

Reto de Hackathon

Qiskit Fall Fest Uniandes Colombia 2025

Universidad de los Andes

Noviembre 2025

Enunciado: Configuración Óptima del Planificador del Clúster HPC “Cóndor”

Contexto

La Universidad de los Andes opera el clúster de *High-Performance Computing* (HPC) “Cóndor”, que da servicio a docenas de grupos de investigación en física, ingeniería, biología computacional, finanzas cuantitativas y ciencia de datos. Para balancear la carga de trabajo y aprovechar al máximo los recursos, el equipo de TI está desplegando una nueva política de configuración para el planificador de trabajos (*scheduler*).

En producción, esta política se controla mediante un archivo de configuración interno con decenas de parámetros binarios. Sin embargo, para efectos de pruebas y simulación se ha diseñado una versión *reducida* de esta configuración que captura las decisiones más importantes del planificador en un vector binario de longitud fija:

$$n = 16.$$

Cada configuración de prueba del planificador se representa como un vector de bits

$$x = (x_0, x_1, \dots, x_{15}) \in \{0, 1\}^{16}.$$

Usaremos índices desde 0 (para x_0) hasta 15 (para x_{15}).

En esta versión reducida:

- Bits x_0 – x_3 : modos globales de CPU (afinidad, SMT, modo seguro, modo de protección y verificación).
- Bits x_4 – x_7 : activación y partición de nodos GPU (modo básico vs. nodos de alta gama).
- Bits x_8 – x_{10} : parámetros del sistema de archivos paralelo y cachés distribuidas.
- Bits x_{11} – x_{13} : política principal de planificación de colas (modelo reducido de políticas).

- Bits x_{14} – x_{15} : parámetros de red de alta velocidad (configuración de enlaces Infiniband) y tolerancia a fallos.

Tras una migración de software, la documentación detallada de las interacciones entre estos bits se ha perdido. Lo único que queda es una rutina interna de diagnóstico, desarrollada por los administradores del sistema, que determina si una configuración es válida y estable. Esta rutina es una “caja negra”: no se conoce su lógica interna, pero ha sido compilada en forma de un *oráculo cuántico* que se les proporcionará.

La rutina de diagnóstico es exhaustiva: simula el lanzamiento de cientos de trabajos, comprueba la coherencia de la asignación de recursos (CPU, GPU y memoria), verifica que no haya conflictos de red, valida las políticas de equidad (*fair-share*) entre departamentos y descarta configuraciones que generen inestabilidades o condiciones de carrera bajo alta concurrencia.

Información parcial recuperada

Aunque la documentación oficial se perdió, el equipo de TI logró recuperar algunos fragmentos de información a partir de bitácoras (*logs*) y copias de seguridad antiguas de configuraciones que funcionaron en el pasado. Estos datos no describen completamente la configuración válida, pero sí introducen pistas y restricciones parciales sobre el vector

$$x = (x_0, x_1, \dots, x_{15}).$$

En particular, se sabe lo siguiente sobre tres posiciones específicas y algunas restricciones globales:

- **Bit x_3 : modo de protección del scheduler.** En todos los registros históricos en los que el clúster operó de forma estable bajo carga mixta (trabajos cortos e interactivos mezclados con trabajos largos en lote), el bit x_3 estaba activado. Se concluye que cualquier configuración válida debe satisfacer

$$x_3 = 1.$$

- **Bit x_8 : sistema de archivos paralelo.** El sistema de archivos paralelo de alta capacidad y las cachés distribuidas están asociados al bit x_8 . Los registros muestran que, en todas las configuraciones estables utilizadas en producción, el sistema de archivos paralelo se encontraba siempre activo. Por tanto, para cualquier configuración válida se tiene

$$x_8 = 1.$$

- **Bit x_{11} : política de planificación preferida.** Los bits x_{11} , x_{12} y x_{13} codifican una familia reducida de políticas globales de planificación (por ejemplo, variantes de FIFO, Fair-Share, Backfill). Los administradores han observado que, en todos los periodos en los que el clúster se comportó de manera correcta y estable bajo distintas cargas,

la política seleccionada correspondía al bit x_{11} activado. En consecuencia, se sospecha que toda configuración realmente válida debe cumplir

$$x_{11} = 1.$$

- **Límite de activación de nodos de cómputo.** Se sabe que existe un límite de presupuesto de energía y de capacidad de refrigeración. Sea $\mathcal{I}_{\text{nodos}}$ el conjunto de índices que representan activación de nodos de cómputo (en esta versión reducida, por ejemplo $\mathcal{I}_{\text{nodos}} = \{1, 2, 4, 5, 6, 7\}$). Entonces, la rutina de diagnóstico rechaza una configuración si el número de bits activos en ese conjunto supera cierto umbral k :

$$\sum_{i \in \mathcal{I}_{\text{nodos}}} x_i \leq k,$$

con k un entero fijo (por ejemplo $k = 4$ en las instancias de prueba del reto).

- **Condición de paridad sobre parámetros de red.** Algunos subconjuntos de bits de red e Infiniband deben satisfacer una condición de paridad para que el hardware los acepte. Por ejemplo, para el subconjunto $\mathcal{I}_{\text{red}} = \{14, 15\}$ se exige que la suma tenga paridad *impar*:

$$x_{14} + x_{15} \equiv 1 \pmod{2},$$

es decir, exactamente uno de los dos bits está a 1 (y el otro a 0).

- **Estabilidad bajo cargas de *big data* y GPU.** Las configuraciones que activan la cola de *big data* ($x_{12} = 1$) al mismo tiempo que un número máximo de nodos GPU de alta gama (por ejemplo cuando $x_6 = 1$ y $x_7 = 1$) tienden a generar inestabilidades bajo carga extrema. La rutina lanza trabajos de prueba concurrentes para detectar condiciones de carrera y solo acepta configuraciones que gestionan la concurrencia de forma estable.

En resumen, se sabe *explícitamente* que en toda configuración válida los bits x_3 , x_8 y x_{11} deben estar en 1, mientras que el resto de bits están sujetos a restricciones lógicas globales (capacidad, paridad, estabilidad), pero no se conocen de antemano sus valores exactos.

Problema

Escriba un circuito cuántico que descubra al menos una configuración válida (una cadena de 16 bits) para el planificador del clúster “Cóndor”, utilizando exclusivamente el oráculo cuántico proporcionado.

Formalmente:

- Existe una función desconocida

$$f : \{0, 1\}^{16} \longrightarrow \{0, 1\},$$

que devuelve 1 si la configuración x es válida y estable según la rutina de diagnóstico interna, y 0 en caso contrario.

- Se les proporcionará un módulo de Python llamado `condor_oracle.py`. Dicho módulo contiene una función que construye un circuito de Qiskit que implementa un oráculo cuántico de consulta

$$U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle,$$

donde los primeros 16 qubits representan la configuración x y el último qubit es un ancilla $|y\rangle$.

- Ustedes *no* tienen acceso a una implementación clásica de $f(x)$ ni a la lógica interna que lo define; sólo pueden llamar al circuito U_f como subrutina cuántica dentro de los circuitos que diseñen para resolver el reto.

Obsérvese que, en un enfoque clásico de fuerza bruta, un computador convencional debería considerar, en principio, todas las posibles configuraciones de 16 bits, es decir

$$2^{16} = 65\,536$$

cadena distintas. Sin embargo, gracias a la información parcial recuperada (los bits x_3 , x_8 y x_{11} están fijados en 1 para cualquier configuración válida), el número efectivo de combinaciones que queda por explorar se reduce a

$$2^{16-3} = 2^{13} = 8192$$

configuraciones posibles. Aunque esta cifra es mucho menor que 65 536, sigue siendo un espacio de búsqueda considerable si cada intento implicara simular detalladamente el comportamiento del clúster. El reto consiste en aprovechar un algoritmo cuántico para reducir de forma significativa el número de aplicaciones necesarias del oráculo U_f .

- El número de bits $n = 16$ sigue siendo lo bastante grande como para que una búsqueda exhaustiva clásica (probar las 2^{13} configuraciones efectivamente posibles, más todas las restricciones internas adicionales) resulte costosa si se quisiera reproducir el diagnóstico completo.

Su tarea es diseñar un algoritmo cuántico que encuentre al menos una configuración válida, si existe, *minimizando* el número de aplicaciones del oráculo (es decir, el número de veces que su circuito utiliza U_f).

Su enfoque será evaluado con base en:

1. **Correctitud:** La configuración devuelta debe ser marcada como válida por la función f , es decir, debe corresponder a un estado $|x^*\rangle$ sobre el que el oráculo de fase actúa con un signo negativo.
2. **Complejidad en llamadas al oráculo:** Se valorará el número de veces que se invoca U_f dentro de su circuito. Se espera que su solución aproveche una ventaja cuántica sobre una búsqueda aleatoria o una fuerza bruta clásica.
3. **Claridad y justificación:** Debe explicar con claridad su estrategia de búsqueda, cómo mapea las restricciones lógicas del problema al oráculo cuántico y cómo decide el número de iteraciones/cuotas de consulta al oráculo en función del tamaño efectivo del espacio de búsqueda.

Como comprobación adicional, recuerde que los valores de los bits x_3 , x_8 y x_{11} han sido *recuperados* de los logs y son conocidos de antemano por el equipo de TI. Por tanto, la cadena de bits que su circuito produce con mayor probabilidad tras ejecutar el algoritmo debe coincidir en esas tres posiciones con los valores descritos en la sección de información parcial recuperada. Si la configuración más probable que obtienen al medir no respeta estos tres bits, probablemente haya un error en la implementación de su oráculo efectivo de fase o del difusor.

Recuerde que el oráculo cuántico sólo marcará como soluciones las configuraciones que satisfacen *todas* las condiciones (restricciones de recursos, dependencias de software, políticas de colas, límites de energía, paridad de la red y estabilidad bajo concurrencia).

Acceso al oráculo en el notebook

En el repositorio del reto se les proporcionará el módulo de Python `condor_oracle.py`. Este módulo contiene una función

```
Uf_condor = build.Uf_condor()
```

que devuelve un circuito de Qiskit sobre 17 qubits, que implementa un oráculo de consulta

$$U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle,$$

donde los primeros 16 qubits codifican la configuración x y el último qubit es un ancilla.

En sus notebooks, el módulo puede utilizarse de forma directa, por ejemplo:

```
from condor_oracle import build_Uf_condor, N_CONFIG_BITS
from qiskit import QuantumCircuit

Uf_condor = build_Uf_condor()          # circuito del oráculo
qc = QuantumCircuit(N_CONFIG_BITS + 1) # 16 bits de config + ancilla
qc.compose(Uf_condor, inplace=True)    # aplicar Uf dentro de su circuito
```

- Si se prepara un estado clásico $|x\rangle|0\rangle$ y se aplica U_f , entonces el ancilla queda en el estado $|f(x)\rangle$. Midiendo ese qubit se obtiene directamente el valor $f(x) \in \{0,1\}$ asociado a la configuración x .
- El mismo circuito puede aprovecharse, si así lo desean, para construir operadores que introduzcan fases dependientes de $f(x)$ sobre el registro de datos (por ejemplo, transformaciones diagonales del tipo Z_f). La manera concreta de lograrlo forma parte del diseño que cada equipo debe proponer.

El archivo `condor_oracle.py` debe tratarse como una *caja negra*: no es necesario abrirlo ni modificarlo para resolver el reto, y se pide explícitamente no inspeccionar su contenido durante el hackathon. En particular, se solicita no aplicar métodos como `.decompose()` ni transformaciones análogas sobre el circuito `Uf_condor` con el fin de revelar su estructura interna; a efectos del reto, el oráculo debe considerarse un componente opaco. Su trabajo consiste en utilizar el circuito U_f (y cualquier construcción adicional que diseñen a partir

de él) como bloque básico dentro de sus propios circuitos para localizar una configuración válida del planificador.

Formato de entrega

Cada equipo (de máximo tres personas) debe entregar dos componentes:

1. Notebook de trabajo (.ipynb).

- Debe contener todo el código necesario para reproducir su solución: importación del módulo `condor_oracle.py`, construcción de los circuitos, ejecución en simulador y obtención de una configuración candidata.
- El notebook debe correr de principio a fin sin errores en un entorno estándar con Qiskit instalado, generando explícitamente como salida la(s) cadena(s) de bits propuesta(s) como configuración válida.
- Incluya celdas de texto (Markdown) que expliquen, con sus propias palabras, la idea general del algoritmo, cómo incorporan el oráculo U_f dentro del circuito y cómo verifican empíricamente que su solución es correcta.

2. Presentación técnica.

- Deben preparar una presentación (diapositivas) donde describan con detalle el diseño de su solución: decisiones sobre la estructura del circuito, elección de registros auxiliares, forma de explotar el oráculo y estrategia de búsqueda en el espacio de configuraciones.
- La presentación debe incluir un *análisis del circuito* en términos de complejidad:
 - número total de qubits utilizados (registro de datos + ancillas),
 - profundidad aproximada del circuito y tamaño (número de compuertas),
 - número total de aplicaciones del oráculo U_f y cómo escala esa cantidad al aumentar el número de bits del problema,
 - posibles mejoras o variantes que consideraron (aunque no las hayan implementado completamente).
- Durante la (posible) exposición se evaluará tanto la corrección técnica como la claridad de la explicación y la justificación de las decisiones de diseño.