Comparación entre algoritmos evolutivos y aprendizaje por refuerzo para autoescalado de workflows en Cloud

Luciano Robino*[1,2,3], Yisel Garí†[2,3], Elina Pacini‡[2,3], Cristian Mateos§[1,3], Virginia Yannibelli‖[1,3], and David A. Monge#[4]

[1] ISISTAN-UNICEN-CONICET. Tandil, Buenos Aires, Argentina
[2] Laboratorio de Sistemas Inteligentes (LABSIN) - Facultad de Ingeniería, UNCuyo. Mendoza, Argentina
[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET). Argentina
[4] Universidad Nacional de Cuyo. Mendoza, Argentina

Resumen En los últimos años, muchos experimentos científicos son realizados por medio de workflows científicos. Estas tecnologías facilitan la realización de experimentos que son computacionalmente intensivos, y que muchas veces requieren ser ejecutados en Clouds públicas. Esto hace que optimizar la ejecución de estas aplicaciones sea un problema desafiante debido a que la virtualización de recursos en Cloud crea necesidades de planificación a la vez de representar incertidumbre en la ejecución. Por este motivo, se han usado heurísticas y metaheurísticas para este problema. En particular, se ha intentado resolver el problema usando técnicas de Aprendizaje por Refuerzo y algoritmos evolutivos. En este trabajo se presenta un problema markoviano de decisión para resolver este problema desde el punto de vista de Aprendizaje por Refuerzo. En conjunto con este modelado, se presenta también una variación que permite abordar el mismo problema como un algoritmo evolutivo multiobjetivo. Estas dos estrategias son comparadas usando 4 workflows de referencia de la literatura, utilizando el simulador CloudSimPlus y máquinas virtuales presentes en Amazon. Para este análisis se estudia el costo monetario de ejecución, el tiempo total de ejecución (makespan) y la norma L2 de estas dos métricas.

Keywords: Aprendizaje por Refuerzo, Algoritmo Evolutivo, Cloud Computing, Workflow Científico

*ORCID: 0009-0002-2110-6026. e-mail: lrobino@conicet.gov.ar
†ORCID: 0000-0003-4353-7257. e-mail: ygari@uncu.edu.ar
‡ORCID: 0000-0003-2882-766X. e-mail: epacini@uncu.edu.ar
§ORCID: 0000-0001-5761-1898. e-mail: cristian.mateos@isistan.unicen.edu.ar
‖ORCID: 0000-0001-7854-7610. e-mail: virginia.yannibelli@isistan.unicen.edu.ar
#ORCID: 0000-0001-6444-4610. e-mail: dmonge@uncu.edu.ar

Comparison between evolutionary algorithms and Reinforcement Learning for workflow autoscaling on Cloud

Luciano Robino[1,2,3], Yisel Garí[2,3], Elina Pacini[2,3], Cristian Mateos[1,3], Virginia Yannibelli[1,3], and David A. Monge[4]

[1] ISISTAN-UNICEN-CONICET. Tandil, Buenos Aires, Argentina
[2] Laboratorio de Sistemas Inteligentes (LABSIN) - Facultad de Ingeniería, UNCuyo. Mendoza, Argentina
[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET). Argentina
[4] Universidad Nacional de Cuyo. Mendoza, Argentina

Resumen In recent years, many scientific experiments have been performed using scientific workflows. These technologies facilitate the performance of computationally intensive experiments, which often require to be executed on public Clouds. Optimizing these applications becomes a challenging problem since resource virtualization demands scheduling needs to be satisfied, at the same time it has to deal with uncertainties during execution. For this reason, heuristic and metaheuristic solutions have been proposed to this problem. Indeed, Reinforcement Learning and evolutionary algorithms have been used to tackled this problem. This paper presents a Markovian Decision Problem that can be used to solve this problem using Reinforcement Learning. Additionally, a variation of this modeling is introduced to solve the same problem using multi-objective evolutionary algorithms. These two strategies are compared using 4 benchmark workflows using the simulator CloudSimPlus and virtual machines present on Amazon public clouds. Total monetary cost of the execution, total execution time (i.e. makespan), and the L2 norm of these two quantities are used for the comparative analysis.

Keywords: Reinforcement Learning, Evolutionary Algorithm, Cloud Computing, Scientific Workflow

1. Introducción

Los experimentos científicos requieren grandes capacidades de computo debido a la complejidad de los problemas que abordan. Frente a esto, los workflows científicos han surgido como una alternativa para estructurar estos experimentos. Además, las infraestructuras de *Cloud Computing* han sabido responder a las necesidades de workflows científicos. La elasticidad de los servicios de *Cloud Computing* permite adaptar dinámicamente los recursos usados por los workflows según las necesidades al momento de la ejecución (Mell et al., 2011).

Una buena estrategia de asignación de recursos es aquella que sabe adaptar estos recursos a las necesidades de ejecución buscando minimizar una función

objetivo. Por ejemplo, se puede buscar la minimización del tiempo de ejecución o la minimización de los costos asociados a la ejecución. Este aprovisionamiento de recursos y la posterior asignación de esos recursos a cada tarea del workflow buscando minimizar alguna de las cantidades anteriores es un problema conocido como autoescalado o *autoscaling* (Mao & Humphrey, 2013; Monge et al., 2017). Este problema pertenece al conjunto de los problemas NP-duro, por lo que la búsqueda de soluciones ha sido explorada a través del uso de técnicas heurísticas y metaheurísticas (Monge et al., 2018).

Los servicios de *Cloud* públicos[‖] ofrecen a sus usuarios máquinas virtuales (MV) que pueden ser adquiridas según sus necesidades. Esto se logra a partir de estrategias de virtualización de los recursos, lo cual genera una incertidumbre en el rendimiento de la infraestructura contratada de un 20 % (Ericson et al., 2017; Pu et al., 2010; Schad et al., 2010). Anteriormente, el Aprendizaje por Refuerzo (AR) (Sutton & Barto, 2018) fue utilizado en el problema del autoescalado (Garí et al., 2022, 2024) dado que permite trabajar en entornos con alta incertidumbre.

En Chellapilla y Fogel (2001) se propone la búsqueda de agentes por medio de algoritmos evolutivos (AE). En este sentido, se busca que los AE encuentren funciones aproximadoras que permitan identificar las mejores acciones de un agente para un estado del sistema. Cabe señalar que este modelado de agentes es similar a AR y podría usarse en problemas de autoescalado.

El objetivo del presente trabajo consiste en utilizar AE usando las ideas de Chellapilla y Fogel (2001) para crear agentes de autoescalado de ejecuciones de *workflows* científicos en *Cloud*. El resto del trabajo está estructurado de la siguiente manera. En la sección , se describe el contexto del problema que motiva al presente trabajo. Allí también se introducen las técnicas a comparar. En la sección 3, se describe el modelado de las políticas de autoescalado a encontrar por métodos evolutivos. En la sección 4, se discuten los resultados obtenidos, y para poder analizar la calidad de los nuevos resultados obtenidos, se los compara con los resultados de (Garí et al., 2022). Finalmente, en la sección 5 presentan las conclusiones y se discuten los trabajos futuros.

2. Contexto del Problema

Desde casi el inicio de la computación y hasta hoy, el aumento de la potencia computacional disponible ha permitido lograr ejecutar eficientemente gran cantidad de aplicaciones. En este contexto, los workflows científicos se han establecido como una importante abstracción para el procesamiento de datos y la ejecución de grandes y complejos experimentos (Liu et al., 2016; Meade & Fluke, 2018; Vandenbrouck et al., 2019). Las tecnologías de workflows han permitido acelerar el desarrollo de aplicaciones científicas porque desacoplan los saberes disciplinares del experimento del know-how específico de ciencias de la computación.

[‖]Por ejemplo, Amazon EC2 https://aws.amazon.com/ec2, Microsoft Azure https://azure.microsoft.com/en-us/, Google Cloud Platform https://cloud.google.com/

En general, un workflow define un objetivo complejo que es alcanzado mediante la realización de un conjunto de tareas que poseen dependencias entre sí. Además, las características de las tareas y las dependencias determinan una carga de trabajo variable durante la ejecución. Es decir, en diferentes momentos de la ejecución, múltiples tareas pueden ser ejecutadas en paralelo, mientras que en otros momentos las tareas deben ser ejecutadas de forma secuencial. Además, cada tarea del workflow requerirá distintos tipos de recursos computacionales, como por ejemplo, poder de procesamiento, ancho de banda o memoria.

El paradigma *Cloud Computing* ha sido aplicado a workflows científicos principalmente por su elasticidad a la demanda. *Cloud* permite un acceso ubicuo, conveniente y bajo demanda a través de la red a un conjunto compartido de recursos configurables que pueden ser aprovisionados o liberados rápidamente con un mínimo esfuerzo de gestión o interacción del proveedor de servicios (Mell et al., 2011).Un usuario puede decidir aumentar o disminuir el número de CPUs a utilizar, o la cantidad de instancias según la necesidad de la aplicación. Para poder ofrecer a los usuarios distintos tipos de recursos, *Cloud* utiliza tecnologías de virtualización. Luego, las MVs están disponibles para ser adquiridas por medio de un esquema de pago por su uso.

Las *Cloud* públicas permiten modificar de manera dinámica los recursos disponibles y ajustarlos según la demanda durante la ejecución de workflows. Esta modificación dinámica permite introducir lo que se conoce como el problema del autoescalado que ha sido abordado en la literatura reciente (Garí et al., 2019, 2022; Monge & Garino, 2014; Monge et al., 2017, 2018, 2020).

Dado un conjunto de tareas a ejecutar, un número de recursos de diferentes tipos, y ciertos objetivos de optimización, el autoescalado se define a través de dos subproblemas de optimización interrelacionados:

- Se llama *escalado* (scaling) al problema de decidir cuántos recursos de cada tipo se necesitan en cada momento de acuerdo a la demanda de la aplicación.
- Se llama *planificación* (scheduling) al problema de decidir la asignación de las tareas a los recursos específicos para su ejecución.

Es muy importante señalar que el problema queda definido una vez fijado el objetivo a optimizar. Por ejemplo, en el contexto de workflows en *Cloud*, se puede buscar minimizar el tiempo de ejecución o el costo de asociado al uso de diferentes MVs, entre otros.

Se sabe que ambos subproblemas de autoescalado son *NP-duros* (Mao & Humphrey, 2013), y en particular, ambos problemas han sido abordados usando Aprendizaje por Refuerzo (AR) (Garí et al., 2021). Por medio de AR se genera un agente que aprende un comportamiento adecuado para optimizar un determinado objetivo. Este aprendizaje ocurre mediante la interacción del agente con el entorno. Por cada *acción* tomada por el agente, se observa su impacto a través de una señal numérica de *recompensa*. Al mismo tiempo, las acciones del agente, pueden modificar el entorno (Sutton & Barto, 2018). Cuando esto ocurre se dice que el sistema ha pasado de un *estado* a otro estado. El análisis estadístico de estos factores permiten que el agente aprenda a definir qué acciones tomar en

qué momento. Esta decisión está representada con las *políticas* que son aprendidas durante el entrenamiento. En Garí et al. (2022) se evaluaron de políticas de autoescalado de workflows científicos generadas por AR en entornos *Cloud*.

Cabe señalar que es difícil realizar estos estudios en infraestructuras Cloud reales ya que los tiempos de entrenamiento serían del orden de años. Por este motivo, se utilizan simuladores como CloudSimPlus (Silva Filho et al., 2017). Al recurrir a simulaciones, un trabajo que demoraría horas en un servidor real, transcurre en unos pocos segundos dentro del simulador. De esta manera, se puede entrenar una política en unos pocos minutos. Otra ventaja de usar simulaciones, es que los tiempos de entrenamiento son menores y por lo tanto el consumo de energía es menor. Esto es muy importante, dado que nuevos desarrollos en Machine Learning han dado lugar a mayores preocupaciones en el impacto ambiental de estos experimentos (Ding & Shi, 2024).

En el presente trabajo se propone utilizar un algoritmo genético para la creación de estos agentes. Como se dijo anteriormente, la optimización de autoescalado de workflows es un problema desafiante debido a lo complejo del problema y a la dinámica del mismo. Por esta razón, las políticas deben ser eficientes y adaptativas. Garí et al. (2021) señala que AR es una técnica muy usada para autoescalado, pero también lo es la computación evolutiva. De hecho, los autores de este trabajo señalan al menos 3 trabajos que combinan ambas técnicas.

La computación evolutiva tiene muchísima relevancia en la búsqueda de soluciones en grandes espacios de búsqueda garantizando gran diversidad y calidad de las soluciones encontradas. Por otro lado, AR es altamente efectivo en el aprendizaje adaptativo basado en la interacción con ambientes dinámicos. Por lo anterior, resulta interesante comparar los resultados entre algoritmos genéticos y AR para entender sus limitaciones. En la siguiente sección se describe cómo este problema de decisión markoviano fue modificado para crear un agente que pudiera ser encontrado por métodos evolutivos.

3. Modelado del problema

Para la obtención de políticas para el problema de autoescalado se partió de un problema de decisión markoviano conocido (Garí et al., 2022). Sobre este problema, se plantearon modificaciones para poder transformarlo en un problema de optimización basado en algoritmos evolutivos (AE). En las siguientes subsecciones se presentan las características de estos dos espacios de estados y acciones. En primer lugar, se presenta el espacio de estados y su discretización. Luego, se describe el espacio de acciones, así como las limitaciones existentes a las acciones para cada estado. Tras esto, se introduce esquemáticamente qué es un AE. Finalmente, se describe el problema de optimización presentado y cuál es la codificación de las políticas usadas como solución.

3.1. Espacio de estados

En Garí et al. (2022) se propuso un modelo para definir el estado de la ejecución de un workflow basado en la estructura de dependencias, las tareas en

ejecución y las máquinas virtuales (MV) en uso. Esta caracterización se basa en las variables:

- N: Número de tareas listas para ejecutar en el workflow. Esas tareas son aquellas tareas para las cuales todas sus dependencias han sido completadas.
- F: Número de tareas listas para ejecutar que son dependencias de más de una tarea. Se usa F para señalar que son tareas con una estructura tipo *fork*. En la figura 3.1 se corresponden con la estructura llamada *distribution*.
- J: Número de tareas listas para ejecutar que tienen más de una tarea como dependencia. Se usa J para señalar que estas tareas tienen una estructura tipo *join*. En la figura 3.1, las estructura tipo *join* son llamadas *aggregation*.
- P: Número de tareas listas para ejecutar que dependen y son dependencia de una única tarea. Se usa P para señalar que estas tareas tienen una estructura tipo *pipeline*. En la figura 3.1 se muestra una estructura *pipeline*.
- I: Número de MVs listas para ser usadas durante la ejecución. Se usa I para hacer referencia a la infraestructura contratada para la ejecución.

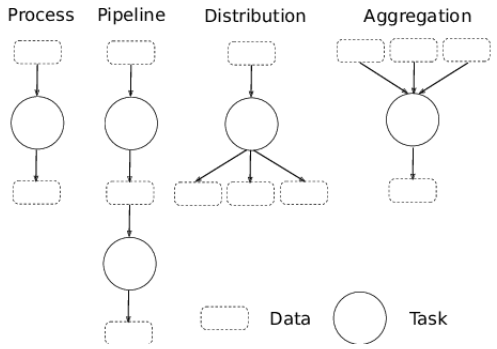

Figura 1. Representación gráfica de las tareas en un *workflow*. Se muestran las estructuras *fork*, *pipeline* y *join* con los nombres *distribution*, *pipeline* y *aggregation*, respectivamente

A partir de las variables N, F, J, P, I se definen 5 nuevas variables:

$$f = \frac{F}{N} \qquad j = \frac{J}{N} \qquad p = \frac{P}{N}$$
$$i = \begin{cases} \text{si } I > 0 & \frac{N}{I} \\ \text{si } I = 0 & -1 \end{cases} \qquad m = \begin{cases} \text{si } I \geqslant 60 & 1 \\ \text{si } I < 60 & 0 \end{cases} \tag{1}$$

En la ecuación 1 se observa, $i = \frac{N}{I}$ solo si el número de MVs listas para usar es mayor a 0. De lo contrario, $i = -1$. Adicionalmente, m representa que se ha llegado a la cantidad máxima de MVs disponibles para alquilar. Los workflows usados tienen 100 tareas a ejecutar. Al elegir 60 MVs como el número máximo de tareas a usar se evita que se seleccione una MV por tarea. Además 60

MVs representa más del 50 % de las tareas. Las variables f, j, p, i son variables continuas; esto lleva a que existan infinitos estados.

Para evitar esto y trabajar con una cantidad discreta de estados en Garí et al., 2022 se realizó una discretización de la siguiente forma:

$$f = \begin{cases} \text{si } f = 0 & \text{None} \\ \text{si } 0 < f \leqslant \frac{1}{3} & \text{Low} \\ \text{si } \frac{1}{3} < f \leqslant \frac{2}{3} & \text{Medium} \\ \text{si } \frac{2}{3} < i \leqslant 1 & \text{High} \end{cases} \tag{2}$$

La discretización de la ecuación 2 aplica también a las variables j y p.

Adicionalmente, la variable i fue discretizada siguiendo la ecuación 3:

$$i = \begin{cases} \text{si } i = -1 & \text{None} \\ \text{si } 0 < i < 1 & \text{Low} \\ \text{si } i = 1 & \text{Medium} \\ \text{si } i > 1 & \text{High} \end{cases} \tag{3}$$

Dado que la variable m ya es discreta (solo toma valores 0 y 1), ahora el espacio de estados S tiene forma definida por la ecuación 4:

$$\begin{array}{l} A = \{\text{None}, \text{Low}, \text{Medium}, \text{High}\} \\ B = \{0, 1\} \\ S = A^4 \times B \end{array} \tag{4}$$

De la ecuación 4 se puede observar que $|S| = 512$

3.2. Espacio de acciones

En el trabajo presentado en (Garí et al., 2022) se realizó la implementación de 3 acciones de dos tipos:

- Acciones de escalado: son acciones donde se alquila una MV adicional para la ejecución, es decir,aumenta la infraestructura disponible para las tareas que se encuentren listas para ejecutarse. Se consideran dos tipos de MVs:
  - T1: Se contrata una MV tipo `t2.micro` del servicio Amazon EC2. Esta MV consta de 1 vCPU, su uso por hora cuesta 0.013 USD y tiene un poder de computo de 1 ECU. Un ECU es una unidad llamada EC2 Computing Unit (ECU). Esta medida es usada por Amazon para estimar las características de sus MVs en términos de su poder de cómputo.
  - T3: Se contrata una MV tipo `c3.2xlarge`. Consta de 8 vCPU. su costo por hora es de 0.42 USD y tiene un poder de computo total de 28 ECU, es decir 3.5 ECU por vCPU
- Acciones de planificación: son acciones donde se envía a ejecutar las tareas listas para ejecutar a los recursos que se tienen disponibles en un determinado momento. En el modelo actual existe una sola acción llamada ECU. En esta acción se eligen MVs para cada tarea, priorizando los ECU altos.

Limitaciones a las acciones Como se dijo anteriormente, existen dos tipos de acciones, acciones de escalado y acciones de planificación. Conviene señalar que ambas acciones están limitadas por el estado en el que se encuentra el sistema. Es decir, no todas las acciones pueden llevarse a acabo para cada estado.

La limitación más intuitiva es el caso de la variable m: esta variable informa si se ha alcanzado o no el número máximo de MVs a usar. En estudios anteriores se observó que cuando este valor es de 60 se obtienen buenos resultados (Garí et al., 2022). En este sentido, las acciones de escalado quedan excluidas cuando $m = 1$, ya que no se dispone de recursos.

La otra limitación está asociada a las acciones de planificación. Cuando no hay MVs alquiladas, carece de sentido ejecutar una acción de planificación. Por lo tanto, las acciones de planificación quedan excluidas cuando $i = \text{None}$.

Dada la forma en que m e i dependen de la infraestructura contratada, no existen estados con $m = 1$ y $i = \text{None}$. Esto sería equivalente a tener 60 MVs contratadas (para satisfacer $m = 1$) al mismo tiempo que no tener contratada ninguna MV (para satisfacer $i = \text{None}$)

De esta manera, las acciones permitidas quedan representadas en la figura 2.

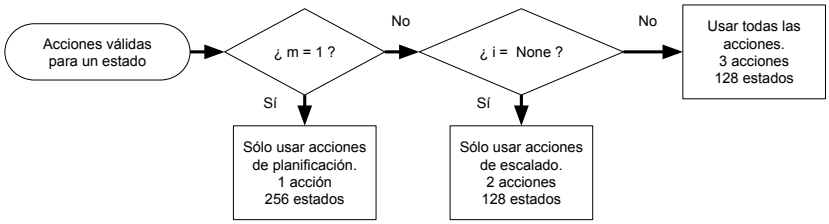

Figura 2. Acciones válidas para el sistema en función de los estados. Existe una única acción de planificación. Por lo tanto cuando solo se pueden usar acciones de planificación, existen 256 estados donde el comportamiento es determinista.

3.3. Algoritmos evolutivos

Los algoritmos evolutivos (AE) son un tipo de metaheurística para problemas de optimización que se inspiran los procesos evolutivos por selección natural de las especies. En líneas generales, un algoritmo evolutivo consta de la estructura señalada en el Algoritmo 1. Existe un proceso de generación de soluciones iniciales de manera aleatoria. Luego, se evalúa la aptitud de estas soluciones, es decir, se analiza qué tanto optimiza cada solución el problema. A partir de estos valores de aptitud, se eligen soluciones para ser recombinadas. El proceso de recombinación consiste en tomar información de una o más soluciones padres para generar nuevas soluciones hijas buscando que tengan lo mejor de las soluciones padres. De esta manera, el proceso de recombinación es un proceso de

explotación en un esquema de explotación-exploración, usa información conocida para mejorar las soluciones. Seguido al proceso de recombinación, ocurre un proceso de mutación, es decir, se modifica una de las variables o genes de manera aleatoria. Al agregar nueva información a la población, se dice que este proceso de mutación es un paso de exploración, en tanto se obtiene nueva información de las posibles soluciones del problema.

Una vez generadas las nuevas soluciones hijas, se elige cuáles de ellas son las mejores que reemplazaran a las soluciones de la población original. De esta manera, se obtiene por analogía un esquema de selección del más apto en la versión más simplista de la teoría evolutiva. Este proceso es repetido varias veces como ocurren los cambios de las especies a los largo de varias generaciones.

Algoritmo 1 Evolutive Algorithm

```
1: procedure EVOLUTIVE(maxGenerations, population):
2:     Initialize population
3:     Evaluate population using FitnessFunction
4:     for i ← 1 to maxGenerations do
5:         Select best fit parents from population
6:         Recombine parents with CrossoverOperator creating offsprings
7:         Mutate offsprings with MutationOperator
8:         Evaluate offsprings using FitnessFunction
9:         Select fittest offsprings and population creating new population
```

A continuación se presenta cómo se encuentra codificada la solución del problema de optimización. Los detalles de implementación del algoritmo se discuten en la siguiente sección.

3.4. Problema de optimización

El problema de minimización multiobjetivo está dado por: mín (makespan, cost) Donde mín hace referencia al operador minimización multiobjetivo, que devuelve un conjunto de soluciones donde mejorar uno de los objetivos empeora otro de los objetivos. Esto da lugar a conjuntos Pareto dominantes. Esta minimización es llevada a cabo sobre las soluciones al problema de autoescalado. En el contexto del problema markoviano de decisión propuesto, se utiliza un simulador basado en CloudSimPlus para calcular el costo monetario de la ejecución (*cost*) y el tiempo de ejecución (*makespan*)

La política solución deseada debe ser una política que para cada estado del sistema detecte cuál es la mejor acción posible a realizar. Como se mencionó en las subsecciones anteriores, el problema tiene 512 estados y 3 acciones. Sin embargo, cuando $m = 1$, solo está disponible la acción de planificación. De esta manera, hay 256 estados que tienen solo una única acción posible. Dicho de otro modo, hay 256 estados donde el sistema tiene su evolución definida, pues solo se permite una acción.

Este hecho permite reducir la representación de las políticas. En lugar de tener 512 estados con 3 acciones, ahora solo se necesitan 256 estados con 3 acciones, pues en los 256 estados restantes solo es posible una única acción. De esta manera, la política puede ser representada por un arreglo de enteros de 256 elementos, donde cada elemento es un entero entre 0 y 2 (incluido). Esto da a lugar a un espacio de búsqueda de 3^256 posibles políticas.

Dada la longitud de la solución y que se utilizan números enteros para este problema se utilizaron los operadores de recombinación *uniform recombination* y de mutación (random resetting), que se explican en la figura 3. El operador de mutación aplica sobre un elemento del arreglo usado para representar políticas, es decir, sobre un gen.

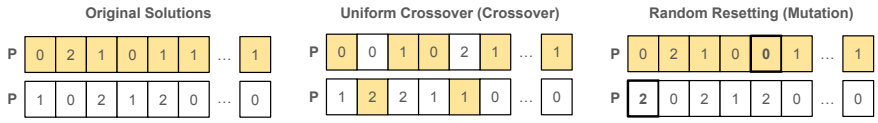

Figura 3. Operadores de recombinación y de mutación usados en este trabajo. *Uniform crossover* es un tipo de operador que toma dos soluciones y crea dos soluciones nuevas intercambiando elementos con probabilidad P_c. *Random Resetting* es un operador de mutación que cambia un elemento a un nuevo valor con probabilidad P_m

Para para la búsqueda de soluciones se utilizó una versión del Algoritmo NSGA-III (Deb & Jain, 2014) incluido en el *framework* MOEA. NSGA-III es un algoritmo de optimización multiobjetivo que usa frentes de Pareto para la optimización de múltiples objetivos. Luego de generar la nueva población de soluciones, NSGA-III establece los diferentes niveles de frentes de Pareto y utiliza estos frentes para crear la próxima población. Primero incorpora los diferentes frentes de mejor a peor hasta antes de exceder el tamaño de la población. Si al agregar el próximo frente de Pareto, la cantidad de soluciones excede el tamaño de la población, se agregan solo la cantidad necesarias para completar el tamaño deseado de la población usando técnicas que aseguran la diversidad de las soluciones en la población. En particular, en Deb y Jain (2014) los autores proponen una elección basada en un conjunto de puntos de referencia. Estos puntos de referencia son elegidos de manera tal que se encuentren amplia y uniformemente distribuidos en hiperplanos de los objetivos a optimizar. Luego, las soluciones en el frente de Pareto son comparadas con los puntos de referencia eligiendo las más similares.

4. Resultados y discusión

Los experimentos fueron realizados con 4 workflows usados de referencia en la literatura (Garí et al., 2022): CyberShake, Montage, LIGO's Inspiral y SIPHT. Para cada uno de esos *workflows* se realizaron 100 ejecuciones con una población

de 1000 individuos usando el algoritmo NSGA-III de MOEA. El operador de recombinación usado ocurrió con una probabilidad de 100 % mientras que el operador de mutación ocurrió con una probabilidad de 90 %.

Los experimentos han sido realizados mediante el uso del simulador CloudSimPlus. Los valores de *makespan* y costo que guían el paso de selección de NSGA-III son promedios de *makespan* y costo después 30 ejecuciones de un mismo workflow. Cada una de las 30 ejecuciones usadas en el promedio, son ejecuciones en donde se usan técnicas estocásticas para simular la variabilidad de la duración de cada tarea del workflow.

Usando de referencia Garí et al., 2022 se decidió comparar los resultados de ese trabajo con los obtenidos por NSGA-III. Para ello, se comparó el costo, el *makespan* y la métrica L2 del trabajo anteriormente citado con los presentados a continuación. La métrica L2, consiste en realizar el cálculo de la raíz cuadrada de la suma de los cuadrados de las cantidades de los valores objetivo. Es decir, es análoga a la distancia euclidiana. El uso de está métrica fue justificado en Garí et al., 2022 dado que el *makespan* medido en horas y el costo medido en dólares tenían valores del mismo orden de magnitud.

Para comparar estas dos técnicas se eligió en ambas los agentes con la menor métrica agregada L2. Esto se debió a que una menor métrica agregada se corresponde a menor *makespan* y menor costo. En el caso de los resultados con AE, NSGA-III entrega como resultados las soluciones que pertenecen al primer frente de Pareto. Sobre este primer frente, se elige aquella política que tiene menor L2. En el caso de los agentes de AR, se elige aquel agente que ha reportado el menor valor de L2, luego de un 30 ejecuciones de evaluación. En este sentido la elección de agentes es similar a la realizada para las soluciones evolutivas, buscando aquellas con menor L2.

	makespan [horas]			costo [USD]		
	AR	NSGA-III	ganancia	AR	NSGA-III	ganancia
CyberShake	5.12	5.14	-0.39 %	6.66	**6.09**	8.56 %
Inspiral	28.37	28.33	0.14 %	**36.13**	37.33	-3.32 %
Montage	1.83	1.83	<0.55 %	**2.06**	2.53	-22.82 %
SIPHT	80.22	80.39	-0.21 %	49.15	**33.76**	31.31 %
	métrica agregada L2					
	AR	NSGA-III	ganancia			
CyberShake	8.41	**7.98**	5.11 %			
Inspiral	45.94	46.86	-2.00 %			
Montage	**2.77**	3.14	-13.36 %			
SIPHT	94.08	**87.19**	7.32 %			

Tabla 1. Comparación de resultados entre las dos técnicas analizadas. Se reportan variaciones en *makespan*, costo y métrica L2 entre los dos algoritmos. Se utilizó el estimador U de Mann-Whitney para analizar si las diferencias encontradas son significativas. En negrita se señalan los valores menores cuando estos son significativos.

Una vez identificadas las mejores soluciones de ambos algoritmos se utilizaron las 30 evaluaciones realizadas en cada una de las soluciones. De esta manera, se compararon los 30 valores de *makespan*, costo y métrica L2 de cada algoritmo por medio del test de Mann-Whitney de dos colas con $p = 0{,}01$. Estos resultados se resumen en la tabla 1. Se observa que el *makespan* de CyberShake y SIPHT no mejoran con NSGA-III, mientras que sí lo hace para Montage y LIGO's Inspiral. Sin embargo, estas diferencias son muy pequeñas (menores en valor absoluto a 1 %) y no significativas. Esto quiere decir que ambos autoescaladores producen distribuciones estadísticas de *makespan* muy parecidas, tan parecidas que no se puede descartar que sean la misma distribución. También se puede ver que para el costo esta ganancia se invierte. Es decir, ahora CyberShake y SIPHT mejoran, mientras que Montage y LIGO's Inspiral empeoran. No obstante, las diferencias sí son significativas para los 4 workflows.

Para la métrica agregada, se observa que los aumentos siguen la tendencia observada en el costo: CyberShake y SIPHT mejoran, mientras que Montage y LIGO's Inspiral empeoran. Por otro lado, se observa que en CyberShake, SIPHT y Montage, las diferencias en métrica agregada son significativas, mientras que no lo es para LIGO's Inspiral. La tendencia de la métrica agregada es explicable por dos razones. Por un lado, el costo tiene diferencias significativas mayores al 5 % en todos los casos donde persiste la significatividad. Es decir, estas diferencias son lo suficientemente grandes para influir en la métrica agregada. Por el otro lado, para LIGO's Inspiral, la pérdida en costo del 3.33 % es muy baja. Esto, sumado a la similitud de las distribuciones de *makespan* hace que los valores de la métrica agregada se parezcan demasiado entre ellos, con lo cual el test de significancia da negativo.

5. Perspectivas a futuro

Los resultados aquí presentados parecen prometedores. Se han logrado mejoras con respecto a lo obtenido con otras técnicas. No debe descartarse el uso de AE para problemas similares a futuro o incluso proponer alternativas al modelado del problema.

En este trabajo, la búsqueda de políticas óptimas ha sido por medio de AE. Sin embargo, existe la posibilidad de buscar políticas con ideas similares a las propuestas en Chellapilla y Fogel, 2001. En ese caso, se debe redefinir el problema de decisión markoviano. Estas modificaciones pueden darse tanto en el espacio de estados (por ejemplo usando las variables continuas en lugar de la discretización) como en el espacio de acciones (ya sea agregando nuevas MVs o agregando acciones de planificación)

Otra posible modificación al problema de decisión markoviano podría consistir en incluir variables de impacto ambiental. Shaw et al. (2022) es un ejemplo de un trabajo que incluye el consumo de energía entre los objetivos a minimizar con su función de recompensa. Del mismo modo, la optimización multiobjetivo usada podría redefinirse para incluir el consumo de energía como otra variable a optimizar.

Finalmente, se podrían usar las políticas aprendidas con AEs como valores iniciales para agentes de AR. Se sabe que las condiciones iniciales de AR son un problema abierto (Garí et al., 2021).

Referencias

Chellapilla, K., & Fogel, D. (2001). Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, *5*(4), 422-428. https://doi.org/10.1109/4235.942536

Deb, K., & Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part I: solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, *18*(4), 577-601.

Ding, Y., & Shi, T. (2024). Sustainable LLM Serving: Environmental Implications, Challenges, and Opportunities : Invited Paper. *2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC)*, 37-38. https://doi.org/10.1109/IGSC64514.2024.00016

Ericson, J., Mohammadian, M., & Santana, F. (2017). Analysis of performance variability in public cloud computing. *Proceedings of the 2017 IEEE International Conference on Information Reuse and Integration*, 308-314.

Garí, Y., Monge, D. A., & Mateos, C. (2022). A Q-learning approach for the autoscaling of scientific workflows in the Cloud. *Future Generation Computer Systems*, *127*, 168-180. https://doi.org/https://doi.org/10.1016/j.future.2021.09.007

Garí, Y., Monge, D. A., Mateos, C., & García Garino, C. (2019). Learning budget assignment policies for autoscaling scientific workflows in the cloud. *Cluster Computing*. https://doi.org/10.1007/s10586-018-02902-0

Garí, Y., Monge, D. A., Pacini, E., Mateos, C., & García Garino, C. (2021). Reinforcement learning-based application Autoscaling in the Cloud: A survey. *Engineering Applications of Artificial Intelligence*, *102*, 104288. https://doi.org/https://doi.org/10.1016/j.engappai.2021.104288

Garí, Y., Pacini, E., Robino, L., Mateos, C., & Monge, D. A. (2024). Online RL-based cloud autoscaling for scientific workflows: Evaluation of Q-Learning and SARSA. *Future Generation Computer Systems*, *157*, 573-586. https://doi.org/https://doi.org/10.1016/j.future.2024.04.014

Liu, J., Feld, D., Xue, Y., Garcke, J., Soddemann, T., & Pan, P. (2016). An efficient geosciences workflow on multi-core processors and GPUs: a case study for aerosol optical depth retrieval from MODIS satellite data. *International Journal of Digital Earth*, *9*(8), 748-765.

Mao, M., & Humphrey, M. (2013). Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, 67-78.

Meade, B. F., & Fluke, C. J. (2018). Evaluating virtual hosted desktops for graphics-intensive astronomy. *Astronomy and computing*, *23*, 124-140.

Mell, P., Grance, T., & Grance, T. (2011). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *National Institute of Standards and Technology Special Publication 800-145 7*.

Monge, D. A., Garí, Y., Mateos, C., & García Garino, C. (2017). Autoscaling Scientific Workflows on the Cloud by Combining On-demand and Spot Instances. *International Journal of Computer Systems Science and Engineering*, *32*(4 Special Issue on Elastic Data Management in Cloud Systems).

Monge, D. A., & Garino, C. G. (2014). Adaptive Spot-Instances Aware Autoscaling for Scientific Workflows on the Cloud. *Latin American High Performance Computing Conference*, 13-27.

Monge, D. A., Pacini, E., Mateos, C., Alba, E., & García Garino, C. (2020). CMI: An online multi-objective genetic autoscaler for scientific and engineering workflows in cloud infrastructures with unreliable virtual machines. *Journal of Network and Computer Applications*, *149*, 102464. https://doi.org/https://doi.org/10.1016/j.jnca.2019.102464

Monge, D. A., Pacini, E., Mateos, C., & García Garino, C. (2018). Meta-heuristic based autoscaling of cloud-based parameter sweep experiments with unreliable virtual machines instances. *Computers and Electrical Engineering*, *69*, 364-377. https://doi.org/10.1016/j.compeleceng.2017.12.007

Pu, X., Liu, L., Mei, Y., Sivathanu, S., Koh, Y., & Pu, C. (2010). Understanding performance interference of I/O workload in virtualized cloud environments. *Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010*, 51-58. https://doi.org/10.1109/CLOUD.2010.65

Schad, J., Dittrich, J., & Quiané-Ruiz, J.-A. (2010). Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, *3*(1-2), 460-471.

Shaw, R., Howley, E., & Barret, E. (2022). Applying Reinforcement Learning towards automating energy efficient virtual machine consolidation in cloud data centers. *Information Systems*, *107*, 101722. https://doi.org/https://doi.org/10.1016/j.is.2021.101722

Silva Filho, M. C., Oliveira, R. L., Monteiro, C. C., Inácio, P. R. M., & Freire, M. M. (2017). CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 400-406. https://doi.org/10.23919/INM.2017.7987304

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book.

Vandenbrouck, Y., Christiany, D., Combes, F., Loux, V., & Brun, V. (2019). Bioinformatics tools and workflow to select blood biomarkers for early cancer diagnosis: an application to pancreatic cancer. *Proteomics*, *19*(21-22), 1800489.