

Algoritmos genéticos distribuidos usando SOAP

J.J. Merelo, J.G. Castellano, P.A. Castillo y G. Romero

Resumen—SOAP (simple object access protocol) es un protocolo que permite acceder a objetos remotos de forma independiente de la máquina y del lenguaje. Un cliente usando SOAP puede enviar o recibir objetos, o acceder a un método de un objeto remoto. A diferencia de otros métodos de llamada remota a procedimientos, como XML-RPC o RMI, SOAP puede usar muchos tipos de transporte diferentes, desde llamarse como un CGI hasta llamada usando sockets.

En este trabajo se ha hecho una primera aproximación a la computación evolutiva distribuida usando SOAP y el lenguaje PERL, mostrando las capacidades que ofrece, aunque todavía están poco evolucionadas.

Palabras clave— Computación distribuida, algoritmos evolutivos, XML, SOAP, Internet

I. INTRODUCCIÓN

SOAP es un protocolo estándar propuesto por el consorcio W3 ([1]) que va un poco más allá de la llamada remota a procedimientos, para permitir acceso remoto a objetos [2]. Un cliente SOAP puede acceder a servicios remotos usando el interfaz de objetos residentes en tales servidores remotos, usando cualquier lenguaje. Actualmente existen implementaciones completas de SOAP en PERL, Java, Python, C++ y otros lenguajes populares [3].

SOAP emite y recibe mensajes usando XML [4], [5] [6], envuelto en una cabecera HTTP. Un mensaje SOAP puede tener la apariencia siguiente:

```
POST /objectURI HTTP/1.1
Host: www.foo.com
SOAPMethodName:
    urn:develop-com:IBank#getBalance
Content-Type: text/xml
Content-Length: nnnn

<?xml version='1.0'?>
<SOAP:Envelope
  xmlns:SOAP=
    'urn:schemas-xmlsoap-org:soap.v1'>
<SOAP:Body>
  <i:getBalance
    xmlns:i='urn:develop-com:IBank'>
    <account>23619-22A</account>
  </i:getBalance>
</SOAP:Body>
</SOAP:Envelope>
```

Las primeras cinco líneas corresponden a la cabecera, similar al envío de un formulario mediante HTTP, mientras que el resto es un documento XML que

especifica la petición SOAP que se está haciendo. Consta de un sobre (SOAP:Envelope) y de un cuerpo (SOAP:Body), y una petición codificada en XML. En este caso, se trata de pedir un balance de una cuenta corriente; la naturaleza de la petición se expresa en la etiqueta XML más exterior, mientras que el número de cuenta se incluye en la etiqueta account.

Los servicios SOAP especifican los interfaces de los métodos a los cuales se puede acceder usando el lenguaje WSDL (Web Services Description Language, [7], [8]). WSDL es un lenguaje de descripción de interfaces, que indica de forma precisa qué llamadas se pueden hacer a un servidor SOAP y qué tipo de respuesta cabe esperar. Un ejemplo de descripción de servicios usando WSDL sería el siguiente [9]

```
<message name="GetLastTradePriceInput">
  <part name="body"
    element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
  <part name="body"
    element="xsd1:TradePriceResponse"/>
</message>
```

,con el que se describe la entrada y salida de un servicio de, para variar, precios de acciones. Con un fichero WSDL se puede especificar para lenguajes dispares un servicio, de forma que se puede usar un cliente Java para acceder a un servidor escrito en PERL, o viceversa. En algunos casos, si el cliente y el servidor están escritos en el mismo lenguaje, o si el acceso es a tipos de datos simples, no hace falta la especificación WSDL; pero sí lo es en caso de clientes genéricos.

Uno de los intereses principales en el conocimiento y uso de este protocolo es su relación con la iniciativa .NET de Microsoft [10], que pretende basar la mayor parte de sus ofertas en software en servicios remotos, que usarán el protocolo SOAP para entenderse entre sí. Aunque parezca paradójico, esta propuesta de Microsoft abre la posibilidad de muchos clientes y servidores de diferentes empresas ofreciendo servicios y usándolos, compartiendo SOAP y la especificación .NET como protocolo común.

Posiblemente debido a la juventud de este protocolo (la última recomendación [1] data del año 2000, y la mayoría de las herramientas todavía no han alcanzado la versión 1.0), todavía no se ha propuesto el uso de SOAP para computación distribuida, salvo excepciones [11]. Sin embargo, SOAP es un protocolo de alto nivel, que facilita al programador la tarea de distribuir objetos en diferentes servidores, sin tener

que preocuparse por los formatos de los mensajes, ni, en muchos casos, la llamada explícita a servidores remotos. En este trabajo proponemos tal tipo de uso, y demostramos cómo se podía utilizar, en concreto, para computación evolutiva.

Se puede imaginar un futuro en el cual diferentes ordenadores remotos ofrezcan servicios a la comunidad científica, incluyendo servicios de cálculo: por ejemplo, todos los servicios actualmente disponibles mediante formularios HTML se pueden convertir de forma más o menos fácil en servicios SOAP. Aunque se trataría básicamente de una arquitectura cliente-servidor, con todos los problemas que ello conlleva, se podría imaginar muchos ordenadores ofreciendo servicios SOAP de cálculo a la comunidad científica, igual que hoy en día se ofrecen a través de la Web.

El resto del trabajo se organiza de la forma siguiente: en el siguiente apartado (sección II) se explica la implementación de un algoritmo evolutivo usando SOAP, en la sección III se comentan los resultados obtenidos, para pasar posteriormente a la sección de conclusiones.

II. IMPLEMENTACIÓN DE ALGORITMOS EVOLUTIVOS USANDO SOAP

de innovar, aunque sí en el sentido de la implementación. Hay diversas maneras de implementar un algoritmo genético distribuido, una de las cuales es la paralelización estándar o global (Farming), en la cual, tal y como proponen Fogarty y Huang [14], Abramson y Abela [15], o Hauser y Männer [16], la evaluación de los individuos y/o la aplicación de los operadores genéticos se realiza en paralelo. Un solo procesador maestro supervisa toda la población y realiza la selección, los procesadores esclavos reciben los individuos para evaluarlos y, a veces, aplicarles los operadores genéticos.

La implementación cliente-servidor ideal de un algoritmo evolutivo distribuido constaría de un proceso servidor con varias hebras; cada una de las hebras contendría una población, y se comunicaría con las otras hebras a través del código compartido entre ellas. Cada una de las hebras escribiría en una cola propia los individuos a enviar a otras hebras. Cada hebra evaluaría sus individuos en un ordenador remoto diferente, comunicándose con un servidor SOAP.

A. Implementación del proceso maestro/esclavo

Para realizar esta implementación elegimos el módulo `SOAP::Lite` [17], de Paul Kuchenko, para el lenguaje de programación PERL, por ser una de las implementaciones más maduras (versión 0.5, de 18 de abril del 2001), y por la familiaridad de los autores con el lenguaje; además, los servidores son más fáciles de implementar con la infraestructura que existe habitualmente en los ordenadores de nuestro departamento (servidores Apache normales). Sin embargo, el principal problema de esta implementación, es que no es *thread-safe*, y por lo tanto no se podía hacer la implementación del proceso maestro usando hebras.

Se optó entonces por una implementación alternativa, tal como aparece en la figura 1: se divide el proceso maestro en un proceso “migrador” y varios procesos “granja”; y los procesos granja se comunican entre sí usando el proceso “migrador”. Cada uno de los procesos granja ejecuta un algoritmo genético autónomamente, y cada cierto tiempo, escriben en una cola situada en el proceso “migrador”, y leen de esa misma cola los individuos a enviar. La comunicación entre los procesos “granja” y el proceso “migrador”, situados en el mismo ordenador, se hace usando SOAP; SOAP permite a los procesos granja acceder a métodos de un objeto llamado `ClearingHouse`, una “cámara de compensación”, que es la que media en todas las comunicaciones entre granjas. De todos los transportes disponibles para SOAP, en este caso se eligió TCP (`SOAP::Transport::TCP::Server`), que permite crear, en un par de líneas, un “daemon” bastante ligero, que no consume demasiados recursos, y que, además, no añade demasiado overhead a la comunicación. De hecho, cuando una de las “granjas” ha encontrado la solución al problema, “mata” al migrador, lo cual hace que, cuando las otras granjas se

Fig. 1. Distribución de las diferentes tareas: en el ordenador “maestro”, hay un proceso “maestro”, también llamado “migrador”, que es el encargado de recibir y distribuir los individuos que se intercambian; el proceso “maestro” se comunica con los otros procesos usando SOAP. Los demás procesos se comunican con los ordenadores remotos usando SOAP; la evaluación de los elementos del algoritmo genético se realiza en los ordenadores remotos.

Hay múltiples implementaciones de algoritmos genéticos [12] distribuidos, habitualmente usando PVM o MPI ([13]), así que en el sentido algorítmico, la aplicación presentada en este trabajo no trata

dan cuenta de este hecho, terminen también.

B. Implementación de los procesos esclavos

Las tareas de un algoritmo genético se pueden dividir de muchas maneras diferentes; por ejemplo, cada uno de los procesos esclavos se puede ejecutar en un ordenador diferente; sin embargo, la naturaleza “libre de estados” del protocolo SOAP hace más conveniente que se implementen mediante llamadas puntuales, por eso lo único que se hace remotamente es la evaluación de las poblaciones.

Para eliminar overhead de la comunicación, los procesos granja mandan una población entera a evaluar, accediendo a un método de un objeto remoto denominado Eval. Este método remoto selecciona los individuos de la población que todavía no han sido evaluados, los evalúa, y establece su Fitness (adecuación), devolviendo una población con todos los individuos evaluados a cada una de las granjas.

Una vez más, SOAP tiene diferentes tipos de transporte, es decir, diferente forma de recibir sus peticiones y emitirlas. En este caso, y con los medios disponibles, se pueden usar principalmente 3: como un CGI (es decir, un programa ejecutado por un servidor web), mediante un CGI con `mod_perl` (similar al anterior, salvo que el intérprete de PERL está integrado con el servidor web, lo cual acelera la ejecución), y mediante un servidor web *standalone*, es decir, un servidor web exclusivo que accede de forma más directa a los objetos a los cuales se pretende acceder mediante SOAP.

Después de diferentes pruebas, se vio que un algoritmo genético con evaluación usando SOAP era 30 veces más lento usando CGI, aproximadamente 25 veces usando CGI/mod_perl, y unas 10 veces más lento usando un servidor solitario, el siguiente (SOAPdaemon.pl):

```
use SOAP::Transport::HTTP;
$SIG{PIPE} = 'IGNORE';
my $daemon = SOAP::Transport::HTTP::Daemon
  -> new (LocalPort => 9000)
  -> dispatch_to('Eval');
print "Contact to SOAP server at ",
  $daemon->url, "\n";
$daemon->handle;
package Eval;
use Indi;
sub do {
  my $class =shift;
  my @pop = @\_;
  for ( @pop ) {
    if ( !defined( $$_->Fitness() ) ) {
      my $total;
      for ( my $i = 0;
            $i < length( $$_->{\_bitstr} );
            $i ++ ) {
        $total +=
          substr( $$_->{\_bitstr}, $i, 1 );
      }
      $$_->Fitness( $total );
    }
  }
}
```

```
}
return @pop;
}
```

Este servidor SOAP, aparte de consumir pocos recursos (unos 8 megas de memoria) en la máquina donde se ejecuta, no añade demasiado overhead a las llamadas.

Otra opción habría sido usar el protocolo TCP para el transporte; el problema es que, en este caso, las llamadas remotas sí añaden más overhead (comparando con las llamadas locales), y además, en algunos casos, como es el nuestro, los ordenadores están protegidos por cortafuegos, lo que impide usar los puertos habituales usados por TCP.

C. Algoritmo Genético

El problema que se ha tratado de resolver mediante un algoritmo genético es el clásico *OneMax*, es decir, hallar una cadena binaria que tenga todos los bit a 1; aparentemente es un problema simple, sin embargo, aunque es fácil mejorar el fitness o adecuación en las primeras generaciones, en las últimas generaciones se convierte en un problema complicado de optimización combinatoria, pues es bastante complicado poner a “1” el último bit que está a 0. Eso hace que, en muchos casos, el algoritmo genético se atranque, por lo cual resulta esencial mantener la diversidad.

Los operadores genéticos usados fueron la mutación “bit-flip”, con un 50% de los bits cambiados, en media, y el entrecruzamiento clásico de dos puntos.

La política de migración era la siguiente: cada n generaciones, el algoritmo colocaba en el migrador un número e de individuos, y recogía un número i . Generalmente, $e > i$, para evitar que la cola se vacíe, y de esta forma, siempre hay individuos disponibles en la cola para que las otras granjas puedan recogerlos, y, hasta cierto punto, se mantiene la diversidad.

El método de reemplazo que se escogió era de tipo “estado estacionario”. Cada generación, un porcentaje de la población (habitualmente, el 40%) se generaba mediante mutación, otro tanto similar mediante entrecruzamiento, un 10% se recogía del “migrador” y se dejaban unos 30 individuos en el mismo.

La longitud de la cadena a hallar era de 64 bits, la población de 260 individuos por isla, y la condición de terminación del algoritmo era el encontrar una cadena con los 64 bits puestos a 1.

Los resultados de una ejecución típica se ven en la figura 2

El código del algoritmo genético, incluyendo las extensiones paralelas están disponibles escribiendo a los autores de este artículo.

III. RESULTADOS

Los resultados de una ejecución típica se ven en la figura 3

Los resultados se presentan en esta sección solamente a título informativo. Teniendo en cuenta el overhead que introduce el protocolo, y la relación

Fig. 2. Evolución del fitness: Ejecución típica de un algoritmo genético con el problema OneMax; en este caso, en 200 generaciones, no se pudo hallar el máximo absoluto.

entre tal overhead y el tiempo de cálculo del fitness, los resultados van a ser poco significativos. Teniendo en cuenta, además, que la implementación del “maestro” tampoco es la ideal, lo único que se trata de probar aquí es la posibilidad de la implementación, y no su utilidad como un procedimiento de computación distribuida y evolutiva. En este caso, el tener que ejecutar el “maestro” como diferentes procesos en una máquina introduce gran cantidad de overhead, que se suprimiría usando una implementación basada en hebras. Esta implementación tendrá que esperar a versiones posteriores del intérprete de PERL, ahora mismo las hebras están disponibles solamente de forma experimental, e incluso cuando lo estén, todos los módulos que están por debajo de SOAP::Lite tendrán probablemente que ser reprogramados para una implementación real en hebras.

En cualquier caso, como se puede observar en la figura 3 el tiempo empleado en las simulaciones desciende con el número de procesadores disponibles, hasta el tercero, a partir del cual, probablemente debido a la carga a la que está sometido el procesador (4 procesos “granja”, más un proceso “migrador”, más un servidor SOAP), el tiempo de ejecución se ve incrementado considerablemente.

IV. CONCLUSIONES Y TRABAJO FUTURO

Este trabajo es solamente una introducción y unos primeros pasos para el uso del protocolo SOAP en computación científica distribuida. Los resultados son prometedores, pero demuestran que las implementaciones tienen todavía que ganar mucho en eficiencia, eliminando overhead, como en facilidades ofrecidas (por ejemplo, el usar múltiples hebras simultáneamente). Cuando esté disponible una imple-

Fig. 3. Análisis de la ejecución del algoritmo con 1 a 4 máquinas, en cada caso, se representa el mejor tiempo del usuario obtenido.

mentación multihebra, será posible hacer una serie de servidores peer-to-peer, de forma que cada “granja” actuará como cliente y servidor (*servent*), pudiendo comunicarse con otras granjas en otras máquinas mediante mensajes SOAP.

Asimismo, se puede imaginar un futuro de servicios en la Web para diferentes aplicaciones científicas de cálculo: entrenamiento de redes neuronales distribuido, resolución de problemas de optimización combinatoria, y otros múltiples problemas. De hecho, todos los sistemas mal llamados P2P (tales como *SETI@home*), se pueden convertir al protocolo SOAP, de forma que los clientes de cálculo (que servirán de servidores SOAP) respondan a peticiones genéricas, o se puedan reconfigurar fácilmente.

En el futuro, se tratará de combinar servidores y clientes SOAP escritos en diferentes lenguajes, tales como Java, Python y C++, para comprobar su eficiencia, así como usar otros servidores, como servidores basados en XML, como el Cocoon, para ver si incrementan su eficiencia con respecto a las implementaciones en PERL, y se pueden obtener resultados competitivos con otras implementaciones paralelas de algoritmos distribuidos.

AGRADECIMIENTOS

Los autores quieren agradecer su apoyo a los siguientes proyectos: FEDER I+D project 1FD97-0439-TEL1, CICYT TIC99-0550, INTAS 97-30950. Asimismo, nos gustaría agradecer a los participantes de la lista de correo soaplite@yahoo.com su colaboración en la solución de los problemas técnicos encontrados durante el desarrollo de este trabajo, y especialmente a Paul Kuchenko, por su apoyo y simpatía.

REFERENCIAS

- [1] Don Box; David Ehnebuske; Gopal Kakivaya; Andrew Layman; Noah Mendelsohn; Henrik Frystyk Nielsen; Satish Thatte; Dave Winer, "Simple object access protocol (soap) 1.1, w3c note 08 may 2000," Available from <http://www.w3.org/TR/SOAP>.
- [2] DevelopMentor, "SOAP frequently asked questions," Available from <http://www.develop.com/soap/soapfaq.htm>.
- [3] SoapRPC.com, "SOAP software," Available from <http://www.soaprpc.com/software/>.
- [4] Erik T. Ray, *Learning XML: creating self-describing data*, O'Reilly, January 2001.
- [5] Elliotte Rusty Harold, *XML Bible*, IDG Books worldwide, 1991.
- [6] Don Box, "Inside SOAP," Available from <http://www.xml.com/pub/a/2000/02/09/feature/index.html>.
- [7] Arthur Ryman, "Understanding web services," Available from <http://www7.software.ibm.com/vad.nsf/Data/Document4362?OpenDocument&p=1&BCT=1&Footer=1>.
- [8] Venu Vasudevan, "A web services primer," Available from <http://www.xml.com/pub/a/2001/04/04/webservices/index.html>.
- [9] S. Semionov, "Web services description language," Available from http://www.sys-con.com/xml/archives/0205/simeonov/index_b.html.
- [10] Pranish Kumar Keith Ballinger, Jonathan Hawkins, "SOAP in the microsoft .NET framework and visual Studio.NET," Available from <http://msdn.microsoft.com/library/techart/Hawksoap.htm>.
- [11] Davide Marcato, "Distributed computing with soap," Available from <http://www.devx.com/upload/free/features/vcdj/2000/04apr00/dm0400/dm0400.asp>.
- [12] David E. Goldberg, *Genetic Algorithms in search, optimization and machine learning*, Addison Wesley, 1989.
- [13] J. G. Castellano; M. García-Arenas; P. A. Castillo; J. Carpio; M. Cillero; J. J. Merelo; A. Prieto; V. Rivas; G. Romero; P.R. Parra, "Objetos evolutivos paralelos," in *XI Jornadas de Paralelismo*, Universidad de Granada Depto. ATC, Ed., 2000, pp. 247–252.
- [14] Fogarty T. C.; Huang R, "Implementing the genetic algorithm on transputer based parallel processing systems," *Parallel Problem Solving From Nature*, p. 145-149, 1991.
- [15] Abramson; Abela J. A, "Parallel genetic algorithm for solving the school timetabling problem," *In Proceedings of the Fifteenth Australian Computer Science Conference (ACSC-15)*, vol. 14, p.1-11, 1992.
- [16] Hauser R.; Männer R., "Implementation of standard genetic algorithm on mimd machines," *In Davidor Y., Schwefel H. P., Männer R., Eds., Parallel Problem Solving from Nature, PPSN III*, p. 504-513, Springer-Verlag (Berlin), 1994.
- [17] Paul Kuchenko, "Soap::lite," Available from <http://www.soaplite.com>.