Evaluación de la disponibilidad de los servicios desplegados sobre *Volunteer Computing*

Antonio Escot Praena

Enginyeria Informàtica i Tècnica de Gestió

Dirección del TFC

Ángel A. Juan, PhD.

Eva Vallada Regalado, PhD.

Alberto García Villoria, PhD.

La computación distribuida

Ofrece la posibilidad de poner en marcha proyectos de computación de alto rendimiento de forma escalable y económica, debido principalmente al crecimiento de la potencia de cálculo de los ordenadores y a las mejoras de ancho de banda en las redes de comunicaciones.

- Grid computing. Computación distribuida empresarial.
- Cloud computing. Servicios en la nube, cuyo coste económico depende de los recursos contratados.
- Volunteer computing. Computación sobre recursos excedentes ofrecidos por usuarios de forma voluntaria.

El *Volunteer Computing*

Las redes de voluntarios comparten recursos geográficamente distribuidos, heterogéneos y en general débilmente acoplados, apropiados por diferentes individuos u organizaciones con sus propias políticas de acceso y una gran variabilidad en sus condiciones de disponibilidad y carga.

- Ofrecen una oportunidad a la computación de cálculo intensivo de forma muy barata
- Fácilmente escalable, la potencia de cálculo ofrecida crece con el número de voluntarios participantes
- Diversas políticas de acceso y versiones de sistemas operativos
- Pueden no estar disponibles durante largos periodos de tiempo por fallo o apagados decididos por sus propietarios

Planteamiento del problema

Servicio. Cualquier aplicación ejecutándose en uno o varios ordenadores capaz de recibir y responder a mensajes procedentes de otros ordenadores.

- Formalmente, un servicio queda representado por un grafo acíclico G=(V,A), donde nodos son recursos y aristas son tareas.
- Se busca un conjunto de m nodos de entre N nodos disponibles que permita desplegar un servicio que debe estar operativo durante el tiempo necesario para llevar a cabo las tareas que lo definen con una probabilidad fijada de antemano y que minimiza la función de coste:

$$C = \sum_{i=1}^{N} \delta_i c(v_i), \text{ con } \delta_i = \begin{cases} 1, & \textit{nodo utilizado} \\ 0, & \textit{nodo no utilizado} \end{cases} \textit{ y } c(v_i) \textit{ es el coste del componente } v_i$$

- Es un problema de optimización combinatoria del tipo NP-completo. Para resolverlo se propone un algoritmo híbrido basado en una heurística de búsqueda local y la simulación por eventos discretos.

Exploración de soluciones vecinas /1

- Un problema de optimización está formado por la tripleta (S, Ω, f)
  - S es el espacio de búsqueda, definido sobre un conjunto de variables $\{X_i, i=1,\ldots,m\}$, discretas o continuas dependiendo del tipo de problema a optimizar.
  - Ω es el conjunto de condiciones que deben cumplir las variables X_i
  - f es la función objetivo a optimizar, $f : S \rightarrow R$, de forma que si se buscamos minimizar f, se cumplirá que $f(s) \leq f(s'), \forall\ s' \in S$.
- No siempre disponemos de algoritmos que encuentren óptimos globales
- Muchos problemas NP-completos sólo pueden ser resueltos usando métodos metaheurísticos.

Exploración de soluciones vecinas /2

Una metaheurística es un algoritmo de carácter iterativo que guía una heurística base combinando diferentes conceptos de forma inteligente, para explorar (diversificar) y explotar (intensificar) el espacio de soluciones del problema.

- **Vecindad.** Dada una solución *s* del espacio de búsqueda llamaremos vecindad, V(s), al conjunto de soluciones a las que se puede llegar mediante una regla de transformación (movimiento) a partir de ella.
  - Su estructura tiene un gran impacto en el rendimiento del algoritmo
  - Los vecinos son generados en las metaheurísticas mediante funciones de transformación o movimiento.

Exploración de soluciones vecinas /3

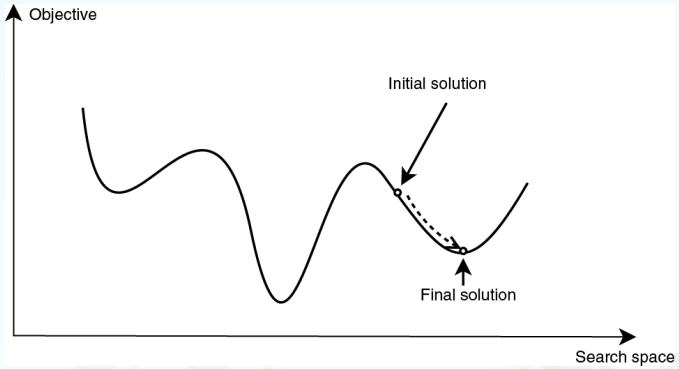
Clasificación de las heurísticas

- **Basadas en trayectorias.** Parten de una solución inicial y tratan de mejorarla en cada iteración de forma tal que parecen trazar una trayectoria en el espacio de búsqueda.
  - Más adecuadas para estructuras discretas de la vecindad
  - Más orientadas a intensificación que a la diversificación
  - Búsqueda Local, Búsqueda Tabú y Recocido Simulado son ejemplos
- **Basadas en poblaciones.** Van generando poblaciones de soluciones en cada iteración, la población de partida para la siguiente iteración se integra mediante un mecanismo de selección.
  - Más adecuadas para problemas continuos
  - Orientadas a la diversificación más que a la intensificación
  - Búsqueda Dispersa, Colonia de Abejas y los Algoritmos Genéticos son ejemplos.

La heurística *Local Search*

Parte de una solución inicial y en cada iteración busca una solución entre sus vecinas que mejore la solución actual. Traza un recorrido en el espacio de búsqueda que nos lleva de una solución inicial s_0 a un óptimo local s^*.

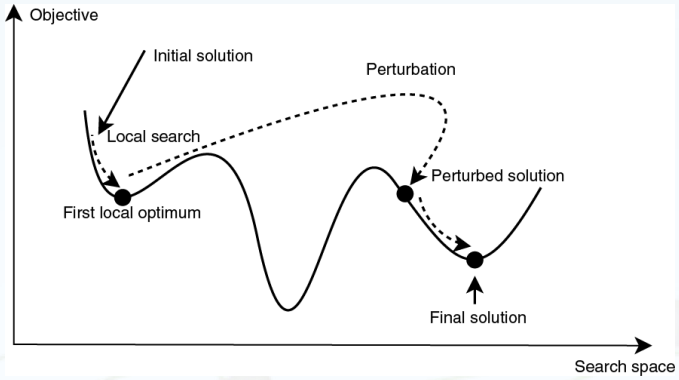
- Su eficiencia depende de la elección de la solución inicial y del método de exploración del vecindario. El conocimiento, a priori, de la estructura de la vecindad no siempre es posible.
- Establece un mapeo varias a una entre el conjunto de soluciones S y el de óptimos locales S*.
- Permiten encontrar soluciones rápidamente
- Es un método determinista
- Tiende a quedar atrapada en óptimos locales –pozos de atracción- que pueden ser de mala calidad.



La metaheurística *Iterated Local Search /1*

El *Iterated Local Search –ILS-* es una algoritmo basado en una heurística *Local Search –LS-* que sigue la estrategia de iterar el proceso partiendo de diferentes soluciones iniciales para escapar de los pozos de atracción.

- Aprovecha el mapeo entre S y S* establecido por la búsqueda local embebida para seleccionar un vecino para empezar en la siguiente iteración.
- El conjunto de óptimos locales S* es mucho menor que S. Un reinicio aleatorio aplicado en S* permite escapar de los pozos.
- Un mecanismo mejor de exploración de S*
  - Generar una solución inicial
  - Aplicar LS para obtener el óptimo local
  - Perturbar la solución obtenida
  - Iterar el proceso partiendo de esta solución perturbada



La metaheurística *Iterated Local Search /2*

```
Procedure ILS
    S0 = GenerateInitialSolution();
    s* = LocalSearch(s0);
    Repeat
        s' = Perturbation(s*, historia);
        s*' = LocalSearch(s');
        if AcceptanceCriterion(s*', s*, memoria) then
            s* = s*';
        endif
    Until EndCondition
End
```

La metaheurística *Iterated Local Search /3*

- **GenerateInitialSolution.** Procedimiento que genera la solución de partida.
- **LocalSearch.** Heurística base del ILS que explora el espacio de búsqueda en la vecindad de la solución candidata (intensificación), puede ser determinista o estocástica.
- **Perturbation.** Mecanismo de transformación de la solución actual, que puede hacer uso de la historia o no, que permite diversificar el espacio de búsqueda explorado.
  - Una buena función de perturbación transforma una buena solución en un buen punto de inicio para la LS.
- **AcceptanceCriterion.** Procedimiento o condición que determina si la solución candidata propuesta por la LS es aceptable o no.
- **EndCondition.** Condición de finalización del proceso iterativo, en general, depende de las variables sujetas a optimización y del tiempo asignado.

Evaluación de la disponibilidad

- Basada en la simulación por eventos discretos –SED- requiere de:
  - Generador de números pseudoaleatorios.
  - Funciones inversas de las distribuciones de probabilidad que modelan el problema.
- Dos implementaciones similares basadas en SED
  - El SAEDES
    - Utiliza el concepto de adyacencia , las trayectorias críticas y un algoritmo *Breadth-first search* para comprobar la disponibilidad
    - Disponibilidad es igual a 1 si el sistema está disponible durante Tvida, igual a 0 si no
    - Buen rendimiento con topologías complejas
  - EL DiseASim
    - Utiliza el concepto de recorrido de un grafo para verificar la disponibilidad del sistema
    - Disponibilidad = Tdisp/Tvida , Tdisp es el tiempo acumulado durante Tvida en el que el sistema ha estado disponible
    - Mejor rendimiento con topologías con pocas redundancias

Implementación del DS-ReliaSim

El DS-ReliaSim trata con redes de gran escala, topologías complejas y modelos estadísticos de fallo/reparación para los diferentes recursos, para obtener una solución de mínimo coste que permite desplegar un servicio sobre ellas.

- Es una metaheurística del tipo ILS que combina una heurística de búsqueda local y la simulación por eventos discretos.
- Es un algoritmo con una estructura modular.
- Admite diferentes formas de optimización aplicadas sobre cada uno de los módulos o incluyendo información específica del problema al que se aplica.
- Es un algoritmo fácil de implementar que nos devuelve soluciones de gran calidad (cercanas a óptimos globales)

DS-ReliaSim: Pseudocódigo del algoritmo

```
Procedure DS-ReliaSim
nodes = sortByAvailability(nodes)
currentSol = buildDummySol(nodes; topology)
sysAvailability = DiSeASim(currentSol; topology; time; iter)
currentNodes = extractNodes(currentSol)
nodes = extractNodes(currentNodes; nodes)
currentNodes = sortByCost(currentSol)
auxSysAvail = sysAvailability
while nodes is not empty and auxSysAvail > threshold do
        inNode = selectNextNode(nodes)
        outNode = selectNextNode(currentNodes)
        while outNode exists and cost(inNode) < cost(outNode) do
                newSol = buildShiftedSol(currentSol; inNode; outNode)
                auxSysAvail = DiSeASim(newSol; topology; time; iter)
                if auxSysAvail > threshold then
                        currentSol = newSol
                        break
                end if
                outNode = selectNextNode(currentNodes)
        end while
end while
return currentSol
```

Solución inicial
Simulador SED
Condición de finalización
Perturbación
Criterio de Aceptabilidad
Búsqueda local

DS-ReliaSim: Elementos destacados /1

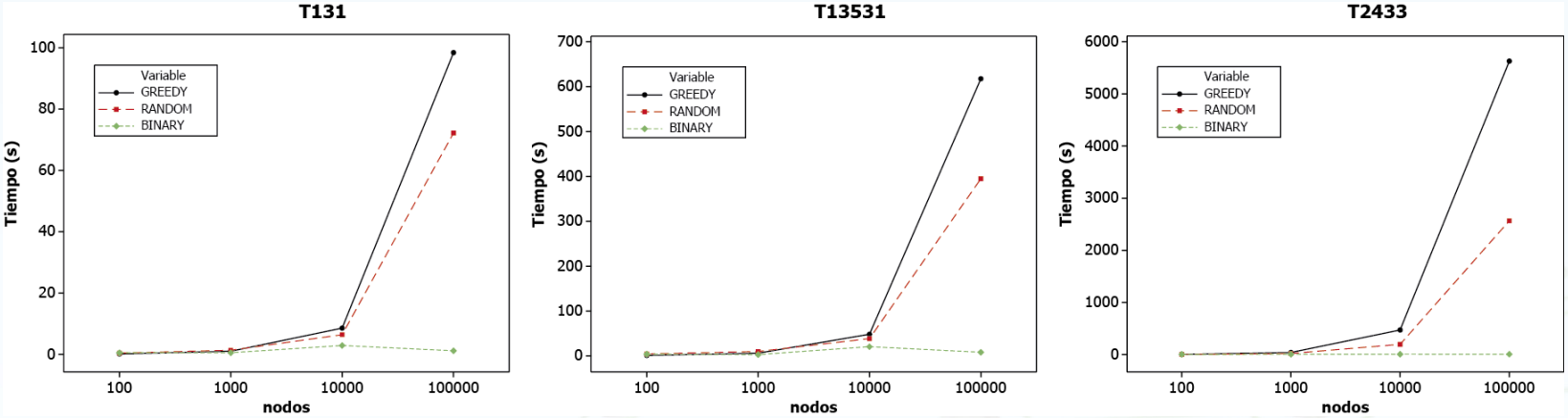
- **Descripción del servicio**. Los usuarios detallan el servicio que quieren desplegar, así como sus requisitos de disponibilidad, su topología y las condiciones de finalización.
- **Solución inicial**. El DS-ReliaSim empieza ordenando la lista de recursos disponibles por disponibilidad y genera una solución inicial.
  - GREEDY. Versión por defecto, construye la solución de inicio escogiendo los primeros nodos de la lista, los de mayor disponibilidad y mayor coste.
  - RANDOM. Reinicio aleatorio hasta encontrar una solución que cumple el criterio de aceptabilidad (auxSysAvail > threshold).
  - BINARY. Utiliza una búsqueda binaria sobre la lista ordenada para encontrar una solución aceptable de partida cercana a la solución óptima.

DS-ReliaSim: Elementos destacados /2

- **Función de perturbación.**
  - DEFAULT. Se sustituye el nodo más caro de la solución actual por el siguiente nodo en la lista de nodos disponibles (más barato), la solución construida se aceptará como solución actual si pasa el test de aceptabilidad.
  - FIXED. Aplicar un salto mayor en la selección del siguiente nodo más barato de la lista de nodos disponibles, fijo y conocido a priori, en nuestro caso se ha tomado una longitud del salto igual al número de nodos requeridos por el servicio.
  - ADAPTIVE. Aplicar un salto variable y adaptativo en función del número de nodos disponibles, el número de nodos requeridos por la topología y la diferencia entre la disponibilidad de la solución actual y el umbral especificado en el servicio.
- **Criterio de aceptabilidad.** El algoritmo sólo acepta soluciones que tienen una disponibilidad por encima del umbral (**auxSysAvail > threshold).**
- **Condición de finalización.** El proceso acaba cuando la disponibilidad de la solución actual no alcanza el umbral o bien no quedan más nodos disponibles.

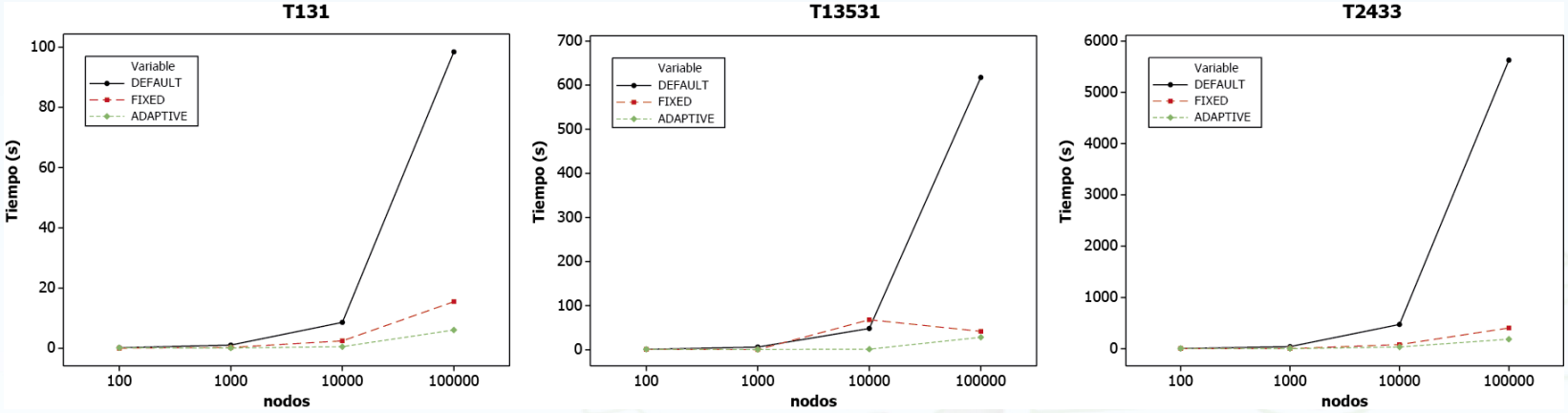
DS-ReliaSim: Experimentos comparativos /1

- **Estrategias de selección de la solución inicial.** Se compara el impacto de la solución de inicio en el rendimiento del algoritmo para diferentes topologías.
  - Hasta 1000 nodos la estrategia greedy es más eficiente que el reinicio aleatorio, por encima de 1000 nodos la opción aleatoria resulta más eficiente.
  - La opción binaria es la más eficiente excepto para muy pocos nodos disponibles.
  - Resultados equivalentes se han obtenido con otras topologías diferentes.



DS-ReliaSim: Experimentos comparativos /2

- **Diferentes funciones de perturbación.** Prueba la velocidad del algoritmo aplicando diferentes funciones de perturbación.
  - Hasta 1000 nodos disponibles las diferencias son inapreciables.
  - Por encima de 1000 nodos saltos mayores que una unidad en la selección del nodo de la lista mejora los tiempos de proceso del algoritmo.
  - Los resultados entre la opción FIXED y ADAPTIVE se igualan para topologías complejas –muchas redundancias-.



DS-ReliaSim: Ejemplo de aplicación

- **3D Rendering Service para bioinformáticos**
  - La bioinformática y la medicina requieren de sistemas de alta computación para analizar, simular o modelar las interacciones entre proteínas, la expresión génica, nuevos fármacos, etc.
  - Son comunidades de usuarios no necesariamente muy grandes.
  - Requieren que el renderizador , con la topología definida en la descripción cargada en el sistema, esté operativo por el tiempo necesario para completar el proceso con una probabilidad fijada al 95%.

DS-ReliaSim: Conclusiones

- El DS-ReliaSim implementa una metaheurística conceptualmente simple, sencilla y de propósito general.
- Aporta soluciones de alta calidad que permiten desplegar un servicio a coste eficiente con una probabilidad determinada de antemano.
- Baja eficiencia cuando se cuenta con muchos nodos disponibles, requiere alguna estrategia de mejora (con posible pérdida de calidad de la solución):
  - Partir de una solución inicial aceptable más barata, aplicando algoritmos aleatorios o de búsqueda binaria.
  - Aplicar saltos más grandes en la selección del siguiente nodo disponible más barato
    - usando funciones de perturbación de longitud de salto superior a 1
    - o funciones adaptativas que calculen el salto teniendo en cuenta, el número de recursos y la diferencia de disponibilidad entre la solución actual y el umbral requerido.
- El algoritmo admite funciones de coste de los recursos más realistas

DS-ReliaSim: Trabajos futuros

- Definir modelos de estado de los recursos compartidos basados en informes empíricos que tenga en cuenta todos los estados posibles de los mismos y no sólo fallo/reparación.
- Desarrollar funciones de cálculo de coste de los recursos más realistas que tengan en cuenta el valor del recurso, su mantenimiento, el ancho de banda de su conexión, su situación geográfica, etc.
- Encontrar la topología mínima de un servicio que permita ser desplegado en una determinada lista de nodos.

Gracias por su atención.

Evaluación de la disponibilidad de los servicios desplegados sobre *Volunteer Computing*

27 de diciembre de 2012