Inteligencia Artificial

Informe Final: Problema [Nombre Problema]

Ignacio Ureta

21 de septiembre de 2015

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

En el presente documento se describirá el problema de reasignación de máquinas, presentado por google en desafío ROADEF/EURO 2011-2012, el cuál consiste en asignar una serie de procesos a máquinas con la idea de hacer mejor uso de las estas. También se describirá el estado del arte del problema para finalmente definir un modelo matemático para utilizar en la futura implementación de un método de resolución que se comparará con lo encontrado en el estado del arte.

1. Introducción

En el presente trabajo se presentará una descripción del estado del arte del problema de reasignación de máquinas de google, que consiste en optimizar la reasignación de procesos dentro de un conjunto de máquinas de acuerdo a ciertos recursos que éstas poseen, se presentará además un modelo en base a este estudio para posteriormente ser utilizado en la resolución del problema, comparándolo con el estado del arte. Se comenzará con la sección 1, donde se describirá brevemente el problema, luego se continuará en la sección 2 donde se presentará el estado del arte para finalmente presentar el modelo a utilizar a futuro

2. Definición del Problema

El problema de reasignación de máquinas, presentado por google en el ROADEF/EURO challenge 2011-2012, consiste en un conjunto de máquinas M y procesos P. La idea es asignar a cada proceso una máquina respetando las siguientes restricciones:

2.1. Restricciones

2.1.1. Capacidad

Cada máquina tiene una cierta cantidad de cada recurso y cada proceso necesita cierta cantidad de recursos, la idea es asignar a cada proceso una máquina de tal manera que la suma de recursos requeridos por los procesos de una máquina no superen los recursos disponibles por ésta

2.1.2. Servicios

Un servicio esta compuesto por un conjunto de procesos (el conjunto de servicios particiona el conjunto de procesos). Dos procesos de un mismo servicio no pueden correr en una misma máquina

2.1.3. Distribución

Cada máquina tiene una localización. Los procesos de un servicio deben estar distribuidos entre al menos un mínimo de spreadMin ubicaciones

2.1.4. Dependencia

Un servicio puede depender de otro servicio. Una maquina pertenece a una vecindad de máquinas. En caso que un servicio S_a dependa de un servicio S_b, todos los procesos de S_a deben correr en máquinas que sean vecinas a aquellas en que corre al menos algún proceso de S_b

2.1.5. Uso transitorio

Al transferido un proceso a una nueva máquina, no se pueden liberar los recursos que éste estaba utilizando en la máquina inicial hasta no haber acabado el proceso, y ya que no hay una variable tiempo en el problema, se asume que todos los cambios son realizados en el mismo momento, por lo que un proceso reasignado utilizará recursos tanto en la máquina donde estaba como en la que va a estar.

2.2. Objetivos

Los objetivos consisten en mejorar la utilización del conjunto de máquinas, minimizando una función de costos compuesta por los distintos tipos de costos del problema, cada tipo de costos con sus propios pesos

2.2.1. Costo de Carga

Existe una umbral de seguridad para cada recurso en cada máquina, pasada ésta cantidad, se incurre en un costo que es igual a la cantidad de ese recurso que se utiliza por sobre el umbral de seguridad

2.2.2. Costo de Balance

La idea es balancear los recursos utilizados en cada máquina, de manera que no queden máquinas sin cierto recurso (o con muy poco), ya que el tener determinado recurso y no de otro es inútil en términos de futuras asignaciones. El costo de tener una máquina desbalanceada será igual a la diferencia de las capacidades restantes de cada recurso (las diferencias son calculadas por pares de recursos, y a cada par se le asigna un cierto peso)

2.2.3. Costo de movimiento de proceso

Existen algunos procesos que son particularmente complicados de cambiar de máquina. El costo de movimiento de proceso viene a representar ese costo, el cual es asignado por proceso.

2.2.4. Costo de movimiento de servicio

Es el número máximo de procesos movidos por servicio entre todos los servicios

2.2.5. Costo de movimiento de máquina

Cada proceso al ser reasignado a una nueva máquina tiene un cierto costo, la suma de todos los costos de los procesos reasignados compone éste tipo de costo.

Finalmente, la función de costos a minimizar queda:

$Min\ costoTotal = \sum_{m \in M} peso_{costoCarga} \cdot \sum_{r \in R} max(0, U_{mr} - SC_{mr}) + \sum_{b \in B} peso_{costoBalance} \cdot \sum_{m \in M} max(0, peso \cdot (D_{mr_1} - D_{mr_1})) + peso_{costoMoverProceso} \cdot \sum_{X_p \neq X_{0p}} PMC(p) + peso_{costoMoverServicio} \cdot max(p \in S_i, X_p \neq X_{0p}) + peso_{costoMoverMaquina} \cdot \sum_{p \in P} MMC(p)$

Es importante recalcar que a pesar del gran espacio de búsqueda del problema y la complejidad de las restricciones que debe cumplir, se debe encontrar una solución en tiempos muy limitados (en el desafío los concursantes disponían de tan solo 5 minutos), lo que hace más complejo el problema e impide que ciertos tipos de técnicas sean utilizadas

3. Estado del Arte

El problema de reasignación de máquinas fue presentado en el desafío ROADEF/EURO 2011-2012 como uno de los desafíos a resolver (el cual fue descrito en la sección anterior). El espacio de búsqueda del problema es demasiado extenso, razón por la cual las técnicas completas no son una buena aproximación para este problema. Es por esto que se utilizan heurísticas para resolver el problema. Algunas de las heurísticas utilizadas son las de programación entera (solo para instancias pequeñas), búsqueda local aleatoria (para instancias pequeñas), Satisfacción de restricciones con búsqueda en una gran vecindad (CSP with LNS), recocido simulado y búsqueda local híbrida, obteniendo los mejores resultados este último, muy cercanos a los óptimos (además del primer lugar en el desafío).

El algoritmo con los mejores resultados (búsqueda local híbrida) funciona de la siguiente manera:

Se inicia con una solución válida. Se ordenan todos los procesos en una lista ordenada de acuerdo con los requerimientos absolutos de cada proceso. Se asigna en número de procesos a considerar en cero (N = 0). Luego para cada iteración se aumenta el número de procesos a considerar y se intenta mejorar la solución reasignando solo los procesos de la lista que vayan de la posición uno a la N, lo que se repite hasta que todos los procesos son considerados para su reasignación.

Para la generación de vecindades, la heurística utiliza cuatro tipos de movimientos: Shift, mover un proceso; Swap, intercambiar dos procesos, cada uno en una máquina diferente. Los dos anteriores son los tipos de movimientos más comunes en la literatura. Además se utilizan los movimientos de BPR (big process reassignement), en el que un proceso que requiere grandes cantidades de recursos se cambia a otra máquina, moviendo varios procesos más pequeños y Chain, que consiste en hacer una cadena de intercambios de procesos, donde el proceso uno toma la posición del proceso dos, el proceso dos del proceso tres, y así hasta el proceso enésimo, el cuál toma la posición del proceso uno.

Las vecindades exploradas por los anteriores movimientos se revisan en el siguiente orden: BPR, shift, swap y chain. Cada uno explora hasta que cumple un criterio de parada, ya sea tiempo de ejecución o bien un mínimo de mejora en la solución. Luego de terminar, le da paso al siguiente tipo de movimiento.

Para aumentar la diversificación del método y así escapar de óptimos locales, se utiliza un método llamado "shake", el cual consiste en elegir un recurso y cambiar su costo de carga, con lo que se puede escapar de un óptimo local.

Es importante recalcar que se determinó experimentalmente en "An algorithmic Study of the Machine Reassignement Problem" que Cambiar todos los procesos de unas cuantas máquinas tiene mejores resultados que cambiar unos pocos procesos de cada una de todas las máquinas (sin embargo los resultados dependen mucho de los criterios de selección de estas máquinas) y que la búsqueda a través del espacio de los infactibles no es muy efectiva ya que volver de ellos al espacio factible es muy difícil (y que una penalización en la función objetivo no es suficiente para cumplir con ese cometido).

4. Modelo Matemático

4.1. Función de evaluación

$Min\ costoTotal = \sum_{m \in M} peso_{costoCarga} \cdot \sum_{r \in R} max(0, U_{mr} - SC_{mr}) + \sum_{b \in B} peso_{costoBalance} \cdot \sum_{m \in M} max(0, peso \cdot (D_{mr_1} - D_{mr_1})) + peso_{costoMoverProceso} \cdot \sum_{X_p \neq X_{0p}} PMC(p) + peso_{costoMoverServicio} \cdot max(p \in S_i, X_p \neq X_{0p}) + peso_{costoMoverMaquina} \cdot \sum_{p \in P} MMC(p)$

4.2. Variables

X_p: Máquina en la que corre el proceso P
U_{mr}: Cantidad del recurso R utilizado por la máquina M (variable dependiente)

4.3. Constantes

X_{0p}: Máquina inicial del proceso P
C_p: Booleano, 1 si el proceso p ha sido reasignado, 0 si no
C_{mr}: Capacidad en la máquina M del recurso R
SC_{mr}: Capacidad de seguridad de la máquina M del recurso R
R_{pr}: Requerimiento del recurso R para el proceso P
S_p: Servicio al que pertenece el proceso P
L_m: Localización a la que pertenece la máquina M
$V_{M_j M_k}$: Booleano, 1 si la máquina M_j es vecina a M_k, 0 si no

$D_{S_jS_k}$: Booleano, 1 si el servicio S_j depende del servicio S_k, 0 si no

4.4. Restricciones

4.4.1. Capacidad

$C_{mr} \geq \sum R_{pr} \forall ptq X_p, X_{0p} = m$

4.4.2. Servicios

$X_{p_i} \neq X_{p_j} si S_{P_i} = S_{P_j}$

4.4.3. Distribución

$N(distinct(Lm)) \geq spreadmin \forall mtq X_p = myp \in S$

4.4.4. Dependencia

$D_{S_jS_l} = 1 => V_{X_{Pi}X_{Pk}} = 1 \forall P_i \in S_j, P_k \in S_l$

4.4.5.

Uso transitorio Considerada en la restricción de capacidad.

5. Representación

El sistema se sustenta en varias estructuras, las principales son: Machine, Resource, Service, Process, Variables (para las constantes), Solution, Possible_moves y Tabu_list.

Los primeros 4 son elementos del problema y representan lo que su nombre dice. Existe un arreglo de cada uno de ellos dentro de Variables. Por ejemplo, si hay 4 máquinas en el problema, habrá un arreglo de 4 máquinas dentro de Variables. Cada una de estas estructuras almacena los datos para su correcto funcionamiento, muchos de los cuales son solo de consulta, pero hay otros de apoyo que ayudan al algoritmo a funcionar de mejor manera

5.1. Máquinas

Vecindad: contiene la vecindad a la que pertenece
Location: contiene la ubicación donde se encuentra
Capacities: contiene un arreglo de capacidades de cada recurso
Safety Capacities: contiene un arreglo de capacidades de seguridad de cada recurso
Resources Used: contiene un arreglo de las capacidades utilizadas de cada recurso para mantener el estado de uso de la máquina

5.2. Resource

Transient: booleano que define si el recurso es transiente o no
Weight Load Cost: int que representa el costo de carga

No se opera sobre el struct resource, solo se utiliza para leer datos.

5.3. Service

Spread Min: entero con el mínimo de ubicaciones en las que se debe distribuir el servicio
Amout Of Dependencies: entero con la cantidad de dependencias del servicio
Dependent On Service: arreglo de todos los servicios de los que depende
Locations: Arreglo de las ubicaciones en las que se encuentra el servicio. Este se actualiza cada vez que un proceso que pertenece a él se mueve.

La Estructura de servicio solo se utiliza para leer datos y usar el arreglo Locations para verificar que la condición de spread_min se cumple.

5.4. Process

Service: entero que indica el índice donde se ubica su servicio
Requirements: arreglo que indica cuanto requiere de cada recurso para funcionar
Process Move Cost: entero que representa el costo de mover este proceso

Process también es una estructura de solo lectura.

5.5. Variables

resources amount: entero de la cantidad de recursos existentes
machines amount: entero de la cantidad de maquinas existentes
services amount: entero de la cantidad de servicios existentes
processes amount: entero de la cantidad de procesos existentes
balance costs amount: entero de la cantidad de costos de balance existentes

process move cost: entero del costo de mover un proceso
service move cost: entero del costo de mover un servicio
machine move cost: entero del costo de mover una máquina

resources: arreglo que almacena todos los recursos.
machines: arreglo que almacena todas las máquinas.
services: arreglo que almacena todos los servicios.
processes: arreglo que almacena todos los procesos.
balance costs: arreglo que almacena todos los costos de balance.

Esta es la estructura principal del sistema, existe como estructura global y almacena todos los datos del sistema.

5.6. Solution

best value: double con el mejor valor hasta el momento
value: double con el valor de la solucion actual
best process asignations: arreglo con las asignaciones de la mejor solucion
process asignations: arreglo con las asignaciones de la solucion actual
initial solution: arreglo con las asignaciones de la solucion inicial

Esta es una de las estructuras principales del sistema, es global y se opera en casi todos sus campos para ir generando y evaluando nuevas soluciones.

5.7. Tabú list

List: arreglo con los valores de la lista tabu (guarda los últimos procesos que fueron cambiados)
length: largo de la lista tabu

Con esta estructura se maneja la navegacion del método Tabú Search (TS)

6. Descripción del algoritmo

El algoritmo utilizado fue una mezcla de las heurísticas de algoritmo Greedy con Tabú Search.

Para generar la solución inicial con el algoritmo Greedy, se ordenan todos los procesos de mayor a menor en base a la sumatoria de los recursos utilizados. Luego para cada proceso se buscan todas las posiciones a las cuales este se podría mover, una vez obtenidas, se evalúan y se elije la que genera la solución con el menor costo. Así se hace para cada uno de los procesos, generando así una solución inicial en base a una función miope que no se preocupa de que la ordenacion del conjunto sea la mejor, solo que cada parte caiga en el mejor lugar que pueda. Se ordenan primero los de mayor consumo de recursos ya que son en general los más dificiles de alocar.

Para mejorar esa solución, el algoritmo de Tabú search elige al azar un proceso, verifica si esta dentro de la lista tabú. Si lo está se elige otro hasta que se elija uno que no este. Si no está en la lista tabú, se procede a ver todas las alternativas donde se podría mover y luego se elige una de ellas al azar. Al finalizar, se actualizan las cargas de cada máquina, se recalcula el valor de la solucion y se verifica si es mejor solucion que la mejor solucion hasta el momento.

A continuación, se detallarán las partes más relevantes del código y se explicará su funcionalidad

6.1. Algoritmo Greedy

```
Greedy
        agregar todos los procesos a una lista
        reordenarlos aleatoriamente
        para cada proceso:
```

```
        seleccionar aleatoriamente 3 maquinas
        para cada maquina:
                si el movimiento proceso-maquina es valido,
                    guardar
        para cada maquina guardada:
                calcular valor de la solucion con ese
                    movimiento
                si valor < mejor_valor, guiardar
        aplicar movimiento con mejor valor
```

6.2. Algoritmo Tabu Search

```
Tabu Search
        repetir hasta que tiempo_pasado >= tiempo_maximo
        hacer:
                elegir un proceso
                elegir una maquina
        si movimiento proceso-maquina no es valido o esta en
            lista tabu, repetir
        ejecutar movimiento proceso-maquina
        actualizar servicios y recursos
        si valor_solucion_actual < valor_menor solucion,
            actualizar mejor solucion e imprimir
```

6.3. Algoritmo Valid_Move

```
Valid_Move
        si maquina no contiene servicio del proceso
                si vecindario de maquina contiene todas las
                    dependencias del servicio de proceso
                        si maquina tiene los recursos para almacenar
                            proceso
                                si servicio del proceso se distribuye
                                    en al menos spread_min locaciones
                                        retornar verdadero
        en otro caso, retornar falso
```

7. Experimentos

Se realizaron experimentos sobre las instancias a1_1, a1_2, a1_3 y a1_5, en cada una se realizaron 2 pruebas, una con un tiempo máximo de 500 segundos y un largo tabu de 2 y otra con un largo de 700 segundos y un largo de lista tabu de 5. Los resultados se incluyen en la carpeta instances con los nombres: salida para la solucion con mejor resultado y best_salida que en cada linea incluye los segundos que tardo y el valor de la solucion. Existe uno de estos archivos para cada ejecucion realizada

8. Conclusiones

Cada vez es mayor el uso de las tecnologías de virtualización, ya sea en aplicaciones médicas, económicas, astronómicas. Es por esto que es de suma relevancia hacer más eficiente esta herramienta para dispones de una mayor cantidad de recursos con los que trabajar. Es en ese contexto que el problema de reasignación de máquinas tiene relevancia.

El proceso de reasignación de máquinas se hace continuamente, por lo que es necesario contar con una herramienta de optimización que logre generar una distribución de procesos en las máquinas en tiempos muy limitados (en el caso del desafío, los concursantes solo disponían de 300 segundos de tiempo de ejecución para sus algoritmos). Debido a esto, una técnica completa no es posible de usar, por lo que es necesario usar heurísticas, heurísticas que sean capaces de encontrar buenas soluciones en poco tiempo.

Las dos heurísticas que obtuvieron mejores resultados fueron recocido simulado y un híbrido de búsqueda local, siendo la mejor esta última. Las mayores diferencias que tenía esta última era que utiliza dos movimientos extras (BPR y Chain) que le permiten generar distintos tipos de vecindades, además que prioriza la reasignación de los procesos más grandes versus los de menor tamaño, lo que le permite revisar primero espacios de búsqueda con distintas posiciones de los procesos más grandes, optimizando sus posiciones antes de comenzar a mover los procesos de menor tamaño, lo que mostró en la práctica lograr mejores resultados.

Además, se encontró experimentalmente que trabajar en el espacio de las soluciones infactibles es una mala opción ya que volver de esas soluciones al espacio de los factibles es muy costoso y que es mejor mover todos los procesos de un grupo de máquinas que mover unos pocos procesos de cada máquina entre todas las máquinas, sin embargo, los resultados dependen mucho de los criterios de selección de estas máquinas.

9. Bibliografía

[1] Gabriel Marques Portal, “An Algorithmic Study of the Machine Reassignement Problem”, Porto Alegre, 2012.
[2] Google, “Machine Reassignement problem description”, ROADEF/EURO challenge 20111-2012
[3] Haris Gavranovic, Mirsad Buljubasic, Emir Demirovic, “Variable Neighborhood Search for Google Machine Reassignement problem”, Electronic Notes in Discrete Mathematics 39 (2012)
[4] Deepak Mehta, Barry O’Sullivan, Helmut Simonis, “Comparing Solution Methods for the Machine Reassignment Problem”, M. Milano (Ed.): CP 2012, LNCS 7514, pp. 782–797, 2012