursos computacionales, conocida como el problema de "noisy neighbours", puede afectar drásticamente el rendimiento del sistema, lo que subraya la importancia de una gestión eficiente de recursos y una planificación cuidadosa de la capacidad.

En este sentido, es crucial adoptar un enfoque holístico y adaptable para la gestión de recursos en entornos vRAN. Esto implica no solo considerar las demandas de rendimiento inmediatas, sino también anticipar y adaptarse a las fluctuaciones en las cargas de trabajo y las condiciones de la red. La aplicación efectiva de técnicas de aprendizaje automático y optimización puede desempeñar un papel crucial en este proceso, permitiendo una asignación más precisa y dinámica de recursos en tiempo real.

Este entregable representa un avance significativo en la optimización de recursos y consumo energético en redes de acceso radio virtualizadas. Nuestras propuestas ofrecen soluciones innovadoras respaldadas por una sólida fundamentación teórica y evidencia experimental. Al abordar los desafíos clave relacionados con la asignación de recursos y la gestión de la caché, esperamos contribuir al desarrollo de vRANs más eficientes, adaptables y sostenibles en el futuro.

3 RELACIÓN CON ENTREGABLES PREVIOS Y ESTRUCTURA DEL DOCUMENTO

En el entregable anterior (SORUS-RAN-A2.3-E1), se presentaron una serie de algoritmos de aprendizaje automático diseñados para abordar el desafío crucial de equilibrar eficazmente la energía y el rendimiento en redes virtualizadas. Estos algoritmos representan un paso significativo hacia la optimización de recursos en entornos dinámicos y cambiantes.

Sin embargo, es importante destacar que estos algoritmos se centraron exclusivamente en la configuración individual de vBSs, lo que limita su capacidad para abordar los efectos de la agregación de vBSs. La agregación de vBSs es un aspecto fundamental de la virtualización de redes,

ya que permite compartir recursos entre múltiples vBSs, lo que a su vez conlleva ahorros significativos de costos. Este concepto es particularmente relevante en el contexto de redes virtualizadas, donde la eficiencia operativa y el uso óptimo de los recursos son prioridades clave.

Por lo tanto, en este nuevo entregable, proponemos algoritmos de aprendizaje automático centrados en la gestión de recursos considerando la agregación de vBSs y sus implicaciones en términos de consumo de recursos y rendimiento del sistema. Uno de los desafíos principales que enfrentamos al agregar múltiples estaciones base en una misma plataforma de computación es la dificultad para predecir con precisión el consumo de recursos y el rendimiento, especialmente debido al fenómeno conocido como “noisy neighbours” en la interferencia de la memoria caché.

Este fenómeno se abordó detalladamente en el entregable anterior (SORUS-RAN-A2.1-E2), donde se exploraron en profundidad a través de medidas experimentales los efectos de la interferencia en la memoria caché y su impacto en el rendimiento del sistema. Así, a través de una exhaustiva investigación experimental, identificamos patrones y tendencias en el consumo de recursos y el rendimiento del sistema cuando se agregan múltiples estaciones base en una misma plataforma de computación. En base a estos resultados, en el presente entregable desarrollamos algoritmos de aprendizaje automático para optimizar la asignación de recursos y minimizar los efectos negativos de la interferencia entre procesos de vBS.

En resumen, este nuevo entregable representa un paso importante en nuestra investigación sobre la optimización de recursos en redes virtualizadas. Al centrarnos en la agregación de estaciones base y la gestión eficiente de recursos compartidos, buscamos abordar uno de los desafíos clave en el despliegue y operación de redes virtualizadas. Nuestro objetivo final es desarrollar soluciones innovadoras que mejoren tanto la eficiencia operativa como el rendimiento del sistema, contribuyendo así al avance de las tecnologías de redes virtualizadas.

Este entregable se divide en dos grandes secciones. En la Sección 4, abordamos la asignación óptima de recursos computacionales en vRANs. En particular en la Sección 4.1 se presenta el modelo del sistema, en la Sección 4.2 se presenta el framework de optimización y en la Sección 4.3 se incluyen los resultados experimentales de dicho framework.

Después, en la Sección 5 abordamos el problema de la asignación óptima de memoria caché LLC en vRANs. En particular, en la Sección 5.1 formulamos el problema, en la Sección 5.2, 5.3 y 5.4 presentamos el algoritmo de aprendizaje automático propuesto. En la Sección 5.5 presentamos los resultamos experimentales. Finalmente, en la Sección 6 presentamos la conclusiones de este entregable.

Financiado por el Programa “Next Generation” de la Unión Europea en el marco del acuerdo n.º 2022/0004810

4 ASIGNACIÓN ÓPTIMA DE RECURSOS COMPUTACIONALES EN VRANS

4.1. Modelo del sistema

Consideramos una plataforma de computación en nube O-RAN (O-Cloud) que proporciona recursos de computación para múltiples instancias vBS desplegadas en ella. Es importante destacar que cada instancia vBS comparte el mismo conjunto de recursos de computación, lo que introduce desafíos significativos en la gestión eficiente de estos recursos compartidos. Además, en este contexto, también consideramos la presencia de un agente encargado de dos tareas fundamentales: (i) observar el contexto asociado a cada vBS y (ii) idear qué núcleos de computación deben estar activos en el pool para atender la demanda de cada vBS, particularmente en lo que respecta al procesamiento de tráfico de enlace ascendente y descendente.

Siguiendo las especificaciones de O-RAN, nuestro agente se aloja en el Service Management and Orchestration (SMO) del sistema. Este agente opera tomando decisiones en intervalos de tiempo discretos, representados por $t \in N$, que denominamos *intervalos de decisión*. Estos intervalos de tiempo se encuentran en el rango de varios segundos a minutos, lo cual está en línea con las especificaciones de O-RAN para el controlador inteligente de RAN en tiempo no real (Non-RT RIC). Esta arquitectura temporal garantiza una gestión dinámica y adaptable de los recursos de computación, permitiendo que el sistema responda de manera eficiente a las fluctuaciones en la demanda de tráfico y las condiciones de la red.

Para llevar a cabo sus funciones, nuestro agente emplea un sistema de monitorización compatible con O-RAN, el cual recopila métricas de los distintos componentes de O-RAN, como O-RU, O-DU y O-CU. Además, también se recopilan mediciones de la plataforma O-Cloud, es decir, métricas de infraestructura que incluyen aspectos como el uso de computación y el consumo de energía. El RIC near-RT utiliza interfaces E2 para recibir periódicamente diferentes métricas de radio de los componentes desplegados en la plataforma O-Cloud. Posteriormente, el RIC near-RT transmite estos datos utilizando la interfaz O1 al RIC non-RT, lo que permite una comunicación fluida y eficiente entre los distintos componentes del sistema.

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810

Por otro lado, para recopilar métricas de la plataforma O-Cloud[3], el agente establece trabajos de gestión del rendimiento (PM) que recopilan diferentes métricas de infraestructura utilizando la interfaz O2. Estas métricas son esenciales para comprender el estado y la capacidad de la infraestructura subyacente, lo que permite al agente tomar decisiones informadas sobre la asignación de recursos. Finalmente, para hacer cumplir las diferentes políticas de computación que nuestro agente calcula, se utiliza la interfaz O2 para pasar esas políticas a la plataforma O-Cloud. Esta integración garantiza una coordinación efectiva entre la gestión de recursos a nivel de la red y a nivel de la infraestructura de computación en nube.

La Figura 1 ilustra cómo nuestro agente (AIRIC) se integra en la arquitectura O-RAN, desempeñando un papel crucial en la optimización y coordinación de recursos a lo largo de toda la infraestructura.

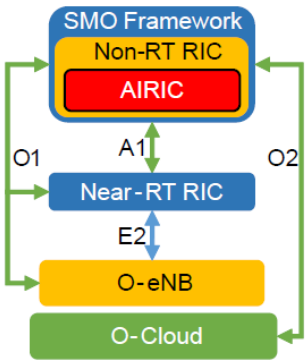

Figura 1: Integración de la solución propuesta en la arquitectura O-RAN

Dada la naturaleza intrínsecamente compleja del problema noisy neighbour, consideramos que el aprendizaje por refuerzo (RL) es una estrategia apropiada para diseñar nuestro agente. Esta elección se basa en la capacidad del RL para adaptarse dinámicamente a entornos cambiantes y aprender de la interacción con el entorno. En nuestro enfoque, el agente de RL observa el contexto del sistema y toma una acción al comienzo de cada intervalo de decisión. Luego, al final de cada intervalo de decisión, recibe una recompensa que refleja la calidad de su acción. Este proceso de observación, acción y retroalimentación continua permite que el agente aprenda y mejore su comportamiento a lo largo del tiempo.

[3] O-RAN Alliance, "O-RAN O2 General Aspects and Principles 2.0,"O-RAN Alliance, Technical Specification (TS), 10 2022.

Para facilitar este proceso de aprendizaje, el agente de RL almacena muestras de tuplas de 3, compuestas por el contexto observado, las acciones tomadas y las recompensas asociadas en cada intervalo de decisión. Estas experiencias acumuladas se utilizan posteriormente para actualizar el modelo del agente y mejorar su desempeño futuro. Es importante destacar que nuestro enfoque no solo se centra en maximizar el rendimiento del sistema en términos de asignación de recursos, sino que también tiene en cuenta la minimización de la interferencia entre las instancias vBS, lo que contribuye a mitigar el problema de noisy neighbour.

Aunque el problema del control de admisión, es decir, determinar cuántas instancias vBS pueden estar activas simultáneamente, no es el foco principal de este trabajo, reconocemos su importancia y permitimos que el número de instancias activas de vBS varíe dinámicamente en el sistema. Es relevante destacar que nuestra propuesta representa un avance significativo en el campo de la optimización de recursos en sistemas vRAN. Hasta donde sabemos, esta es la primera solución que aborda de manera óptima la asignación de recursos computacionales en un entorno vRAN, teniendo en cuenta tanto la sobrecarga asociada con el problema de noisy neighbours como la variabilidad en el número de instancias vBS activas en el sistema. Esta capacidad para adaptarse dinámicamente a las condiciones cambiantes del entorno es fundamental para garantizar un rendimiento óptimo y una eficiencia energética en entornos de red virtualizados.

4.2. Framework de optimización

Un número variable de instancias vBS implica que la dimensionalidad de la información de contexto también varía con el tiempo. Esto es particularmente difícil de soportar con soluciones de RL estándar. Para hacer frente a esto, aumentamos un enfoque clásico *Deep Q-Network* (DQN)[4] con un mecanismo *Relational Network* (RN)[5], como muestra la Figura 2.

[4] V. Mnih et al., “Playing Atari with deep reinforcement learning,” arXiv preprint arXiv:1312.5602, 2013.

[5] D. Raposo et al., “Discovering objects and their relations from entangled scene representations,” arXiv preprint arXiv:1702.05068, 2017.

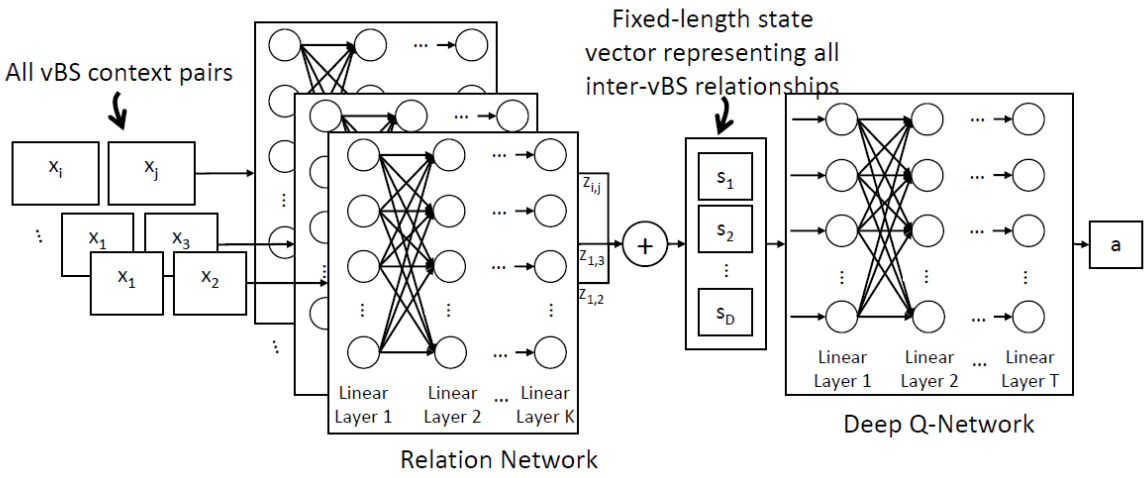

Figura 2: Arquitectura de machine learning propuesta

La idea básica de un agente RL es aprender una política óptima π interactuando con un entorno $\mathcal{E}$ en intervalos de tiempo discretos $t \in \{1, 2, \dots, T\}$. En cada intervalo, un agente observa un estado (o contexto) $s^{(t)}$, selecciona una acción $a^{(t)}$ y recibe una recompensa $r^{(t)}$ al final del paso temporal. Una política π es una distribución de acciones sobre los diferentes estados, que captura como de bueno es el par estado-acción $(s^{(t)}, a^{(t)})$. Una vez medida la recompensa $r^{(t)}$, el sistema pasa al estado $s^{(t+1)}$. Después de T intervalos, $\mathcal{E}$ alcanza su estado terminal y el agente refina su política π utilizando observaciones pasadas $\{\{s^{(1)}, a^{(1)}, r^{(1)}\}, \dots \{s^{(T-1)}, a^{(T-1)}, r^{(T-1)}\}\}$. El objetivo es maximizar la recompensa total descontada:

$$R^{(t)} := r^{(t)} + \sum_{t'=t+1}^{T} \gamma^{t'} r^{(t')}.$$

La mayoría de las soluciones RL aproximan funciones de valor que estiman la importancia de las acciones dado un estado s. Una de esas funciones de valor es $Q^*(s,a) := max_\pi \mathbb{E}\left[R^{(t)} \middle| s^{(t)} = s, a^{(t)} = a\right]$, que representa el máximo rendimiento esperado dado un par acción-estado bajo la política π. La función de valor Q^* óptima sigue la Ecuación de Optimalidad de Bellman, que proporciona $Q^*\left(s^{(t)}, a^{(t)}\right)$ en términos de $Q^*\left(s^{(t+1)}, a^{(t+1)}\right)$:

$$Q^*(\vec{s}, a) = \mathbb{E}\left[r^{(t)} + \gamma \max_{a^{(t+1)}} Q^*(\vec{s}^{(t+1)}, a^{(t+1)}) | \vec{s}^{(t)} = \vec{s}, a^{(t)} = a\right]$$

Usando la Ecuación de Optimalidad de Bellman, podemos encontrar $Q^*(s,a)$ iterativamente[6]. En este trabajo, hemos utilizado redes neuronales para aproximar el óptimo $Q^*(s,a)$, que se llama DQN. En particular, dada la gran escala de tiempo de la RIC No-RT, la acción tomada en un intervalo $a^{(t)}$ tiene poco impacto en el siguiente estado $s^{(t+1)}$ y por lo tanto es suficiente para maximizar la recompensa instantánea. Por lo tanto, para acelerar la convergencia, simplificamos nuestra configuración de RL en un problema de bandido contextual estableciendo $\gamma = 0$ y $T = 1$. A continuación, describimos nuestro diseño para el contexto (estados), las acciones y la función de recompensa del agente de aprendizaje.

4.2.1. Definición de contextos

En línea con la literatura relacionada[7] [8] [9], utilizamos las siguientes métricas para describir el estado:

- **Calidad del canal**: utilizamos la SNR UL media observada por cada vBS en el último intervalo, lo que permite a nuestro agente inferir su capacidad inalámbrica UL, y el indicador de calidad del canal (CQI) DL medio para hacer lo propio con el DL.
- **Demanda de red**: la demanda de red de un vBS es la cantidad de datos almacenados en buffer por el UE tanto para UL como para DL durante el último intervalo de decisión.

Representamos la calidad de los canales DL y UL para un caso de vBS i observado en el intervalo t como $\sigma_{DL,i}^{(t)}$ y $\sigma_{UL,i}^{(t)}$. Además, dejamos que $d_{DL,i}^{(t)}$ y $d_{UL,i}^{(t)}$. denoten su demanda de red DL y UL, respectivamente. También suponemos una correspondencia conocida entre la calidad del canal y el MCS: $g_{DL}(\sigma_{DL,i}^{(t)})$ para DL, $g_{UL}(\sigma_{UL,i}^{(t)})$ para UL, que es una suposición suave. Dado que la calidad del canal limita el MCS más alto, podemos estimar el número medio de bloques de recursos de radio (RB) que cada vBS puede utilizar en ambas direcciones dado un MCS medio y la demanda de la red. Esto puede estimarse utilizando las especificaciones 3GPP[10]. De este modo, podemos establecer la demanda de recursos radioeléctricos (RB) en lugar de basarnos únicamente en la utilización pasada de los bloques radioeléctricos, que puede diferir. En consecuencia, denotamos el número de RB utilizados para DL y UL para vBS i como p_i^{DL} y p_i^{UL}, respectivamente. Utilizando el número de RB y la demanda de red, definimos el contexto de vBS i como

[6] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[7] J. A. Ayala-Romero et al., "vrAIn: A deep learning approach tailoring computing and radio resources in virtualized RANs," in Proceedings of the 25th MobiCom, 2019, pp. 1–16.

[8] J. A. Ayala-Romero et al., "Bayesian online learning for energy-aware resource orchestration in virtualized rans," in IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, 2021, pp. 1–10.

[9] EdgeBOL: Automating Energy-Savings for Mobile Edge AI. New York, NY, USA: Association for Computing Machinery, 2021, p. 397–410. [Online]. Available: https://doi.org/10.1145/3485983.3494849

[10] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures," Link, 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 36.213, 06 2022, version 17.2.0.

$$x_i^{(t)} := \left(p_{DL,i}^{(t)}, d_{DL,i}^{(t)}, p_{UL,i}^{(t)}, d_{UL,i}^{(t)}\right).$$

El diseño de $x_i^{(t)}$ está motivado por la conveniencia de características expresivas y dimensionalidad mínima y sigue el estado del arte[7][8][9]. El reto ahora es codificar la información de contexto $x_i^{(t)}$ para todas las instancias i de vBS en un vector de estado s con una dimensionalidad fija D, que es requerida por el modelo DQN, en escenarios con un número variable de instancias de vBS a lo largo del tiempo. Como se muestra en la Figura 2, abordamos esta cuestión con una RN[5].

4.2.2. Redes relacionales

Como el número de vBSs para los que AIRIC tiene que asignar recursos de CPU en un intervalo de tiempo concreto puede ser diferente que en intervalos anteriores, la longitud del contexto cambia en función del número de instancias de vBS. En lugar de construir otros agentes para cada uno de los distintos números de casos de vBS o de rellenar los distintos contextos posibles para que coincidan con una longitud de contexto fija, optamos por resolver el problema utilizando una red de relaciones. Una RN puede codificar la relación entre el contexto asociado a todas las instancias vBS en un vector de estado de longitud fija s.

Para ello, la RN opera a lo largo de todos los pares de objetos posibles (contexto de las instancias vBS) para capturar dichas relaciones ocultas con un modelo de perceptrón multicapa (MLP). Suponiendo un número máximo de instancias vBS soportadas en el sistema igual a M, entonces tenemos los siguientes pares posibles de vectores de contexto:

$$\mathcal{X} := \{(x_1, x_2), (x_1, x_3), \dots (x_{M-1}, x_M)\}$$

Dado que la cantidad máxima de instancias vBS en un momento dado está acotada, $|\mathcal{X}|$ también está acotada y es fija en el tiempo. El RN ingiere secuencialmente cada par $(x_i, x_j) \in \mathcal{X}$ de posibles combinaciones de contexto no permutadas, y genera un vector de salida $z_{i,j}$ con cardinalidad D. Una vez que todas las permutaciones $\binom{N}{2}$ del vector $z_{i,j}$ son calculados por la RN, lo que se hace secuencialmente, creamos un vector de estado codificado s sumando todos los vectores de salida, es decir, $s = \sum_{i,j} z_{i,j}$. De esta manera, forzamos la invariancia de permutación de orden, que es un requisito crítico de nuestro problema, es decir, como la RN aprende sobre diferentes relaciones latentes a través de instancias vBS (objetos), estas relaciones aprendidas permanecen invariantes independientemente del orden de las relaciones de pares de entrada. Es importante destacar que nuestra RN no sólo ayuda a soportar un número variable de instancias vBS a lo largo del tiempo, sino que también proporciona al modelo DQN información de estado que representa mejor las relaciones entre ellas, lo cual es muy útil para capturar el impacto del problema de *noisy neighbours* en una representación de estado de dimensión fija. Para ello, entrenamos la red RN conjuntamente con el modelo DQN como explicamos más adelante.

4.2.3. Definición de acciones

Dado el estado $s^{(t)}$, nuestro agente activará el conjunto apropiado de núcleos de CPU, descrito con un vector de activación v en el que cada elemento corresponde al índice del núcleo de CPU que se activará. Entonces, todas las instancias vBS compartirán equitativamente el conjunto de núcleos de CPU en v. Al evitar la asignación de cargas de trabajo vBS a núcleos específicos, pretendemos maximizar la multiplexación de recursos y, en consecuencia, reducir el uso global de recursos computacionales. Para garantizar una convergencia rápida, debemos mantener una baja dimensionalidad del espacio de acción. Para ello, dividimos nuestra acción en dos pasos. En el paso 1, nuestro agente RL decide el número total de núcleos de CPU que se activarán para garantizar el servicio. Así, el conjunto de acciones A es $A = \{1, 2, \dots, 2N\}$, donde N es el número total de núcleos físicos disponibles. A continuación, en el paso 2, implementamos una regla $\rho(a)$ determinista para minimizar el coste de infraestructura. Es decir, $\rho : A \mapsto \mathcal{V}_a$, $a \mapsto v$, donde $\mathcal{V}_a$ es un conjunto que contiene todos los posibles vectores de activación tal que $a = |v|$. Dado que ρ es una regla predeterminada para minimizar el coste, el agente puede aprender su política π para garantizar el servicio dado ρ como parte del entorno $\mathcal{E}$.

Véase, por ejemplo, el GPP de la Figura 3 con $N = 2$. Si $a = 1$ entonces $\mathcal{V}_{a=1} = \{(0), (1), (2), (3)\}$ todos los vectores de activación en $\mathcal{V}_{a=1}$ son equivalentes y cualquier $v \in \mathcal{V}_{a=1}$ podría elegirse trivialmente. Sin embargo, este no es necesariamente el caso para otras acciones a porque, como hemos explicado antes, los procesadores modernos aprovechan las CPU multiproceso, siendo dos núcleos virtuales por cada CPU física el caso más común. Por ejemplo, para $a = 2$ (y la misma GPP con $N = 2$), el conjunto de posibles vectores de activación es $\mathcal{V}_{a=2} = \{(0,2), (1,3), (0,1), (0,3), (1,2), (1,3)\}$. Aunque muchos de los vectores en $\mathcal{V}_{a=2}$ son equivalentes, otros no lo son. El subconjunto $\hat{\mathcal{V}}_{1,a=2} = \{(0,2), (1,3)\} \subset \mathcal{V}_{a=2}$ contiene vectores de activación equivalentes; y también lo son los vectores de activación en $\hat{\mathcal{V}}_{2,a=2} = \{(2,3), (0,1), (0,3), (1,2)\} \subset \mathcal{V}_{a=2}$. Pero cualquier $v_1 \in \hat{\mathcal{V}}_{1,a=2}$ y cualquier $v_2 \in \hat{\mathcal{V}}_{2,a=2}$ *no* son equivalentes. Por un lado, cualquier $v_1 \in \hat{\mathcal{V}}_{1,a=2}$ incurre en más contención de caché que cualquier $v_2 \in \hat{\mathcal{V}}_{2,a=2}$ porque todos los núcleos en v_1 comparten la misma CPU física (véase la Figura 3).

Por otro lado, cualquier $v_2 \in \hat{\mathcal{V}}_{2,a=2}$ es más costoso que cualquier $v_1 \in \hat{\mathcal{V}}_{1,a=2}$ porque v_1 permite apagar más CPU físicas, por ejemplo, si $v_1 = (0,2)$ se puede apagar la CPU 1 (véase la Figura 3). La Figura 4 ilustra un ejemplo del funcionamiento de nuestro algoritmo durante tres periodos de decisión. Es importante destacar que, dado un conjunto de núcleos de CPU activados, todas las instancias de vBS utilizarán equitativamente esos núcleos utilizando un scheduler estándar.

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810

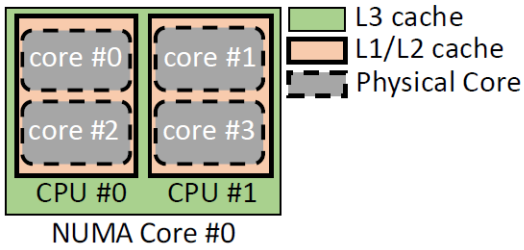

Figura 3: Ejemplo simplificado de un procesador de propósito general

En el supuesto de que, dado cualquier mapeo estático ρ, la política π proporcionará una cardinalidad adecuada para el vector de activación para garantizar el servicio de red ($a = |v|$), sólo tenemos que diseñar ρ con el objetivo de minimizar la cantidad de infraestructura (CPUs físicas) que tiene que ser activada dado a. En consecuencia, proponemos la siguiente regla sencilla. Sea $k(v) \in \{1,2,\dots,N\}$ el número de CPUs físicas que contienen al menos un núcleo virtual activado en v. Entonces, dado un conjunto $\mathcal{V}_a$ con todos los vectores de activación posibles para la acción a, definimos el superconjunto $\mathcal{W}_a := \langle \hat{\mathcal{V}}_{1,a}, \dots, \hat{\mathcal{V}}_{N,a} \rangle$, donde $\hat{\mathcal{V}}_{i,a} = \{v | k(v) = 1, v \in \mathcal{V}_a\}$. En el ejemplo anterior, con $a = 2$ y $N = 2$, $\mathcal{W}_a := \langle \hat{\mathcal{V}}_{1,a=2}, \hat{\mathcal{V}}_{2,a=2} \rangle$. Obsérvese que $\hat{\mathcal{V}}_{i,a} = \emptyset$ para algunos valores de i. Por ejemplo, en nuestro ejemplo simplificado de la Figura 4 con $N = 2$, $\hat{\mathcal{V}}_{1,a=3} = \emptyset$ para $a = 3$. Por lo tanto, dejamos que $\rho(a) = v \in \hat{\mathcal{V}}_{m,a}$ tal que $m := argmin_i \{i | \hat{\mathcal{V}}_{i,a} \neq \emptyset\}$.

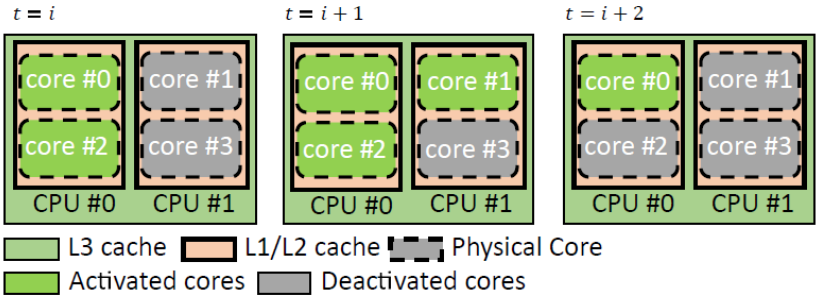

Figura 4: Ejemplo de secuencia de acciones del método propuesto

4.2.4. Diseño de la función de recompensa

Nuestro objetivo es satisfacer la demanda de tráfico de todos los sistemas vBS desplegados en el sistema a lo largo del tiempo con una infraestructura física mínima (para ahorrar costes apagando CPUs). Suponiendo un pool con N CPUs físicas y $2N$ núcleos virtuales, donde los núcleos j y $j + N$ pertenecen a la misma CPU física $\forall j < N$, dejamos que $z(j) \in \{0, \dots, 2N - 1\}$ denote el núcleo virtual *hermano* i dado el núcleo virtual de entrada j. Un núcleo hermano es aquel que utiliza la misma CPU física. Por ejemplo, en el GPP del ejemplo de la Figura 3, con $N = 2$ CPUs físicas y 4 núcleos, $z(0) = 2$ y $z(2) = 0$.

Siguiendo la literatura relacionada[11] [12], codificamos el coste asociado a un vector de activación v utilizando un modelo lineal. Denotemos primero $c_j^{(t)} \in [0,1]$, como el uso relativo del núcleo informático j durante el intervalo t. Si $j \notin v^{(t)}$, entonces $c_j^{(t)} = 0$; de lo contrario, $c_j^{(t)}$ se mide empíricamente. A continuación, dejamos que $E_j^{(t)}$ modele el coste (relacionado con la energía) asociado con la computación del núcleo $j \in \{0, 1, \ldots, 2N-1\}$ de la siguiente manera:

$$E_j^{(t)} := \begin{cases} \alpha_1 + \beta \cdot c_j^{(t)} & \text{if } c_j^{(t)} > 0 \\ \alpha_2 & \text{if } c_j^{(t)} = 0 \text{ and } c_{z(j)}^{(t)} > 0 \\ \alpha_3 & \text{if } c_j^{(t)} = 0 \text{ and } c_{z(j)}^{(t)} = 0 \end{cases}$$

donde $\alpha_1 > \alpha_2 > \alpha_3$. Intuitivamente, α_i modela el coste de sesgo de un núcleo, que es diferente dependiendo del estado de activación del núcleo j y su hermano. Elegimos α_i y β de modo que $0 \leq E_j \leq 1$.

Dejemos ahora que $\tau_{DL,i}^{(t)}$ y $\tau_{UL,i}^{(t)}$ denoten el rendimiento DL/UL experimentado por vBS i durante el intervalo t, y luego formalicemos nuestra función de recompensa como:

$$r^{(t)} := \begin{cases} -1, & \text{if } \tau_{DL,i}^{(t)} < d_{DL,i}^{(t)} \text{ for any } i \\ -1, & \text{if } \tau_{UL,i}^{(t)} < d_{UL,i}^{(t)} \text{ for any } i \\ \frac{1}{2N} \sum_{j=0}^{2N-1} -E_j, & \text{otherwise} \end{cases}$$

4.2.5. Entrenamiento del algoritmo

Como se ha explicado anteriormente, el objetivo es entrenar una política para aproximar una función acción-valor óptima Q^*. Nuestra política π se implementa mediante la estructura de RN+DQN introducida anteriormente y, por tanto, optimizaremos los pesos $\Theta := (\theta_1, \theta_2)$ de las redes neuronales combinadas para estimar la función Q-valor $Q(s, a; \theta) \approx Q^*(s, a)$. Para ello, utilizamos una función de coste Smooth L1[13]:

$$L^{(t)}(\Theta_i) := \begin{cases} \frac{1}{2}\frac{x^2}{1} & \text{if } |x| < 1 \\ |x| - \frac{1}{2} \cdot 1 & \text{otherwise} \end{cases}$$

[11] A. Jaiantilal et al., "Modeling CPU energy consumption for energy efficient scheduling," in Proceedings of the 1st Workshop on Green Computing, 2010, pp. 10–15.

[12] A. Shahid, M. Fahad, R. R. Manumachu, and A. Lastovetsky, "Energy of computing on multicore CPUs: Predictive models and energy conservation law," arXiv preprint arXiv:1907.02805, 2019.

[13] R. Girshick, "Fast R-CNN," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

donde $x = \mathbb{E}_{(s,a,r,s')\sim\rho}\left[\left(y_i - Q(s,a;\Theta_i)\right)\right]$ y $y_i = r + \gamma\, max_{a^{(t+1)}}\, Q\left(s', a^{(t+1)}; \Theta_{i-1}\right)$. ρ es un buffer de repetición desde donde muestreamos (s, a, r, s'), y_i es el objetivo de diferencia temporal, y $y_i - Q$ es el error de diferencia temporal. Utilizamos una red objetivo para estabilizar el proceso de entrenamiento, es decir, el agente de aprendizaje utiliza una red objetivo diferente con pesos fijos que se utilizan para calcular la función de pérdida utilizada a su vez para entrenar la red Q primaria. Es fundamental subrayar que los parámetros de la red objetivo se sincronizan periódicamente con los de la red Q primaria en lugar de entrenarse. La red Q primaria se entrena utilizando los valores Q de la red objetivo para aumentar la estabilidad del entrenamiento. Por último, utilizamos un enfoque estándar ϵ-greedy para la exploración.

4.3. Evaluación Experimental

Para la evaluación experimental presentada en esta sección se utiliza la plataforma experimental detallada en la sección "Plataforma Experimental" del entregable SORUS-RAN-A2.1-E2. La solución propuesta, llamada AIRIC, ha sido implementada utilizando PyTorch[14]. Por un lado, la RN tiene una capa oculta y el mismo número de neuronas que la capa de salida, 128. Por otro lado, la DQN tiene una capa oculta con 256 neuronas. Los parámetros iniciales de las redes neuronales se inicializan a partir de una distribución uniforme. También utilizamos la función de activación ReLU, y una capa de normalización[15] entre las capas ocultas. Para el mecanismo ϵ-greedy, utilizamos un factor de decaimiento igual a 60% del tamaño del conjunto de entrenamiento. Para entrenar el algoritmo, se han generado 60.000 muestras de datos de contexto-acción-recompensa, divididas uniformemente para escenarios con 2, 3 y 4 instancias de vBS funcionando simultáneamente. Aleatorizamos y dividimos el conjunto de datos en un conjunto de entrenamiento y otro de validación de 40.000 y 20.000 muestras, respectivamente. También utilizamos un búfer de repetición con 20.000 muestras y batch size de 128 muestras. Por último, utilizamos Adam[16] como optimizador. Estas opciones de implementación pretenden estabilizar el entrenamiento basado en trabajos previos[15] [17].

4.3.1. Evaluación de la convergencia

Comenzamos evaluando la convergencia. La Figura 5 muestra la recompensa normalizada del algoritmo propuesto a lo largo de las iteraciones de entrenamiento. La carga UL/DL y la SNR generadas en ambos gráficos se eligen uniformemente al azar. Sin embargo, mientras que el

[14] HTTP://WWW.PYTORCH.ORG

[15] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," arXiv preprint arXiv:1607.06450, 2016.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[17] G. Thimm and E. Fiesler, "High-order and multilayer perceptron initialization," IEEE Transactions on Neural Networks, vol. 8, no. 2, pp. 349–359, 1997.

número de instancias vBS también es aleatorio (entre 2 y 4) en Figura 5 (izquierda), llegan secuencialmente en Figura 5 (derecha). En el primer caso, la recompensa converge a 0,95 en menos de 5.000 iteraciones. En el segundo caso, se esperan baches cuando llegan nuevas vBS, pero éstos son pequeños, dentro de 5%. Por lo tanto, llegamos a la conclusión de que la RN en la solución propuesta aprende correctamente la relación entre vBSs y cómo utilizar su experiencia para alcanzar rápidamente un rendimiento cercano al óptimo.

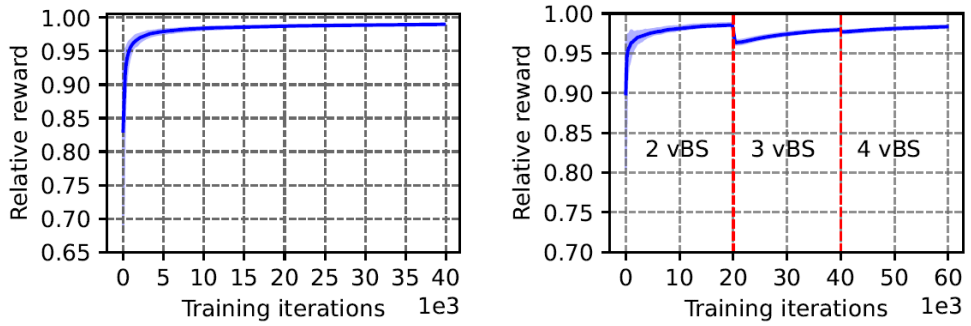

Figura 5: Evaluación de convergencia para contextos aleatorios (izquierda) y un número secuencial de vBSs (derecha)

4.3.2. Tiempo de inferencia

Con el fin de evaluar si AIRIC es adecuado para su ejecución en un non-RT RIC, hemos medido el tiempo de inferencia de nuestro enfoque para el diferente número de casos vBS. Los resultados, representados en la Figura 6, muestran tiempos de inferencia inferiores a 1 ms para todos los casos, que está muy por debajo del ciclo de bucle de control de un controlador RIC y valida AIRIC para operar en él adecuadamente.

4.3.3. Evaluación de rendimiento

Para entender mejor la eficacia de nuestra solución, comparamos ahora con un enfoque de asignación de recursos de instancia única (Single Instance Resource Allocation o SIRA). SIRA está diseñado a propósito para orquestar recursos óptimos a través de instancias vBS bajo el supuesto de un completo aislamiento informático entre instancias. En consecuencia, SIRA representa límites superiores alcanzables por los trabajos existentes sobre orquestación de CPU vRAN[2] [18].

[18] S. Tripathi, C. Puligheddu, S. Pramanik, A. Garcia-Saavedra, and C. F. Chiasserini, "Fair and scalable orchestration of network and compute resources for virtual edge services," IEEE Transactions on Mobile Computing, 2023.

Para la evaluación, en cada intervalo elegimos uniformemente al azar el número de instancias vBS, su carga DL/UL y su SNR DL/UL, y utilizamos ambos enfoques (AIRIC y SIRA) para optimizar la asignación de recursos computacionales de forma dinámica. En el caso de SIRA, utilizamos diferentes modelos (previamente entrenados) en función del número de instancias. Para comparar, también representamos el rendimiento de un oráculo, etiquetado como "Optimal", que encuentra la acción óptima offline mediante búsqueda exhaustiva.

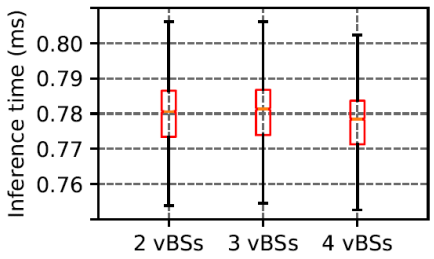

Figura 6: Tiempo de inferencia

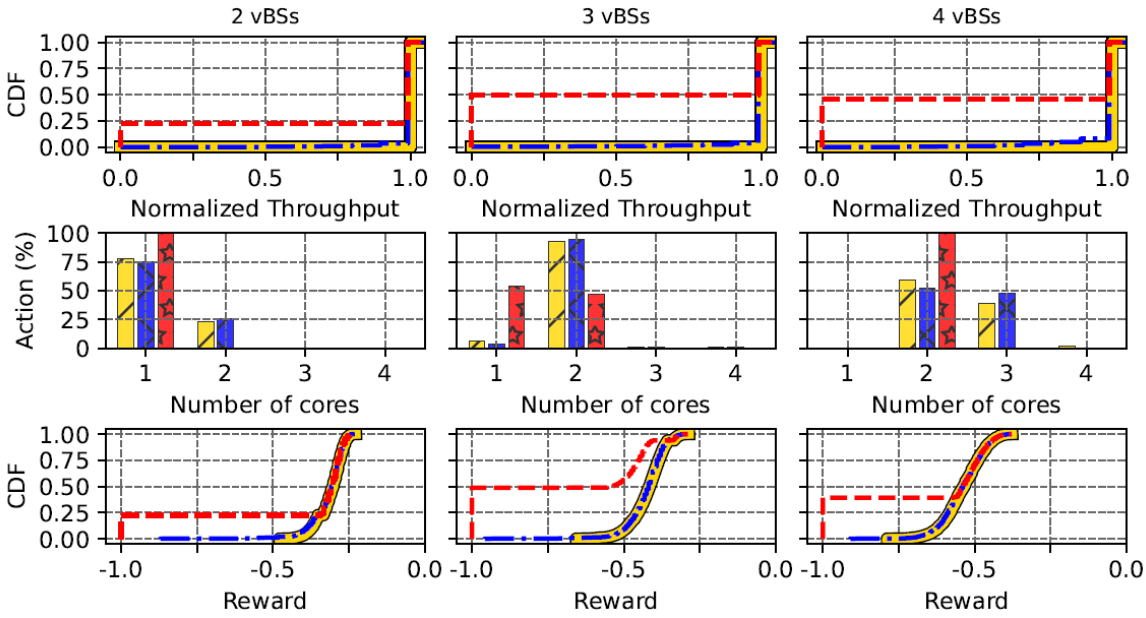

Figura 7: Evaluación de rendimiento

La Figura 7 representa la distribución del rendimiento agregado normalizado del sistema (arriba), las asignaciones de CPU (medio), y la distribución de la recompensa obtenida (abajo), para todos los enfoques condicionados a la presencia de 2 (izquierda), 3 (centro) y 4 (derecha) instancias vBS. A la inversa, la Figura 8 representa el ahorro absoluto (eje y izquierdo) y relativo (eje y derecho) de consumo energético conseguido por los tres enfoques. Estos ahorros se comparan con la energía consumida cuando el programador predeterminado de Linux gestiona todos los núcleos de CPU

disponibles en el sistema, como se indica en el eje x. Los gráficos de caja representan los percentiles 25 y 75 (bordes de la caja), la mediana (línea dentro de la caja) y los percentiles 5-95 (barras de error). Hacemos tres observaciones: La primera es que AIRIC proporciona ahorros sustanciales, comparables a la solución óptima. Tal vez sorprendentemente, SIRA muestra un ahorro ligeramente superior en algunos casos, lo que nos lleva a nuestra segunda observación: el ahorro proporcionado por SIRA tiene un precio enorme en el rendimiento, como se muestra en la Figura 7. Esto se agrava en los escenarios más densos: con 4 vBS, SIRA apenas ahorra un 7 % más de recursos computacionales que AIRIC de media, pero a cambio incurre en una pérdida de rendimiento del 50%. Esto se debe al hecho de que SIRA ignora la sobrecarga computacional adicional causada por el problema de *noisy neighbours* y a menudo infraasigna recursos, lo que provoca violaciones de la PHY y pérdidas de rendimiento. La observación final es que AIRIC proporciona un rendimiento que se aproxima notablemente al de la solución óptima. Por otra parte, la Figura 7 (abajo) confirma que la distribución de recompensa obtenida por la solución propuesta es muy cercana a la proporcionada por el oráculo óptimo. Estas observaciones validan nuestro diseño.

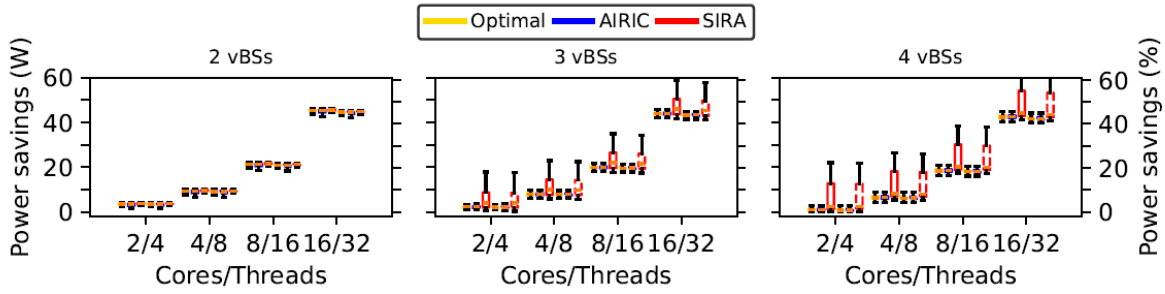

Figura 8: Medidas de ahorro energético

4.3.4. Contextos basados en trazas realistas

Por último, evaluamos nuestra solución con dinámicas de contexto realistas. Para ello, hemos generado perfiles de contexto para 4 instancias vBS diferentes, implementando network slices con diferentes perfiles de contexto, durante 5 días seguidos. La Figura 9 muestra la evolución temporal de la carga de red tanto DL como UL para estas 4 trazas. El slice 1 emula el comportamiento de un vBS eMBB en el centro de la ciudad, con patrones de carga diurnos comunes. El slice 2 emula un EBT que da servicio a un edificio de oficinas, con un pico de carga durante las horas de oficina (9h - 17h). Ambas dinámicas de contexto se han adaptado a partir de las de trabajos previos[8]. Los slices 3 y 4, por su parte, emulan vBS que dan servicio a IoT con cargas constantes cuando están operativos.

La Figura 10 muestra la distribución del rendimiento (izquierda) y el ahorro de recursos computacionales (derecha) de AIRIC, SIRA y el oráculo óptimo. Como en el caso anterior, SIRA proporciona un ahorro medio de CPU en torno al 55%, pero como consecuencia sufre una pérdida

de rendimiento de casi el 25% a lo largo de los 5 días. Por el contrario, el rendimiento de nuestra solución es muy similar al del oráculo, sin pérdida de rendimiento y con un ahorro global de recursos computacionales de alrededor del 17%, lo que valida AIRIC para escenarios realistas.

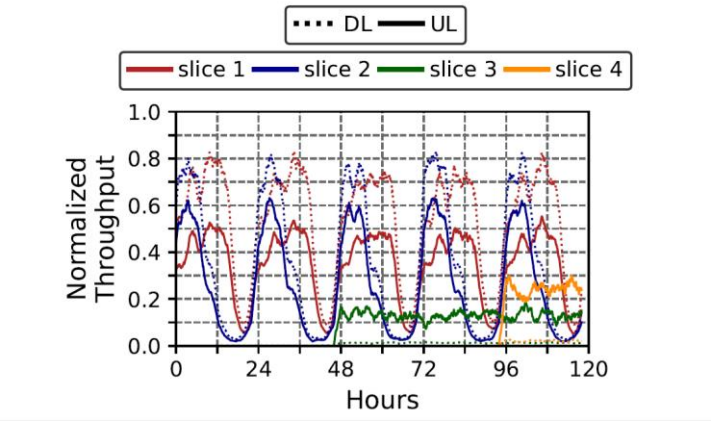

Figura 9: Cargas de tráfico realistas

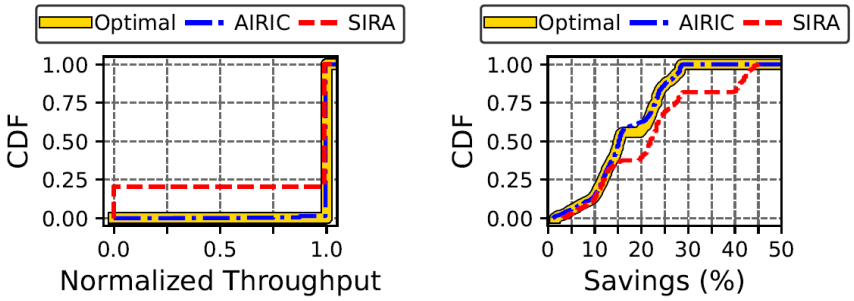

Figura 10: Perfiles contextuales dinámicos basados en trazas realistas

5 Asignación Óptima de Memoria Caché LLC en vRANs

En esta segunda parte del entregable abordamos el problema de la asignación de memoria caché para reducir el consumo total de energía en vRANs. Este es un problema muy desafiante ya que el impacto de la memoria caché LLC en el consumo energético depende de las demandas de computación del vBS, que están influenciadas por varios factores, incluyendo la demanda de tráfico tanto en DL como en UL, el SNR de cada enlace inalámbrico y el MCS específico utilizado para la comunicación. Todos estos elementos interactúan de forma compleja como se detalla en trabajos previos[17].

5.1. Formulación del problema

Consideramos una plataforma vRAN que comprende M_{cores} núcleos de computación y N_{LLC} vías de caché LLC. Consideramos que N_{vBS} instancias vBS están desplegadas en la plataforma. Cada vBS i en la plataforma vRAN tiene una demanda de tráfico UL y DL denotada por d_i^{UL} y d_i^{DL}, respectivamente; un SNR s_i; y un MCS en UL y DL denotado por m_i^{UL} y m_i^{DL}, respectivamente, que el programador de radio selecciona en función de s_i.

La vBS i tiene un conjunto de núcleos aislados P_i tal que $|P_i| > 0$ y un número de vías de caché LLC dedicadas n_i^{LLC}, donde $n_i^{LLC} \geq 1 \; \forall i$. Tenemos que satisfacer que $\sum_i |P_i| \leq M_{cores}$ y $\sum_i n_i^{LLC} \leq N_{LLC}$. Definimos c_i como el uso computacional de vBS i. Definimos $x_i := \left(d_i^{UL}, d_i^{DL}, s_i, m_i^{UL}, m_i^{DL}\right)$ como el contexto de la vBS i. Además, definimos f_i como la función que asigna x_i y n_i^{LLC} al uso de recursos computacionales c_i.

Por último, definimos el vector $\mathcal{X}$ que concatena los vectores de contexto de todas las vBS como $\mathcal{X} := \left(x_1, x_2, \ldots, x_{N_{vBS}}\right)$. Además, definimos $\mathcal{P} := \left(P_1, \ldots, P_{N_{vBS}}\right)$ y $\mathcal{N} := \left(n_1^{LLC}, \ldots, n_{N_{vBS}}^{LLC}\right)$ como los vectores con la asignación del conjunto de núcleos y la asignación de las vías de caché LLC en una plataforma vRAN. Del mismo modo, definimos la función f que asigna $(\mathcal{X}, \mathcal{N})$ al uso total de computación $C^{vRAN} \in [0, M_{cores}]$ de la plataforma vRAN.

Definimos el problema de optimización del conjunto de computación y la asignación de LLC como:

$$\begin{aligned} \min_{\mathcal{N}} \quad & f(\mathcal{X}, \mathcal{N}) \\ \text{subject to} \quad & \sum_i n_i^{LLC} = N_{LLC}. \end{aligned}$$

Como se analiza en el entregable SORUS-RAN-A2.1-E2, el uso de la computación y el consumo de energía son proporcionales. Por lo tanto, la minimización de la función de uso de recursos computacionales f también minimiza el consumo total de energía.

Tenga en cuenta que, en nuestro problema, la asignación de núcleos de CPU a vBS $\mathcal{P}$ ya está dada. Seleccionamos un valor fijo de $\mathcal{P}$ basándonos en nuestros conocimientos experimentales y en trabajos anteriores sobre este tema[2] [19].

Obsérvese que, en la formulación problema, la asignación óptima de la caché LLC depende del contexto $\mathcal{X}$, cuya dimensionalidad aumenta en función del número de vBSs. Además, la acción óptima también depende del número de vBS activas, que puede cambiar con el tiempo. Para evitar la carga de exploración de los algoritmos de aprendizaje (RL, por ejemplo) que pueden conducir a configuraciones subóptimas, descomponemos el problema y utilizamos un Digital Twin (DT).

[19] Lozano, J. X. S., Garcia-Saavedra, A., Li, X., & Perez, X. C. (2023). AIRIC: Orchestration of Virtualized Radio Access Networks with Noisy Neighbours. IEEE Journal on Selected Areas in Communications.

5.2. Algoritmo propuesto: MemorAI

Para resolver el problema formulado arriba, proponemos un framework de optimización que llamamos MemorAI. Este marco considera intervalos de decisión discretos denotados por $t \in \{1, 2, \dots, T\}$. Al principio de cada intervalo de decisión, nuestra solución recibe $\mathcal{X}^{(t)}$, y decide la asignación óptima de LLC $\mathcal{N}^{*(t)}$ entre todas las vBS, para minimizar el consumo de energía. MemorAI está compuesto por un conjunto de DTs y un clasificador basado en NNs. La Figura 11 muestra la arquitectura de la solución propuesta.

5.3. Digital Twin

Gracias al aislamiento completo de los vBS mediante su anclaje a núcleos dedicados y la asignación de vías de caché LLC, observamos que $C^{vRAN} = \sum_i c_i$, ya que no hay efectos conjuntos entre los vBS. Esta observación implica que $f(\mathcal{X}, \mathcal{N}) = \sum_i f_i(x_i, n_i^{LLC})$. Dado que no hay interacción entre las vBS en términos de uso de computación, podemos crear DT de vBS independientes. Así, podemos reflejar su comportamiento en un entorno seguro y controlado para el entrenamiento de algoritmos de machine learning.

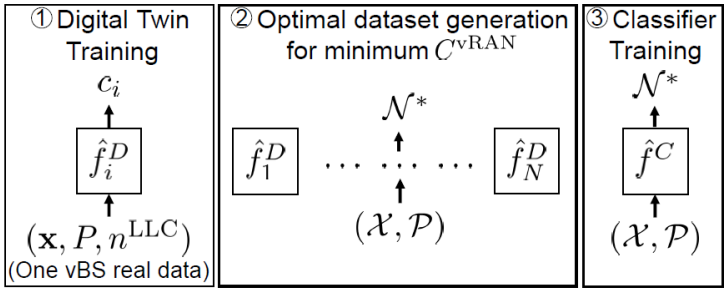

Figura 11: Arquitectura de la solución propuesta (MemorAI)

Cada DT puede modelar las particularidades de cada vBS (por ejemplo, diferentes implementaciones de pilas de protocolos) y podemos emular las complejas interacciones del sistema vRAN completo. Creamos un DT utilizando los datos operativos de un vBS. Utilizando los DT de cada vBS, podemos reducir el coste de tiempo para generar un conjunto de datos para una serie de vBSs, ya que agregamos los resultados de cada DT. Tenga en cuenta que sin los DT, la cantidad de datos necesarios para crear un conjunto de datos de entrenamiento aumenta exponencialmente con el número de vBS (maldición de la dimensionalidad).

Para crear el DT de una vBS utilizamos nuestra plataforma vRAN para generar un conjunto de datos de entrenamiento con un conjunto fijo de núcleos de computación $|P|$. Obsérvese que

seleccionamos un conjunto de núcleos tal que el vBS pueda funcionar correctamente. Esto hace que f_i sea una función continua. Cada muestra del conjunto de datos contiene una tupla de cuatro con (x, P, n^{LLC}, c). Con este conjunto de datos, construimos un DT utilizando una NN que aproxima c utilizando (x, P, n^{LLC}), es decir, aproxima f_i minimizando el MSE. Denotamos la función DT como $\hat{f}_i^D$. La parte izquierda de la Figura 11 muestra el paso de entrenamiento del DT.

5.4. Clasificador basado en redes neuronales

El DT nos permite evaluar el uso de CPU de diferentes configuraciones con gran precisión sin tener que utilizar el sistema real. Por lo tanto, podemos realizar una búsqueda exhaustiva para encontrar la asignación óptima de LLC $\mathcal{N}^*$ para un conjunto de contextos $(\mathcal{X}, \mathcal{P})$. Nótese que el tamaño del conjunto de posibles asignaciones de caché LLC es $|\mathcal{N}| = \binom{N_{LLC}-1}{N_{vBS}-1}$. La parte central de la Figura 11 muestra cómo utilizando los diferentes DTs generamos el conjunto de datos anterior.

Utilizando este conjunto de datos, entrenamos una NN totalmente conectada (fully connected) para predecir $\mathcal{N}^*$ para un contexto dado $(\mathcal{X}, \mathcal{P})$. En concreto, resolvemos un problema de clasificación multiclase utilizando la función de pérdida de entropía cruzada. Denotamos la función clasificadora como $\hat{f}_i^C$. Obsérvese que nuestra solución es muy flexible a los cambios en el sistema (por ejemplo, actualizaciones en la implementación de la pila de software, despliegue de nuevos vBS, etc.). En esos casos, después de haber modelado el DT, podemos fácilmente volver a entrenar el clasificador NN offline sin degradar el rendimiento del sistema. Por último, la parte derecha de la Figura 11, muestra el paso de entrenamiento del clasificador de nuestro marco de optimización.

5.5. Evaluación experimental

En esta sección, evaluamos el rendimiento y el ahorro potencial energético de nuestro enfoque. Llevamos a cabo la evaluación para un despliegue de $N_{vBS} = 5$ vBS. Utilizamos PyTorch para implementar el DT y el clasificador.

5.5.1. Evaluación de la fase de entrenamiento

5.5.1.1. Digital Twin

Implementamos el DT de una vBS utilizando una NN con tres capas ocultas de tamaños {256, 128, 64} respectivamente con una función de activación ReLU. Además, detenemos las iteraciones de entrenamiento utilizando un mecanismo de parada temprana para evitar el sobreajuste. El mecanismo de parada temprana detiene el entrenamiento después de que el valor de la función de

pérdida en un conjunto de datos de validación no haya mejorado durante las últimas iteraciones de entrenamiento N (usualmente denominada como paciencia).

La Figura 12 muestra el valor de pérdida MSE en los conjuntos de datos de entrenamiento y validación del DT en función de las iteraciones de entrenamiento. Seleccionamos una paciencia de $N = 10$ y entrenamos el modelo durante 70 iteraciones.

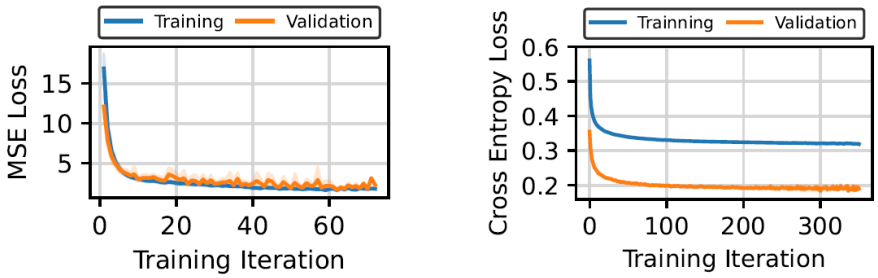

Figura 12: Coste MSE del entrenamiento del Digital Twin (izquierda), coste cross entropy del clasificador basado en redes neuronales (derecha)

5.5.1.2. Clasificador basado en redes neuronales

Por otro lado, implementamos el clasificador como una NN con 4 capas ocultas totalmente conectadas de tamaños {512, 384, 384, 512} respectivamente. Cada capa también utilizó una función de activación ReLU e introdujimos una probabilidad de dropout de 0,2 durante el entrenamiento. También utilizamos el mecanismo de parada temprana con $N = 50$ de paciencia.

La Figura 12 (derecha) muestra la pérdida de entropía cruzada en el conjunto de datos de entrenamiento y validación en función del número de iteraciones de entrenamiento. Entrenamos el modelo hasta la iteración 300 (debido al mecanismo de parada temprana). El coste de entrenamiento es mayor que el coste de validación debido a las capas de dropout. Además, el clasificador alcanza una precisión del 92,1% en nuestro conjunto de datos de test.

5.6. Evaluación de rendimiento

Diseñamos MemorAI para operar en el non-RT RIC de la arquitectura O-RAN como una rApp[20]. Basándonos en esto, seleccionamos una granularidad temporal de 15 minutos para nuestra

[20] O-RAN Alliance, "O-RAN Non-RT RIC Architecture 3.0 (ORAN.WG2.Non-RT-RIC-ARCH-R003-v03.00)," Technical Report, Jun. 2023.

evaluación[21]. Generamos un conjunto de datos con datos de contexto aleatorios y comparamos nuestra solución contra los siguientes benchmarks:

- **Aleatorio**: seleccionamos la asignación de vías de caché para cada vBS aleatoriamente.
- **Partición igualitaria**: a todos los sistemas vBS se les asigna el mismo número de vías de caché. Las vías de caché adicionales se dejan sin asignar.
- **Partición ponderada**: a cada vBS se le asigna un número de vías de caché proporcional a su demanda total:

$$n_i^{LLC} = \frac{d_i^{\mathrm{UL}} + d_i^{\mathrm{DL}}}{\sum_j d_j^{\mathrm{UL}} + d_j^{\mathrm{DL}}} \cdot N_{\mathrm{LLC}}$$

La Figura 13 (izquierda) muestra el ahorro de energía en kilojulios (kJ) de nuestra solución y la estrategia óptima con respecto a los distintos puntos de referencia en un intervalo de decisión de 15 minutos. Nuestra solución supera a los tres puntos de referencia en términos de energía, mientras que produce casi los mismos resultados que la solución óptima. Así pues, MemorAI entiende mejor la utilidad de la partición de la caché LLC que las estrategias de los puntos de referencia. Nuestra solución produce ahorros superiores de hasta 1 kJ en comparación con la estrategia aleatoria, de hasta 0,35 kJ en comparación con el particionamiento igual, y de hasta 0,36 kJ en comparación con la estrategia ponderada. Obsérvese que, aunque la estrategia ponderada muestra un menor consumo de energía, no escala correctamente la caché LLC cuando existe un desequilibrio entre las demandas UL y DL. En estos casos, nuestro enfoque consigue las mayores ganancias con respecto a esta estrategia.

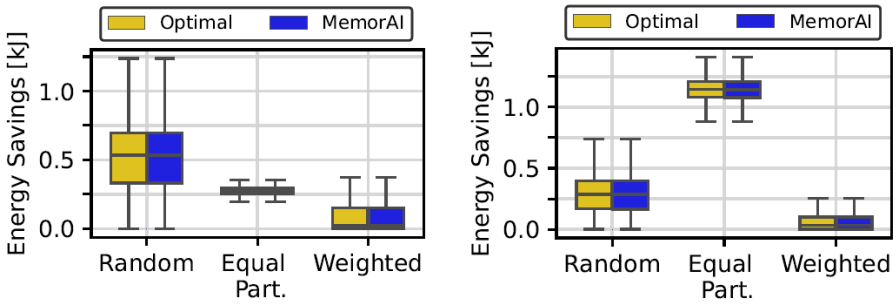

Figura 13: Ahorro de energía comparado con diferentes benchmarks para un intervalo de decisión de 15 minutos y diferente número de vías de caché: 12 (izquierda) y 8 (derecha)

[21] C. Marquez et al., "How should i slice my network? a multi-service empirical evaluation of resource sharing efficiency," in Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, 2018, pp. 191–206.

Por último, la Figura 13 (derecha) muestra las ganancias obtenidas cuando el sistema sólo dispone de 8 vías de caché. En ese caso, el enfoque aleatorio obtiene mejores resultados porque el número de configuraciones es menor y, por tanto, la probabilidad de seleccionar la configuración óptima es mayor. El ahorro en comparación con el enfoque de particiones iguales es mayor, ya que en este escenario el benchmark sólo puede asignar una vía de caché por sistema de base de datos virtual. En comparación con la estrategia ponderada, seguimos observando un mayor ahorro.

6 CONCLUSIONES

Para abordar la problemática de los noisy neighbours en las vRAN, hemos desarrollado AIRIC, un sistema diseñado para adaptarse dinámicamente a diversos contextos mediante reconfiguración basada en Reinforcement Learning (aprendizaje por refuerzo). AIRIC implementa una arquitectura sofisticada que combina dos etapas principales: una Relational Network y, posteriormente, una Deep Q-Network. Estas etapas trabajan en conjunto para gestionar de manera eficiente los recursos compartidos a lo largo del tiempo, optimizando así el rendimiento del sistema.

Los resultados de los experimentos realizados con AIRIC son muy alentadores. Indican que nuestro sistema ajusta de manera adecuada el conjunto de núcleos de computación, evitando así la disminución del rendimiento de las vBS al anticipar con precisión el problema de los noisy neighbours. De hecho, la eficacia de nuestra solución es comparable a la obtenida por un oráculo, lo que significa que logramos mitigar el problema sin incurrir en una pérdida significativa de rendimiento. Además, AIRIC propicia una reducción global del uso de recursos computacionales del 17%, lo que demuestra su capacidad para mejorar la eficiencia operativa del sistema.

Es importante destacar que los experimentos realizados con AIRIC se llevaron a cabo en escenarios realistas, lo que aumenta la relevancia y validez de nuestros resultados. En vista de estos logros, consideramos que AIRIC es una solución sólida y práctica para abordar los desafíos de los noisy neighbours en entornos vRAN, y tiene el potencial de ser implementado en aplicaciones reales con éxito.

Además de AIRIC, hemos desarrollado MemorAI en respuesta a los problemas de compartición de memoria caché LLC identificados en el entregable SORUS-RAN-A2.1-E2. Este framework, basado en un árbol de decisión y un clasificador de redes neuronales, ofrece una solución efectiva para la asignación óptima de la caché LLC. Nuestros experimentos muestran que MemorAI puede lograr ahorros significativos de energía, reduciendo el consumo entre 0,35 kJ y 1 kJ en comparación con

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810

diversas alternativas de asignación de vías de caché. Esto subraya aún más el potencial de nuestras soluciones para mejorar tanto el rendimiento como la eficiencia energética de los sistemas vRAN.

Financiado por el Programa "Next Generation" de la Unión Europea en el marco del acuerdo n.º 2022/0004810