Uso de Tecnología Grid en Algoritmos de Optimización Evolutiva

Grid-Enabled Evolutive Optimization Framework

Elisa Guardo♦, Harold Castro♦, Germán Bravo♦, Andrés L. Medaglia◊
COMIT, Departamento de Ingeniería de Sistemas y Computación♦
Centro de Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial◊
Universidad de los Andes-Colombia
{e.guardo907,hcastro,gbravo,amedagli}@uniandes.edu.co

Resumen

Este trabajo propone un framework para soportar la ejecución en un ambiente de Grid, de aplicaciones de algoritmos genéticos desarrolladas usando JGA (Java Genetic Algorithm framework). A partir de los requerimientos que impone la paralelización de los algoritmos genéticos a la infraestructura de Grid, se definen los servicios de interface que debe articular el framework entre el middleware de Grid y la capa de aplicación en la que se encuentra JGA. Un análisis de dos posibles alternativas de implementación de este framework -gLite y Globus- indica que los servicios requeridos se encuentran en el ámbito de convergencia entre los estándares OGSA y la arquitectura definida por gLite, de modo que la escogencia de alguno de ellos para la materialización del framework puede guiarse exclusivamente por la necesidad de interoperabilidad[1].

Abstract

This work proposes a framework to enable the execution on the grid of genetic algorithms applications developed by means of JGA (Java Genetic Algorithm framework). Paralelization of genetic algorithms requires some characteristics of the underlying Grid infrastructure; these originate the interface services that the framework has to articulate between the Grid middleware and the application layer, where JGA stands. The analysis of two alternatives of implementation –gLite and Globus- reflects that the set of services required, stands in the convergence field of OGSA standard, and the architecture defined by gLite; so that, choosing one of them for implementing the framework may be uniquely stated by the interoperability objective.

1. Introducción

Para facilitar la implementación de algoritmos genéticos existen tanto utilidades comerciales como desarrollos académicos basados en diferentes lenguajes de programación y plataformas computacionales [1-3]. Java Genetic Algorithm framework (JGA) es un *application programming interface* (API) desarrollado por Medaglia y Gutiérrez [4] para facilitar la implementación de algoritmos evolutivos que resuelven problemas complejos de optimización (mono y multiobjetivo). JGA tiene la particularidad de estar escrito en Java, y en consecuencia, se puede ejecutar sobre cualquier plataforma computacional, siempre que esta hospede la Java Virtual Machine.

Medaglia y Gutiérrez [4] compararon varias herramientas para la implementación de algoritmos genéticos (GADS, GAlib, PROC GA y JGA) sobre un problema clásico de loteo e inventarios dinámico. Sobre este problema, JGA obtuvo los mejores resultados en términos de calidad de la solución; en cuanto al desempeño computacional, fue superada por GAlib [1] (implementada en C++) aunque dominó las implementaciones basadas en lenguajes comerciales como Matlab (GADS) y SAS (PROC GA).

Está claro que la portabilidad de las aplicaciones implica una disminución en el desempeño de las

[1] Este proyecto se inscribe en el marco del proyecto Ecos-Colciencias "Desarrollo de una infraestructura de GRID para cálculo y datos - código C06M02". Los autores agradecen el apoyo recibido del Fondo de Promoción a la Investigación del Centro de Investigaciones Facultad de Ingeniería -CIFI- de la Universidad de los Andes.

mismas, como lo evidencia el resultado mencionado anteriormente. Sin embargo, las facilidades de paralelización que presentan los algoritmos genéticos, se traducen en reducciones significativas de los tiempos de ejecución, como lo muestran trabajos como el expuesto en [5]. Con base en la experiencia de uso de JGA, su portabilidad expone una fortaleza significativa para habilitar su ejecución en arquitecturas de computación distribuidas y heterogéneas como las de Grid.

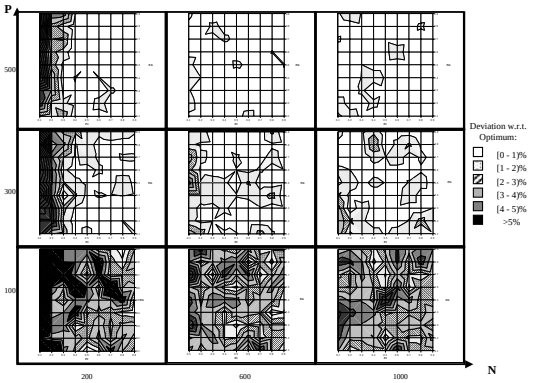
Por su potencial de paralelización, este trabajo presenta un diseño preliminar de una infraestructura de Grid que ajusta los servicios de las tecnologías de Grid existentes para atender los requerimientos de la familia de algoritmos –los de optimización mediante algoritmos evolutivos– y el tipo de proyectos científicos en que ellos participan.

2. Extracción de Requerimientos

2.1. Paralelización

Cuando se propone un nuevo algoritmo evolutivo para resolver un problema particular, el investigador debe determinar el mejor conjunto posible de parámetros para un conjunto amplio de instancias del problema. Esto implica, que el investigador debe ejecutar un gran número de ejecuciones del algoritmo variando sus parámetros. Como ejemplo, la Figura 1 muestra el ajuste de parámetros para un algoritmo evolutivo diseñado para resolver el problema de *matching* multiobjetivo [6]. En el eje horizontal varía *N*, el número máximo de generaciones (100, 300 y 500); en el eje vertical varía *P*, el tamaño de la población (200, 600 y 1000); y para cada combinación determinada de *P* y *N* (cuadro interno), varía la probabilidad de cruce p_c (eje vertical) y la probabilidad de mutación p_m (eje horizontal) entre 0.1 y 0.9 (en pasos de 0.1). El gráfico muestra la calidad de la solución (desviación con respecto al óptimo) para cada una de las 729 combinaciones (ejecuciones del algoritmo).

Figura 1. Ejemplo de ajuste de parámetros



Por tal razón, la exploración de los mejores parámetros para el algoritmo es una labor computacionalmente intensiva, que puede tomar varios días de cómputo ininterrumpidos si se ejecutan de manera secuencial en una sola máquina como es habitual.

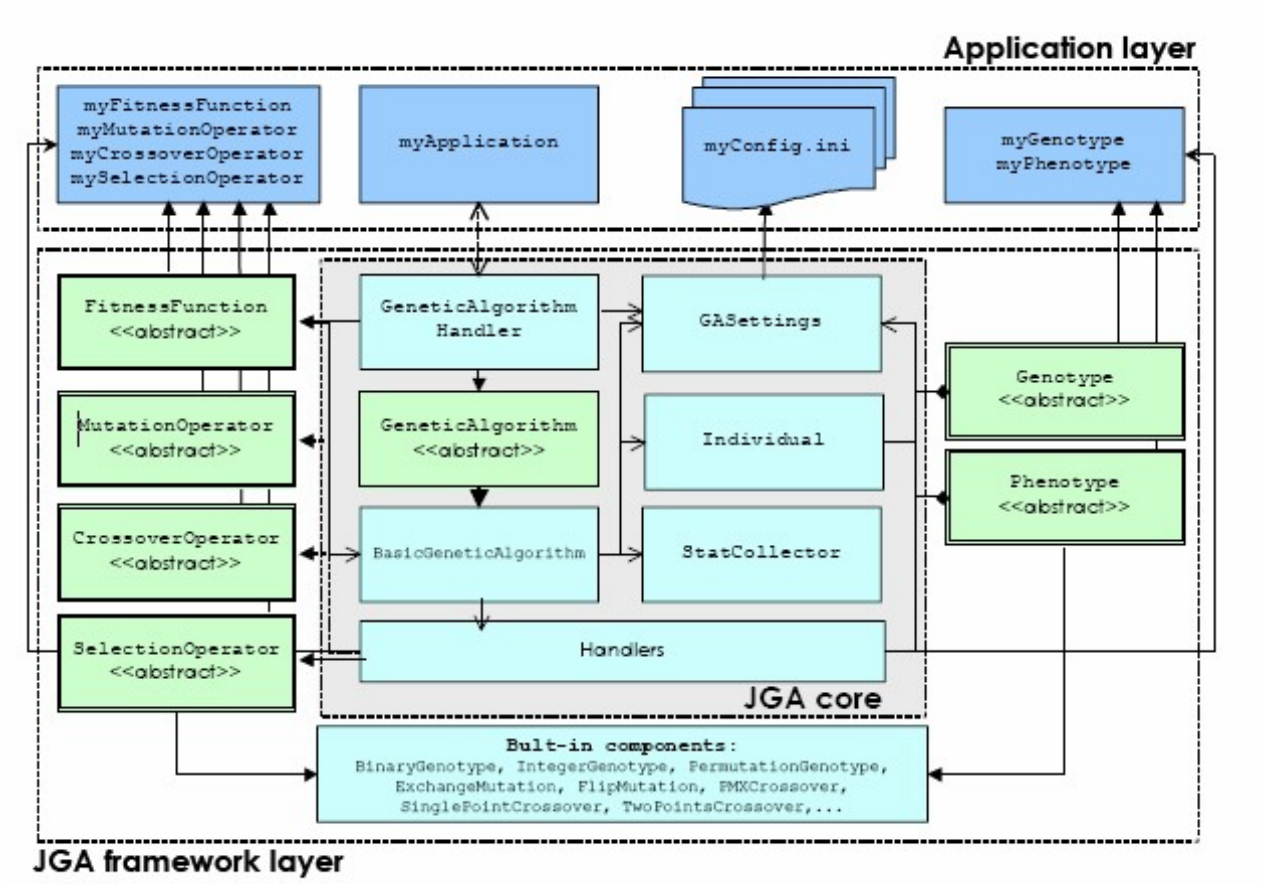
En consecuencia, un requerimiento evidente de este tipo de proyectos a la malla computacional (Grid), es la posibilidad de ejecutar varias ejecuciones del experimento en paralelo de manera que el investigador pueda analizar la incidencia de los parámetros del algoritmo en la calidad de la solución. Esta forma de paralelización se conoce en el ámbito de la computación Grid como *paralelización por datos.*

Para establecer los requerimientos que supone la paralelización de una aplicación de usuario final definida usando JGA, es importante observar el diseño del *framework.* La revisión de la teoría de los algoritmos genéticos excede el alcance de este proyecto, por tanto se sugiere al lector que no esté familiarizado con ella, leer el documento [4].

En la Figura 2 se muestra el diagrama de clases de JGA. En la capa inferior, se ilustran en líneas en negrilla los componentes fundamentales de un algoritmo genético (i.e., genotipo, fenotipo, función de adaptabilidad, operador de selección, y los operadores genéticos de cruce y mutación). En la parte central del diagrama (en gris), se muestra el núcleo de JGA, compuesto por un conjunto de clases cuyo propósito es servir como manejadores, fuera del alcance del usuario final. El núcleo de JGA es el responsable de la lógica del algoritmo genético. En la parte inferior del diagrama se muestran unas clases que implementan componentes comúnmente utilizadas en la literatura de algoritmos genéticos, que permiten el rápido desarrollo de prototipos. La capa superior, denominada también capa aplicativa, es la que programa el usuario final. De ser necesario, el usuario puede usar o extender

cualquiera de los componentes de la capa inferior para implementar un componente específico de su aplicación.

Figura 2. Diagrama de clases de JGA



La Figura 3, muestra la lógica implementada en la clase *BasicGeneticAlgorithm*, que extiende el manejador de la lógica del algoritmo. Como se muestra en el paso 3 y en el 8, cada uno de los nuevos individuos generados debe ser evaluado según la función de adaptación para medir sus méritos y su posible selección en la generación siguiente.

En este proceso de evaluación de los individuos se nota otra oportunidad de paralelización de los procesos. Esta paralelización es particularmente importante en casos en los que la función de evaluación es costosa computacionalmente. Por ejemplo, en el diseño del marco de una motocicleta cada evaluación de una alternativa puede tomar varios minutos al ser necesario utilizar una simulación de elementos finitos [7,8]. Bajo este escenario, conviene ejecutar la evaluación de cada individuo o un grupo de ellos en una maquina diferente. Esta forma de paralelización establece con cierta claridad el requerimiento de un sistema de comunicación en paralelo como Message Passing Interface (MPI) o Parallel Virtual Machina (PVM), considerando un mecanismo que permita retornar los valores calculados para los individuos al flujo de ejecución del algoritmo genético (GA) donde serán conciliados según el esquema “maestro-esclavo” de paralelización de algoritmos geneticos. Aunque existen multiples enfoques de paralelización de algoritmos genéticos, los requerimientos que estos imponen a la infraestructura de Grid son similares, por lo cual no son estudiados en profundidad en este documento.

Figura 3. La lógica detrás de BasicGeneticAlgorithm

```
1: t ← 1
2: initialize P(t)
3: evaluate P(t)
4: while t ≤ T do
5:    mutate P(t) and generate C_m(t)
6:    cross P(t) and generate C_c(t)
7:    C(t) ← C_m(t) ∪ C_c(t)
8:    evaluate C(t)
9:    E(t) ← P(t) ∪ C(t)
10:   select P(t + 1) from E(t)
11:   t ← t + 1
12: end while
```

Para conocer los diferentes esquemas disponibles de paralelización de los algoritmos genéticos se sugiere la referencia [9].

Entonces, como requerimientos funcionales de JGA y los proyectos que soporta se encuentran la ejecución simultánea de instancias y la paralelización de las operaciones de evaluación de individuos, estos servicios serán ofrecidos implementando un método de comunicación en paralelo.

La ejecución en paralelo de diferentes instancias de un experimento implementado usando JGA, es posible con sólo definir un script que repita la instrucción de *submit* -propia del *middleware*- una vez por cada instancia del experimento. La configuración predefinida de cualquier *scheduler* permite realizar una asignación que aproveche los recursos del sistema Grid.

Esta secuencia de instrucciones de *submit* no supone un cambio radical en la estructura de JGA. Además tiene la ventaja de que puede ser utilizado en otros contextos de ajuste de parámetros de metaheurísticas diferentes a algoritmos genéticos, como lo son búsqueda tabú, recocido simulado, colonia de hormigas, entre otros.

La ejecución paralela de las evaluaciones de los individuos, en cambio, requiere que JGA señale la división de trabajos y conserve el control del flujo de ejecución, de modo que pueda usar los resultados de las evaluaciones de los individuos de una población -ejecutados en diferentes maquinas, para determinar la siguiente generación. Así, JGA debe proveer al servicio de ejecución del Grid, al menos, la información de cada individuo, y la función de evaluación de la cual será sujeto. Esto podría hacerse mediante una parametrización en la clase *GeneticAlgorithmHandler* o inclusive en una clase extendida de ella similar a *BasicGeneticAlgorithm*, pero con la capacidad de paralelizar las evaluaciones.

En la sección IV se presenta una visión más detallada de lo que este requerimiento significa en la arquitectura del framework JGA_GRID.

2.2. Requerimientos no funcionales.

La necesidad de paralelizar procesos complejos de cálculo para un proyecto que involucra los algoritmos de optimización, apunta al propósito de reducir el tiempo invertido en las ejecuciones, pero existen otros requerimientos orientados a facilitar el trabajo de investigación que se explican a continuación.

- Monitoreo durante la ejecución: Para cada tarea en ejecución, el usuario espera conocer la cantidad de recursos consumidos.

- Balanceo de carga: Se espera que la infraestructura de Grid, dependiendo del estado de los recursos reformule la asignación de trabajos de modo que los recursos se usen cabalmente.

- Control de fallos: En caso de fallos se espera que se puedan tomar medidas, bien sea de recuperación o de reasignación del trabajo.

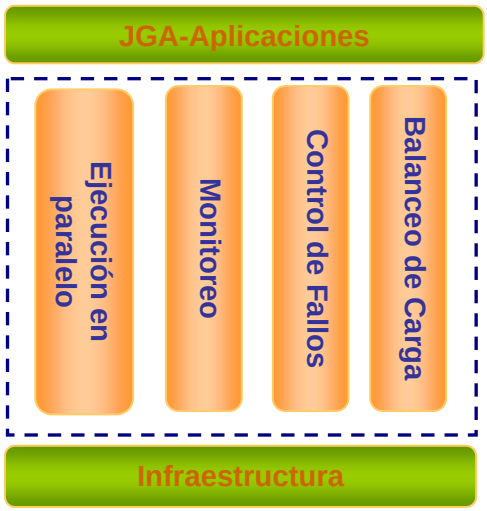
Las operaciones de balanceo de carga y control de fallos, no deben ser escaladas al usuario final, pero el servicio de monitoreo si debe actualizar la información presentada al usuario de acuerdo a lo realizado por estos dos servicios.

3. Componentes del Framework

La interoperabilidad y el bajo acoplamiento entre los componentes son características deseables en el framework que se pretende diseñar. Por ese motivo los requerimientos de JGA expuestos, deben ser provistos por el framework en forma de servicios, según lo sugiere el estándar Service Oriented Architecture (SOA), orientado justamente a las características deseadas.

El conjunto de servicios que requiere la adecuación de JGA a un ambiente de Grid se presenta en la Figura 3.

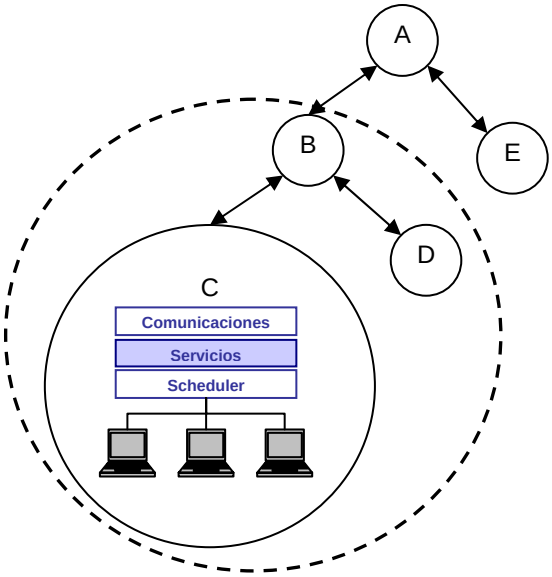
Figura 4. Arquitectura del *middleware*



Cao et al. presentan, en [13], una propuesta de balanceo de carga basada en agentes inteligentes, en la cual el balanceo de carga en el ambiente global de Grid se emula divulgando el balanceo de carga local mediante anuncios y descubrimiento. En la propuesta de Cao et al. los agentes se relacionan jerárquicamente, aunque todos presentan el mismo comportamiento, y se relacionan con sus inferiores en la jerarquía y sus hermanos en la misma. Considerando los excelentes resultados obtenidos por el trabajo de Cao et al. [13] para el proceso de balanceo de carga, y atendiendo a la naturaleza de JGA, se espera usar la arquitectura de agentes para todo el conjunto de servicios que incluye el JGA-Grid.

En la Figura 4, se muestra la estructura global del Grid. Cada agente se compone de un conjunto de estaciones de trabajo que proveen recursos de computación y dispone de un *scheduler* que permite agrupar el conjunto de recursos dispersos en varias máquinas en un solo punto de administración. El comportamiento del agente se compone de un grupo de servicios, definidos por los requerimientos del API de la capa de aplicación JGA. Adicionalmente cada agente implementa unos medios de comunicación que les permiten interactuar entre ellos y con las aplicaciones de usuario final.

Figura 4. Estructura jerárquica de agentes



A su vez, el agente se compone de un conjunto de servicios como se muestra en la Figura 3, y la naturaleza de tales servicios, permite la comunicación entre agentes mediante mensajes. Este proceso requiere un protocolo de comunicación y un tipo de mensajes propicio para la interoperabilidad, como por ejemplo SOAP.

Así, en la capa de infraestructura de la Figura 3, se encuentra un *scheduler*, como Condor, Maui o SGE, capaz de agrupar los recursos computacionales de diferentes estaciones de trabajo en una sola unidad de administración. La capa intermedia -el framework JGA-Grid, objeto de este trabajo- es capaz de: recibir cada trabajo proveniente de la capa de aplicación, que es una instancia de JGA "extendido"; extraer una lista de sub-trabajos correspondiente a la evaluación de cada individuo, para ser ejecutados en los recursos que pueda gestionar de la capa de infraestructura, bien sean local o remota; y finalmente, retornar los resultados al *host* usuario.

A continuación se define el comportamiento de cada uno de los servicios de los agentes, y la comunicación entre ellos.

3.1. Ejecución en Paralelo.

Este servicio es equivalente a la implementación del ambiente de ejecución en paralelo que ofrecen algunos *schedulers* como SGE. Sin embargo, por tratarse de un servicio, recibe la solicitud de ejecución en formato

XML. Por otro lado, el uso de instrucciones específicas de los diferentes *schedulers*, sugiere la implementación del patrón *factory* en el servicio de ejecución para proveer independencia del *scheduler* de cada agente.

La información contenida en cada solicitud es la de la población a evaluar (número de individuos y las características de cada uno) y la función de evaluación por ejecutar. El servicio encapsula entonces esa información en un objeto y la pone a disposición del servicio local de balanceo de carga. La respuesta obtenida del servicio de Balanceo de Carga, permite al Servicio de Ejecución enviar cada uno de los individuos al *scheduler* del agente indicado.

Se requiere que la ejecución del trabajo del individuo se haga efectiva desde el Servicio de Ejecución, puesto que es éste quien controla el ciclo de vida del objeto del trabajo global y es el encargado de recopilar la información de los individuos evaluados en el flujo principal del algoritmo genético.

3.2. Balanceo de Carga

A nivel local, es decir, al interior de un agente, el Servicio de Balanceo de Carga tiene acceso directo al objeto para el cual el Servicio de Ejecución ha solicitado recursos. El Servicio de Balanceo de Carga, determina si los trabajos pueden ser asignados por el *scheduler* local o si debe solicitar recursos a otro agente. Luego de la ubicación de los recursos, el servicio de balanceo de carga modifica la información de recursos asignados a cada individuo en el objeto del trabajo JGA.

El algoritmo de balanceo de carga puede ser tan sencillo como los ofrecidos por los *scheduler*, como first-in-first-out (FIFO) o basados en políticas, o tan sofisticados como el propuesto en [13] que usa una herramienta de evaluación predictiva de los recursos.

En caso de requerirse recursos de un agente remoto para uno o varios individuos, el servicio de Balanceo de Carga inicia un proceso de descubrimiento basado en anuncios, indicando la cantidad de recursos requeridos. Cuando obtiene la respuesta de un agente para atender sus requerimientos, ésta incluye la información que el Servicio de Ejecución local necesita para la asignación del trabajo –la evaluación del individuo- como se indicó en la sección anterior.

3.3 Monitoreo

El proceso de monitoreo se implementa mediante eventos asociados a cada objeto JGA en ejecución y adicionalmente el servicio permite actualizar periódicamente el estado de cada uno de los trabajos a partir de la información del *scheduler* subyacente.

Cuando el *scheduler* que administra el recurso que ejecuta uno de los hilos del trabajo global, dispara un evento en el objeto, el servicio de monitoreo despliega el comportamiento asociado con ese evento. Parte de ese comportamiento es determinar cuál hilo ha disparado el evento y qué tipo de evento se ha disparado. Entre estos eventos se encuentran:

- La finalización exitosa de la ejecución.
- La interrupción de la ejecución.

Adicional al manejo de eventos en la ejecución, el servicio de monitoreo consulta periódicamente al *scheduler* del *host* de ejecución, para construir un *log* de ejecución para cada uno de los hilos de un trabajo y para definir *checkpoints* del proceso de ejecución.

3.4 Control de Fallos.

El servicio de Control de Fallos es llamado cuando el servicio de monitoreo informa una interrupción en la ejecución. En ese caso sencillamente se reinicia el proceso de asignación de recursos o en un enfoque más sofisticado, se reanuda la ejecución a partir del último *checkpoint* registrado por el servicio de control de fallos.

Para concluir, la implementación de tales servicios debe ajustarse a la pila de componentes elegida según la oferta existente en el momento de la implementación. Este proceso constituye la segunda etapa del proyecto al que se suscribe el presente trabajo.

4. Opciones de Diseño

Tanto Open Grid Services Architecture (OGSA) como la arquitectura de gLite se basan en SOA cuya ventaja principal es que facilita la interoperabilidad entre los servicios de Grid con arquitecturas basadas en SOA. A continuación se analizan los servicios declarados por la arquitectura de gLite necesarios para soportar JGA en una infraestructura de Grid, posteriormente se hace un análisis similar con la arquitectura OGSA. La especificación de la arquitectura de gLite puede ser consultada en [10].

4.1. La Arquitectura gLite

De acuerdo con la arquitectura mostrada por Laure et al. [11] los componentes de gLite se agrupan en cinco familias: (1) Servicios de Seguridad, (2) Información y Monitoreo, (3) Servicios de Administración de trabajos y (4) Servicios de Datos y (5) Servicios de Acceso. Para la interface entre JGA y la infraestructura de Grid son necesarios servicios de dos de estos grupos, como se describe a continuación:

4.1.1. Servicios de Acceso. Permite proporcionar una interface estándar de comunicación entre la aplicación desarrollada a partir de JGA y los demás servicios del *middleware* requeridos por el usuario final. La correcta implementación de este servicio debe facilitar la efectiva portabilidad de tales aplicaciones de un Grid gLite a uno OGSA.

4.1.2. Servicios de Administración de Trabajos. Varios servicios de este grupo atienden los requerimientos identificados así:

Elementos de Computación (CE): resuelven las necesidades de asignación y administración de trabajos dependiendo de la información acerca de los recursos disponibles.

4.1.3. Sistema de Administración de Cargas (WMS). Permite proveer balanceo de carga y facilitar el control de fallos mediante un "*meta-scheduler*" que sigue políticas de agendamiento definidas por el usuario.

4.1.4. Procedencia de Trabajos. Permite proporcionar control de fallos dado que se orienta a hacer persistir información de los trabajos en ejecución para diferentes fines, por ejemplo, ser ejecutados nuevamente. Este servicio, al momento de escribir este documento se encuentra en etapa de prototipo dentro del proyecto gLite.

Para la primera versión de las interfaces se decidió no incluir utilidades de seguridad ni de administración de datos. La primera no es necesaria mientras la utilización de JGA se enmarque dentro del proyecto CampusGrid de la Universidad de los Andes y la segunda es temporalmente innecesaria mientras que los problemas a los que apunta JGA no requieran un manejo intensivo de datos.

4.2. La Arquitectura OGSA

La arquitectura OGSA fue diseñada para soportar un grupo disímil de proyectos con requerimientos muy diversos, razón por la cual está constituido por un buen número de servicios. De estos servicios sólo se describen los requeridos por el API que se pretende soportar en la infraestructura Grid de este proyecto. Una revisión completa de la arquitectura OGSA puede obtenerse de [12].

4.2.1. Servicios de Administración de Ejecución (EMS). Este grupo de servicios se orientan a la identificación de recursos, la asignación de trabajos y la administración de la ejecución de los mismos. El servicio de administración de la ejecución se relaciona con el de control de fallos para facilitar la operación de ese servicio. La implementación de este servicio se dirige a la solución de los requerimientos funcionales y no funcionales, como son la tolerancia a fallos y el balanceo de carga.

4.2.2. Servicios de Administración de Recursos (RM). OGSA define estos servicios en colaboración con los servicios de administración de recursos de niveles adyacentes, como son los de infraestructura y recursos responsables de los recursos físicos y lógicos. Estos servicios permiten suplir las necesidades de monitoreo de utilización de recursos.

4.3. Ejecución en paralelo y control de fallos

Tanto la arquitectura OGSA como la de gLite suponen el soporte de una infraestructura de Grid de la cual extraen los recursos necesarios para la prestación de los servicios.

Los requerimientos de ejecución en paralelo y de control de fallos con frecuencia se implementan configurando, inmediatamente sobre la infraestructura o a partir de un *scheduler*; un "ambiente de ejecución" que permita la utilización de los recursos de la forma en que el servicio los requiere.

En el caso de la ejecución se requiere un ambiente de ejecución en paralelo (PE) acorde con algún paradigma de programación en paralelo como MPI o PVM, mientras que en el caso del control de fallos es usual implementar un "ambiente de puntos de chequeo" que permiten establecer puntos de recuperación de los trabajos en ejecución.

Por otra parte, el servicio debe coordinarse con la API JGA en sí misma para determinar el mecanismo de solicitud de los servicios.

5. Conclusiones y Trabajos Futuros

Se analizaron los requerimientos que un API de nivel de aplicación orientado a la implementación de algoritmos genéticos para incluir dentro de sus facilidades la ejecución en paralelo en un ambiente de Grid.

Así mismo, se ha visualizado tales requerimientos en términos de servicios tal como lo sugiere la tendencia de las tecnologías de Grid actuales, para asumir la integración del API en cuestión –JGA- independientemente de la infraestructura, aprovechando la facilidad que para ello aportan la arquitectura orientada a servicios -SOA-y la implementación del API mismo.

La materialización de la integración que se describe en este trabajo, constituye una segunda etapa del proyecto que lo enmarca. Esa implementación requiere un trabajo al interior del API que implemente uno de los enfoques de algoritmos genéticos en paralelo y que los comunique al *framework* o *middleware* a manera de solicitud de servicios. En el nivel del *framework* y de la infraestructura se requiere un trabajo que bien, elija componentes existentes que provean los ambientes de ejecución necesarios para la ejecución en paralelo y el control de fallos o que implemente, o uno a la medida.

6. Referencias

[1] M. Wall (2005). GAlib: A C++ Library of Genetic Algorithm Components. MIT. Ultimo acceso Mayo 21, 2007 de http://lancet.mit.edu/ga

[2] SAS Institute Inc. (2003). SAS/OR 9.1 User's Guide: Local Search Optimization. Cary, NC: SAS Publishing.

[3] S. Silva (2005). GPLAB: A Genetic Programming Toolbox for MATLAB. Evolutionary and Complex Systems Group, University of Coimbra. Ultimo acceso Mayo 21, 2007 de http://gplab.sourceforge.net.

[4] A. L. Medaglia, E. Gutierrez (2006). An Object-Oriented Framework for Rapid Development of Genetic Algorithms. En Handbook of Research on Nature Inspired Computing for Economics and Management. Jean-Phillipe Rennard (Ed.). Capítulo XL, pp. 608-624.

[5] D. Lima, Y. Onga, Y. Jinb, B. Sendhoffb, B. Lee (2006). Efficient Hierarchical Parallel Genetic Algorithms using Grid computing. Future Generation Computer Systems Elsevier B.V, Volume 23, Issue 4, May 2007, Pages 658-670.

[6] A. L. Medaglia, S.-C. Fang, (2003). A genetic-based framework for solving (multi-criteria) weighted matching problems. *European Journal of Operational Research*, 149(1): 77-101.

[7] J.E. Rodríguez, A. L. Medaglia, J. P. Casas (2005). Approximation to the optimum design of a motorcycle frame using finite element analysis and evolutionary algorithms. Ellen J. Bass, (editor). Proceedings of the 2005 IEEE Systems and Information Engineering Design Symposium, pp. 277 – 285.

[8] J.E. Rodríguez, A. L. Medaglia, C.A. Coello-Coello (2006). Diseño del marco de una motocicleta con algoritmos evolutivos multiobjetivo y elementos finitos. XIII Congreso Latino-Iberoamericano de Investigación de Operaciones (CLAIO), Montevideo, Uruguay.

[9] M. Nowostawski, R. Poli (1999). Parallel genetic algorithm taxonomy. Proceedings of the Third International Conference on Knowledge-Based Intelligent Information Engineering Systems.Pages 88-92.

[10] EGEE Middleware Architecture. (2004). EGEE. Ultimo acceso Mayo 21, 2007 de https://edms.cern.ch/document/476451/

[11] E. Laure, S. M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, A. Edlund (2006). Programming the Grid with gLite. Computational Methods Science and Technology. Pages 33-45.

[12] The Open Grid Services Architecture, V 1.0. Global Grid Forum (2005). Ultimo acceso en Mayo 21, 2007 de http://forge.gridforum.org/projects/ogsa-wg.

[13] J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd (2005). Grid load balancing using intelligent agents. Future Generation Computer Systems 21(1):135-1.