XV Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)

ROS: SERVICIO DE OPTIMIZACIÓN REMOTA

E. Alba[1*], J. G. Nieto[1] y F. Chicano[1]

1: Grupo GISUM, Departamento de Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática
Universidad de Málaga
Campus de Teatinos, 29071, Málaga, España
e-mail: {eat,jnieto,chicano}@lcc.uma.es, web: http://neo.lcc.uma.es

Palabras clave: XML, Servicio web, Optimización, Sistemas Distribuidos

Resumen. *En este artículo se describe ROS, un Servicio de Optimización Remota. ROS permite a sus usuarios tener acceso a un conjunto de algoritmos modernos de optimización, situados en repositorios con recursos hardware y software para optimizar problemas académicos y reales. Se presenta su arquitectura, y se detallan la especificación XML de datos intercambiados por la red y el encapsulado de algoritmos. Por último, se realiza una evaluación de ROS tanto numérica como de utilización de recursos con diversos algoritmos y problemas usando distintas configuraciones del sistema.*

1 Introducción

En la actualidad, existe un interés constante por desarrollar sistemas que integren bibliotecas de algoritmos metaheurísticos (como por ejemplo, algoritmos evolutivos [1, 5]), exactos e híbridos. Además, es siempre interesante ofrecer metodologías y servicios para facilitar la utilización de repositorios de software de cara a un mayor impacto internacional. Una característica común en casi todos los trabajos actuales es el uso de *XML* [6], que facilita el desarrollo de aplicaciones para tareas como la creación e integración de algoritmos en servicios.

Nuestra propuesta consiste en proporcionar un servicio de ejecución de algoritmos a través de Internet, ROS (*Remote Optimization Service*) [2]. Con ROS es posible establecer un sistema de servidores que gestionan el acceso y ejecución de diferentes algoritmos de optimización, totalmente distintos entre sí, y posiblemente implementados con distintos lenguajes de programación.

Este artículo se organiza de la siguiente manera. En la Sección 2 se describe la arquitectura de ROS, las entidades que lo componen y sus principales características. La Sección 3 describe los *Wrappers* utilizados para encapsular los algoritmos. En la Sección 4 se introduce la especificación de los documentos XML utilizados en el sistema. Por último, en la Sección 5 se evalúan las prestaciones reales de ROS, finalizando en la Sección 6 con conclusiones y trabajo futuro.

2 ¿Qué es ROS?

La optimización es una rama de trabajo muy dinámica debido a que nos enfrentamos a nuevos retos, nuevos problemas de ingeniería, nuevas situaciones de la industria, y un largo etcétera de nuevos desafíos en optimización [1].

ROS nace con el objetivo de establecer un servicio de optimización en Internet, es decir, facilitar el acceso y uso de múltiples algoritmos (especialmente heurísticos) creados por diferentes desarrolladores, agrupando así algoritmos muy diversos e implementados con diferentes lenguajes de programación, en un repositorio al cual se puede acceder a través de la red de manera sencilla. De este modo, se establecen dos tipos de actores principales que interaccionan con ROS: el *desarrollador*, que añade su algoritmo en un servidor (*Worker*) mediante un *Wrapper* (véase la Sección 3), y el *usuario*, que utilizando el *programa cliente*, puede interactuar de manera remota con los algoritmos disponibles y así resolver problemas de optimización.

Para dotarlo fácilmente de un carácter distribuido y multiplataforma, ROS está implementado en Java y utiliza herramientas de comunicación para establecer una jerarquía de máquinas mediante el modelo Cliente/Servidor (C/S). Esta jerarquía consta de cuatro tipos de entidades conectadas:

- El cliente ROS (*Client* en la Figura 1) es una aplicación que ofrece una interfaz gráfica a través de la cual un usuario puede realizar una serie de operaciones como: la identificación o registro de usuario, la selección del tipo de algoritmo, la selección de los servidores disponibles y la ejecución remota del algoritmo seleccionado.
- El servidor primario (*Primary Server* en la Figura 1) proporciona al cliente el servicio de identificación en el sistema, además de información sobre los tipos de algoritmos y direcciones de los demás servidores accesibles.
- El servidor de distribución (*Server* en la Figura 1) actúa de puente entre los clientes conectados y los servidores de proceso, ocultando su verdadera ubicación y equilibrando la carga entre éstos.
- El servidor de proceso (*Worker* en la Figura 1) aloja los algoritmos y los ejecuta según las órdenes especificadas por el cliente.

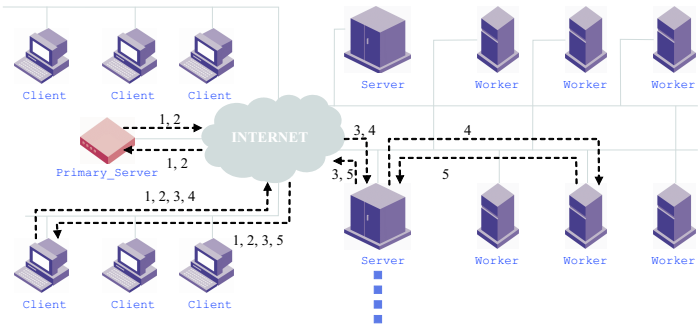

Figura 1: Esquema resumido de la arquitectura de ROS

La Figura 1 nos da una idea de la secuencia y flujo de datos a través del sistema en un ciclo de ejecución mediante los pasos descritos en la Tabla 1.

Paso	Acción	Descripción
1	Identificación de usuario	El usuario introduce su identificador y contraseña en el formulario inicial del cliente y conecta con el servidor primario para su comprobación o registro
2	Selección del tipo de algoritmo, dirección del servidor y lenguaje de programación	El cliente realiza una petición de tipos de algoritmos al servidor primario el cual le envía de vuelta una relación clasificada de los tipos de algoritmos disponibles
3	Petición de parámetros del algoritmo	El cliente realiza una petición al servidor de distribución, obteniendo un fichero XML específico del algoritmo seleccionado que contiene sus parámetros
4	Ejecución del algoritmo	El usuario introduce los valores de los parámetros del algoritmo e inicia la ejecución. El fichero XML con los parámetros introducidos se envía al servidor de distribución, que lo entregará al servidor de proceso adecuado, iniciándose así la ejecución
5	Obtención de resultados	Al terminar la ejecución del algoritmo los resultados se devuelven en el fichero XML de nuevo al servidor de distribución, el cual a su vez lo entrega al cliente

Tabla 1: Secuencia y flujo de datos a través ROS en un ciclo de ejecución

Dependiendo del sistema de comunicación que se utilize, se disponen de dos versiones diferentes del sistema: ROS con intercambio de información mediante RPC con SOAP y ROS con intercambio de información mediante comunicación con sockets Java.

3 Wrappers

Debido a la gran variedad de algoritmos que existen en ROS, que pueden estar implementados en cualquier lenguaje de programación (C, C++, Java, Modula 2, etc.) e incluso cualquier paradigma de desarrollo (Orientado a Objetos, Orientado a Componentes, Funcional, etc.), es necesaria una forma estándar de acceso a estos algoritmos. En este sentido existe una clase `Wrapper` que ofrece una interfaz común y de la cual deben heredar subclases específicas para cada algoritmo. Esta clase provee métodos para: generar ficheros de configuración a partir de un fichero XML, lanzar órdenes de compilación/ejecución e introducir resultados en el fichero XML inicial.

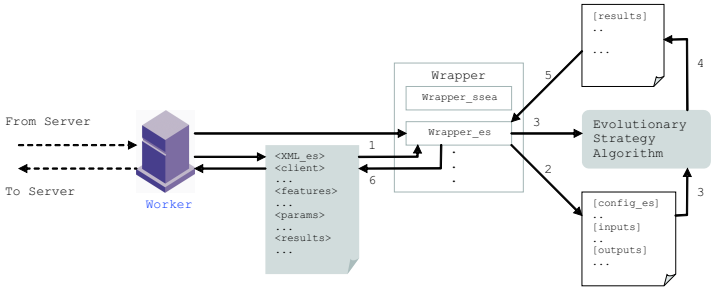

Figura 2: Relación de un Wrapper respecto al algoritmo que encapsula

En la Figura 2 se muestra un esquema del funcionamiento de un *Wrapper* típico para un algoritmo de *Estrategia Evolutiva* [1] al que llamamos `Wrapper_es`. En primer lugar, el servidor de proceso (*Worker* en la Figura 2) obtiene el fichero XML (enviado por el servidor de distribución) y lo hace disponible a `Wrapper_es`. A partir del fichero XML con los parámetros de entrada, `Wrapper_es` genera el fichero de configuración específico del algoritmo. Tras esto extrae la orden de compilación (si es necesario) y ejecución del algoritmo, lanza dicha orden y espera a que termine la ejecución. Finalmente, `Wrapper_es` incorpora los resultados en el fichero XML que será devuelto al servidor de distribución.

4 XML en ROS

En este sistema, se utiliza XML para especificar la *configuración* de un algoritmo. A partir del *DTD (Data Type Document)* `optimization_algorithm.dtd` definido para este servicio (Figura 3 der.), se especifican todos los ficheros XML que fluyen por el sistema. Como se puede observar en la Figura 3 (izq.) cada fichero se organiza jerárquicamente de manera que hay un elemento raíz `<optimization_algorithm>` que contiene a los cuatro elementos principales: `<client>` (datos del cliente/usuario), `<features>` (características de implementación del algoritmo), `<params>` (parámetros de entrada) y `<results>` (resultados). Además, cuenta con elementos `<others_...>` de propósito general con los que podemos especificar otros tipos de datos no contemplados en el DTD.

```
<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE optimization_algorithm SYSTEM "optimization_algorithm.dtd">
<optimization_algorithm>
        client   <client>
                    <client_name>jnieto</client_name>
                    <client_ip>150.214.214.26</client_ip>
                    <client_id>null</client_id>
                 </client>
        features <features>
                    <language>C++</language>
                    <compilation>make MainSeq</compilation>   } Órdenes de
                    <execution>make SEQ</execution>           } compilación y
                    <online />                                  ejecución
                    <comment />
                 </features>
        params   <params type="es">
                    <population_size>100</population_size>
                    <offsprings_size>50</offsprings_size>
                   <crossover>
                      <crossover_prob>1.0</crossover_prob>
                    </crossover>
                   <mutation>
                      <mutation_prob>0.1</mutation_prob>
                    <number_of_generations>100</number_of_genertons>
                   <migration>
                      <migration_rate>4</migration_rate>
                      <migration_number>20</migration_number>
                    </migration>
                    <fitness_function>rastrigin</fitness_function>
                    <replacement type="inclusion" />
                    <selection type="roulette_wheel" />
                 </params>
        results  <results>
                    <best_fitness>0.00288847</best_fitness>
                    <worst_fitness>546.003</worst_fitness>
                    <generation>72 </generation>
                    <computation_time>1751ms</computation_time>
                    <error>null</error>
                 </results>
</optimization_algorithm>
```

```
<?xml version='1.0' encoding='us-ascii'?>
<!ELEMENT optimization_algorithm (client, features, params, results?)>

<!ELEMENT client (client_name,client_ip,client_id?)>
<!ELEMENT client_name (#PCDATA)>
                .
                .
<!ELEMENT features (language, compilation, execution, online?, comment?)>
<!ELEMENT language    (#PCDATA)>
                .
                .
<!ELEMENT params (number_of_genes? ... cost_function?, others?)>
<!ATTLIST params
          type (ssea|mea| ... |bb|dp|simpleea|es|gp) #REQUIRED >
<!ENTITY % range
        "type (int|double|float|long|short|byte|boolean) #IMPLIED
        min CDATA #IMPLIED
        max CDATA #IMPLIED">
<!ELEMENT number_of_genes (#PCDATA)>
<!ATTLIST number_of_genes %range;>
                .
                .
                .
<!ELEMENT strategy EMPTY>
<!ATTLIST strategy
              type (fifo|lifo|other) #IMPLIED>
<!ELEMENT fitness_function (#PCDATA)>
                .
                .
                .
<!ELEMENT others (others_id*,others_value*)>
<!ELEMENT others_id (#PCDATA)>
<!ELEMENT others_value (#PCDATA)>
<!ATTLIST others_value %range;>

<!ELEMENT results (allele?, chromosome?,
individual?, fitness? ... error?)>
<!ELEMENT allele (#PCDATA)>
<!ELEMENT individual (#PCDATA)>
<!ELEMENT fitness (#PCDATA)>
                .
                .
                .
<!ELEMENT error (#PCDATA)>
```

Figura 3: XML de un algoritmo de *Estrategia Evolutiva* (izq.) y DTD general resumido (der.)

5 Evaluación de ROS

A diferencia de trabajos similares, no sólo determinamos la arquitectura del servicio ROS y lo implementamos (red y monoprocesador) sino que también lo evaluamos en este trabajo. Para realizar tal evaluación hemos considerado dos configuraciones: en una LAN y en una máquina. Para la configuración LAN los servidores se ejecutaron en 4 máquinas con procesadores *Intel* 2,4 GHz y 512MB de RAM conectados con una red *Fast Ethernet* de 100Mbps. El cliente se ejecutó en una máquina con procesador *Intel* 2,6 GHz y 512MB de RAM. En la configuración local, se ejecutaron todos los servidores y el cliente en una

misma máquina con procesador *Intel* 2,6 GHz y memoria RAM de 512MB. En la Tabla 2, se muestran los resultados respecto al *tiempo de ejecución* (te) y *tiempo de comunicación* (tc) de los algoritmos. Se presentan media y desviación típica sobre cincuenta ejecuciones independientes (los algoritmos usados no son deterministas). Además, se ha realizado un análisis estadístico que demuestra que todas las diferencias son significativas.

Algoritmo/Características	Problema	Conf.	S.O	Leng. Prog.	$\overline{te}$ (ms)	σ_{te} (ms)	$\overline{tc}$ (ms)	σ_{tc} (ms)
AG de Estado Estacionario	OneMax	LAN	Linux	Java	296.28	68,94	1510.00	549,91
AG Celular	Rastrigin	LAN	Win	Modula 2	1104.20	7.76	1778.10	39,53
AG Distribuido	Rastrigin	LAN	Win	Modula 2	1124.90	4.73	1744.60	57,64
Estrategia Evolutiva	Rastrigin	LAN	Linux	C++	3440.00	92,88	52,84	256.07
Recocido Simulado	OneMax	LAN	Linux	C++	126.00	43.53	264.64	28.91
AG para Mareas de Venecia	Pred. Series Temp.	LAN	Win	C++	148320.00	699,41	652,42	139.70
AG de Estado Estacionario	OneMax	local	Linux	Java	201.76	18.08	995,96	61,35
AG Celular	Rastrigin	local	Win	Modula 2	1336.40	42.13	1838.20	165.73
AG Distribuido	Rastrigin	local	Win	Modula 2	1311.70	23.96	1630.40	70,52
Estrategia Evolutiva	Rastrigin	local	Linux	C++	2278.10	20.45	101.52	24.72
Recocido Simulado	OneMax	local	Linux	C++	70,74	5.08	99,90	40.25
AG para Mareas de Venecia	Pred. Series Temp.	local	Win	C++	276620.00	49361.00	760,96	151.02

Tabla 2: Evaluación experimental de ROS

Observando los resultados obtenidos, encontramos algunos tiempos mayores en el caso de la ejecución local. Esto ocurre en el Algoritmo Genético Celular (cga) [1], el Algoritmo Genético Distribuido (dga) [1] y el Algoritmo Genético usado en el problema de las Mareas de Venecia (ga) [3]. La razón de dicho resultado puede deberse (así lo confirman resultados no mostrados aquí) a cargas relacionadas con el algoritmo base (implementado por otros autores en nuestro caso) y el sistema operativo. Para el resto de algoritmos: Algoritmo Genético de Estado Estacionario (ssga) [1], Estrategia Evolutiva (es) [1] y Recocido Simulado (sa) [4], el tiempo de ejecución en configuración local siempre es menor que para sus versiones paralelas.

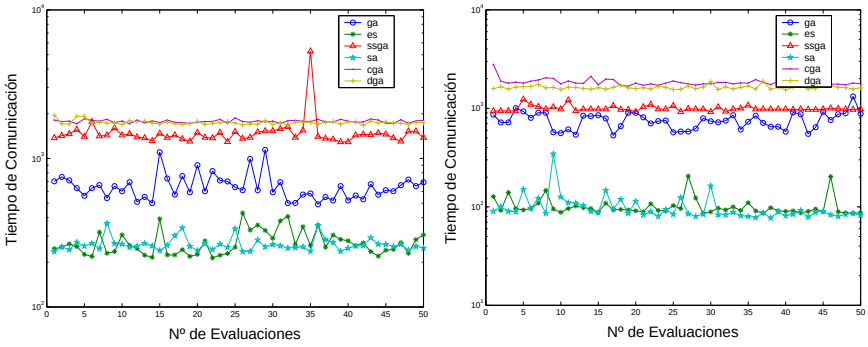

Figura 4: Tiempo de comunicación en configuración LAN (izq.) y local (der.)

Respecto al tiempo de comunicación, se obtienen tiempos mayores en configuración LAN, ya que en este caso los procesos se sitúan en puntos distanciados de una red, con las excepciones de los algoritmos AG (Mareas), Estrategia Evolutiva y AG Celular. Esto puede deberse a que el tiempo de comunicación engloba el preproceso de análisis XML y la escritura en ficheros, que para estos casos retrasa de manera significativa la comunicación en configuración local (véase la Figura 4).

6 Conclusiones y Trabajo Futuro

En este documento se ha presentado una descripción muy general de ROS, un servicio que proporciona el acceso y ejecución remota de algoritmos de optimización a través de Internet. Se basa en una arquitectura C/S ligada a la especificación XML `optimization_algorithm` desarrollada para el propio sistema.

Debido a la gran variedad de algoritmos que se pretenden añadir al servicio, y el gran número de parámetros que maneja cada uno, es necesario generalizar la especificación XML y con esto la funcionalidad del servicio. En este sentido, se pretenden estudiar en un trabajo futuro métodos para agilizar la incorporación de nuevos algoritmos al sistema, imponiendo criterios comunes al diseño, no al problema, que sigue siendo abierto a las necesidades de los usuarios. Se puede obtener el software necesario para utilizar ROS en el sitio web `http://tracer.lcc.uma.es/ros/index.html`.

Agradecimientos

Este trabajo está parcialmente financiado por el Ministerio de Educación y Ciencia y FEDER con número de proyecto TIN2005-08818-C04-01 (proyecto OPLINK). Francisco Chicano disfruta de una beca de la Junta de Andalucía (BOJA 68/2003). José M. García Nieto disfruta de una beca de colaboración del MEC (BOE 144/2005).

REFERENCIAS

[1] E. Alba. *Análisis y Diseño de Algoritmos Genéticos Paralelos Distribuidos.* PhD thesis, Universidad de Málaga, Marzo 1999.

[2] E. Alba and J. G. Nieto. ROS — Remote Optimization Service. Technical Report 2005-08, Universidad de Málaga, 2005.

[3] J.C. Hernández C. Luque, P. Isasi. Forecasting time series by means of evolutionary algorithms. In *Proceedings of the PPSN VIII*, volume 3242 of *LNCS*, pages 1061–1070. Springer-Verlag, September 2004.

[4] T. W. Manikas and J. T. Cain. Genetic algorithms vs. simulated annealing: A comparison of approaches for solving the circuit partitioning problem. Technical Report 96-101, University of Pittsburgh, Dept. of Electrical Engineering, 1997.

[5] J. J. Merelo-Guervós, P. A. Castillo Valdivieso, and J. García Castellano. Algoritmos evolutivos P2P usando SOAP. In E. Alba et al, editor, *Actas del Primer Congreso Español sobre Algoritmos Evolutivos y Bioinspirados, AEB'02*, pages 31–37, 2002.

[6] C. Veenhuis, K. Franke, and M. Köppen. A semantic model for evolutionary computation. In *Proceedings of the 6th International Conference on Soft Computing*, pages 68–73, IIZUKA, Japan, 2000.