Simulación Paralela Basada en Agentes de Sociedades Precolombinas: guanacos y movimiento de agentes

Andrés Rodríguez Rodríguez

Resumen—Con el propósito de entender la evolución social de una sociedad cazador-recolector basada en la Patagonia precolombina se propone un modelo basado en agentes; este está siendo implementado utilizando FLAME (Flexible Large-scale Agent Modelling Environment), la cual es una herramienta para desarrollar simuladores paralelos basados en agentes, con el motivo de ser capaces de obtener ejecuciones más detalladas en un tiempo razonable. Nuestro objetivo es mejorar el rendimiento de la aplicación y sus resultados. Las pruebas realizadas hasta el momento en la actual implementación parcial del simulador muestran resultados prometedores para hasta 32 procesadores.

Palabras clave—Computación de Altas Prestaciones, FLAME, Modelo Basado en Agentes, Paralelismo, Simulación.

Abstract—In order to understand the social evolution of a hunter-gatherer society based on pre-colombian Patagonia an agent based model is proposed; it is being implemented using FLAME (Flexible Large-scale Agent Modeling Environment) which is a framework for developing parallel agent-based simulators, in order to be able to execute more detailed in a reasonable time. Our goal is to improve the performance of the simulation and its results. The tests done so far on the current partial implementation of the simulator show promising results for up to 32 processors.

Index Terms—Agent Based Modeling (ABM), FLAME, High Performance Computing, Parallel, Simulation.

1 Introducción

Presentamos en este trabajo nuestra aportación al proyecto de SimulPast [1], el cual pretende estudiar el pasado en la época precolombina en la región de la Patagonia mediante simulaciones de modelos basados en agentes.

SimulPast es un proyecto transversal entre disciplinas de humanidades, ciencias sociales y ciencias puras, con el objetivo de usar la simulación como laboratorio y que tiene varios casos de estudio con tal de entender el comportamiento humano.

Por lo tanto, contamos con la ayuda del Grupo de Sociología Analítica y Diseño Institucional (GSADI) del departamento de sociología de la Universitat Autònoma de Barcelona, quienes han desarrollado un modelo en NetLogo [2] el cual, por restricciones del propio entorno, no permite simulaciones de una granularidad muy fina ni aprovechar el paralelismo que ofrecen los clusters para mejorar el rendimiento de la aplicación.

Es nuestro propósito, por tanto, conseguir aumentar el rendimiento y la cantidad y calidad de la información extraída de cada simulación, para ello se nos ha proporcionado el código fuente del proyecto PataFlame [3], el cual es un modelo básico y funcional con varios agentes sobre el cual se trabajará.

Hemos realizado en este proyecto el diseño e implementación del modelo de manadas de guanacos, animales de la Patagonia que eran la principal caza de los clanes que habitaban la Patagonia precolombina con las funcionalidades que nos aportan información en la simulación como su movimiento, reproducción o muerte; así como diseñado e implementado el movimiento que estos clanes hacían por el territorio.

Se ha comprobado también el rendimiento de la aplicación después de integrar nuestro trabajo al trabajo realizado por Imbernón [4].

Este artículo está estructurado en los siguientes apartados: en la sección 2, Objetivos, donde explicaremos en detalle los objetivos de este proyecto; en la sección 3, Estado del arte, explicaremos el estado del proyecto SimulPast y su modelo, cómo funciona la herramienta FLAME utilizada para este trabajo y el estado del proyecto PataFlame; en la sección 4, Metodología, explicaremos qué y cómo hemos desarrollado el proyecto y conseguido nuestros objetivos; en la sección 5, Resultados, mostraremos y valoraremos los resultados obtenidos; finalmente en la sección 6, Conclusiones y líneas abiertas, resumiremos los logros alcanzados en este proyecto e indicaremos posibles futuras líneas de acción.

2 Objetivos

El objetivo de este proyecto es diseñar, desarrollar e implementar aspectos específicos del modelo PataFlame, así como comprobar su rendimiento.

Un elemento importante del modelo son los recursos necesarios para la subsistencia de los clanes, los cuales en el modelo inicial proporcionado no había más recursos que los que proporcionaba el terreno lo cual provoca una situación de recursos estáticos con un crecimiento basado

- *E-mail de contacto: andres.roro88@gmail.com*

- *Mención realizada: Ingeniería de computadores.*
- *Tutor del trabajo: Eduardo Cesar Galobardes*
- *Curso: 2014/15*

en la estación. Este modelo no es del todo adecuado dado que las sociedades eran del tipo cazador-recolector y la caza es un recurso móvil y de crecimiento variable que en un ambiente de simulación basado en agentes se puede modelar como tal. En la Patagonia de la época precolombina la caza se realizaba sobre los guanacos, una especie de camélido oriundo de Sudamérica [5].

Nuestro principal objetivo es añadir el agente que modela la manada de guanacos en nuestro modelo, así como todas las acciones que pueden llevar a cabo y que puedan tener una repercusión sobre los resultados de la simulación. Estas acciones son varias y las describiremos a continuación:

Movimiento

Tal como se ha mencionado anteriormente, la caza no es un recurso estático por lo que el agente que modela a los guanacos ha de estar en movimiento bajo ciertas condiciones, como la estación del año, que el movimiento ha de ser un paseo aleatorio siguiendo una distribución de cola pesada o Lévy flight [6] y que aún así estos agentes han de huir de los agentes que modelan a los grupos de personas o clanes.

Reproducción y muerte

La reproducción de estos agentes tiene también un impacto significativo en los resultados de la simulación ya que para simular una población de seres vivos es necesario representar su procedimiento reproductivo y de muerte.

El segundo objetivo para de trabajo es el de modelar las migraciones que realizan los clanes para conseguir recursos. Estos movimientos de clanes tienen un comportamiento parecido a los movimientos que realizan los guanacos puesto que son un paseo aleatorio Lévy flight de cola pesada que depende de la estación, con la diferencia que los clanes van en búsqueda de los guanacos.

3 ESTADO DEL ARTE

3.1 Modelo SimulPast

Este trabajo parte del caso de estudio número 2 del proyecto SimulPast el cual está llevado a cabo por el GSADI en el departamento de sociología de la Universitat Autònoma de Barcelona. Su versión del modelo está realizada en el entorno de simulación basada en agentes NetLogo el cual genera código JAVA para las simulaciones. Las restricciones que tiene el lenguaje y el entorno son tales como el rendimiento en que simulaciones largas y con muchos agentes puede tomar mucho tiempo de cómputo, incluso desactivando los gráficos, dado que NetLogo no corre en paralelo con paso de mensajes aunque sí usa memoria compartida en los núcleos de una misma máquina.

3.2 FLAME

Así pues, para obtener más información y mayor rendimiento se decidió utilizar FLAME (Flexible Large-scale Agent Modeling Environment) [7]. FLAME es una herramienta que permite crear modelos basados en agentes, los cuales están basados en un modelo de computación llamado máquinas de estados finitos. Estas máquinas de estados están formadas por estados y funciones de transición entre estos. Solo hay un estado de inicio y un estado de fin, los cuales serán recorridos mediante sus funciones de transición de principio a fin en cada iteración hasta el final de la ejecución.

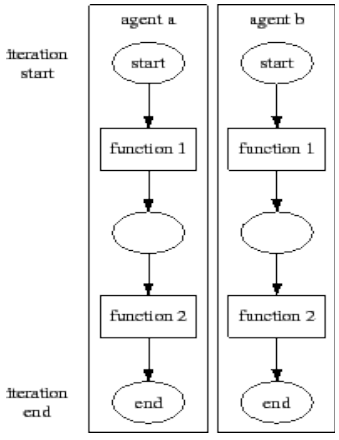

Fig. 1. Iteración de 2 agentes en FLAME con 2 funciones cada uno.

Cada agente tiene una memoria propia para sus variables que pueden ser usadas en las funciones de transición. Esta memoria es individual y para poder compartir la información que tiene con otros agentes se utiliza paso de mensajes. El paso de mensajes en FLAME depende de la librería "libmboard" la cual genera una "pizarra" en el que los agentes dejan y recogen sus mensajes. Para el uso de esta librería en clusters o sistemas distribuidos es necesario tener las librerías de MPI (Message Passing Interface) [8]. Las funciones de transición pueden recibir o enviar mensajes lo cual se realiza de modo síncrono, por lo que no se leerán hasta que se hayan enviado todos. Dado que FLAME puede correr en clusters la transmisión de mensajes se realiza mediante broadcast, todos los agentes reciben mensajes de todos los agentes y por lo tanto cuando se quiere leer un mensaje el agente receptor ha de filtrarlo [9].

En las funciones de transición los agentes pueden desaparecer de la simulación, de la misma forma, durante la simulación pueden generarse nuevos agentes.

Para generar un modelo en FLAME primero hay que describirlo. La descripción del modelo se realiza mediante el lenguaje XML (eXtensive Markup Language). Mediante su estructura de tags se describen: el nombre del agente, las constantes de entorno, el archivo donde estarán sus funciones implementadas, unidades de tiempo, los tipos de datos que se requieran, las variables de la memoria propia del agente y sus tipos, las funciones de los agentes con el orden que han de seguir, la condición que se ha de cumplir para entrar en ese estado y sus mensajes de entrada o de salida y finalmente se declaran los mensajes que puede enviar el agente y los datos que este envía.

Una vez descrito el modelo, se han de implementar sus funciones, esto se realiza en lenguaje C y cada función a

implementar ha de tener el nombre de la función de transición correspondiente del archivo de descripción.

Finalmente, para ejecutar el modelo primero se ha de utilizar una herramienta de FLAME llamada "xparser" que analiza los archivos de descripción del modelo y los traduce en código fuente para la simulación del modelo. A continuación genera un archivo "Makefile" que permite compilar este código fuente introduciendo las funciones de transición implementadas anteriormente para generar el ejecutable con la simulación.

Por último, debemos crear los agentes para usar en esta simulación, los cuales se describen utilizando también el lenguaje de marcadores XML. En estos ficheros se les da el valor inicial a las variables de la memoria de cada agente así como también se puede dar valor a las constantes de entorno.

3.3 Estado del simulador inicial

En el caso del simulador inicial, el cual se nos proporcionó del proyecto PataFlame, teníamos un modelo básico realizado en FLAME con constantes de tiempo como que cada iteración corresponde a un día, una semana a siete días, treinta días a un mes, una estación a seis meses y un año a doce meses y tres agentes con las funciones básicas de cada uno de ellos. Estos agentes son:

1) Patch

Este agente representa el terreno donde se lleva a cabo la simulación, es decir parcelas de la Patagonia, la cual ha sido dividida en una parrilla de 15x15 lo cual, dado que la Patagonia cubre un área de 1 060 631 km^2, nos da un tamaño de parcela de alrededor de 68x68 km.

Este agente en su versión inicial contenía como variables de memoria el identificador de patch, las calorías que contiene, el tipo de terreno, sus coordenadas X e Y, sus tasas de reposición de calorías y la estación del año en la que está.

Este agente tenía definidas cuatro funciones de transición.

La primera se encargaba de inicializar el tipo de patch, la segunda función envíaba a los clanes las calorías que contiene el patch, la tercera función sólo se realizaba el sexto día de cada semana y consistía en reponer las calorías del patch y la última función se encargaba de cambiar la tasa de reposición del patch cada vez que se cambiaba de estación.

2) Clan

El agente clan modela las familias o clanes que vivían en la Patagonia precolombina. Estos clanes estaban formados por individuos que recolectan los recursos del patch en el que se encuentran para sobrevivir. Todo clan tiene un líder y el propio clan se encarga de sustituir al líder en caso de que este no sobreviva.

Las variables de memoria de este agente contenían en su versión inicial un identificador de clan, la posición del grueso del clan (dado que individuos del clan pueden estar alejados de este por que están en otro patch cazando), quién es el líder, y las calorías que necesitan, han obtenido y tienen en reserva.

Este agente también tenía cuatro transiciones definidas.

La primera recogía las necesidades de calorías de los individuos que forman el clan para mandarlas al patch, en la segunda función se recogían del patch las calorías para en la tercera función de transición distribuirlas entre los miembros. La última función se realizaba cuando el líder del clan muere y simplemente escogía el nuevo líder.

3) Individuo

El agente individuo modela a cada individuo que vivía en la Patagonia en la simulación. Modelaremos que estos individuos comen para sobrevivir y se van haciendo mayores con el paso del tiempo.

Las variables de memoria de estos individuos contenían su identificador propio y el del clan, las calorías que necesita, el patch en el que está, su edad, su género, si es el líder y si está embarazada en caso de ser del sexo femenino.

Nuevamente, encontramos cuatro funciones de transición definidas para este agente.

En la primera decía al clan las calorías que necesita para en la segunda recibirlas. En el tercer estado se hacía mayor si ha pasado un mes y finalmente en la última función calculábamos si este individuo sobrevivía.

4 METODOLOGÍA

La metodología utilizada en este proyecto se basa en la metodología de desarrollo en espiral. Se usa esta metodología por el hecho de contar con la integración del modelo con el modelo integrado con Imbernón [4] como una tarea a planear con riesgos que tomar en cuenta antes de desarrollar y probar.

En este proyecto hemos trabajado como se ha dicho en los objetivos sobre el diseño e implementación del modelo de los guanacos y el movimiento de los agentes sobre el terreno.

Como consideración previa debemos tener en cuenta varios requisitos implícitos a la realidad:

En primer lugar, la simulación tiene lugar en un espacio el cual es un agente que ya nos venía definido por el proyecto anterior y por unas unidades de tiempo también definidas en el modelo.

Este espacio y este tiempo influyen en la simulación, puesto que el crecimiento de recursos no es el mismo en la estación seca o en la estación húmeda, las cuales hemos tomado así para simplificar la realidad ya que a grandes rasgos la climatología se puede modelar así. De forma similar, los recursos en zonas más interiores o cercanas a montañas no son iguales a zonas costeras o de río.

Así pues, sabemos que en la Patagonia precolombina los habitantes del ecosistema solían acercarse a las zonas húmedas como ríos o costa durante la estación seca ya que sus probabilidades de supervivencia eran mayores dado a que allí es más probable conseguir los recursos necesarios. Por el contrario, en la estación húmeda y con las lluvias los habitantes se movían por todo el territorio para aprovechar, con menos competencia, los recursos que han

surgido por estas.

Otra consideración previa tiene en cuenta la geografía de la zona. La parte occidental de la Patagonia está formada por la cordillera de los Andes la cual no es abundante en recursos además de ser de difícil acceso para los habitantes de la zona en aquel tiempo. Por lo tanto, en nuestro modelo para simplificar la realidad está modelada la alta montaña, pero hemos de considerar que los agentes no tienen ningún interés en la zona o que muy difícilmente van a acceder a ella. En el apartado de la geografía se encuentra también que la zona norte y sur de nuestro modelo son ríos y la zona más oriental es mar.

Con estas consideraciones previas nos disponemos a explicar la metodología utilizada para el diseño e implementación de las manadas de guanacos con sus características y funciones.

4.1 Manadas de guanacos

Antes de abordar el diseño hemos de conocer más sobre los guanacos. Estos seres son mamíferos oriundos de Sudamérica que se alimentan de bulbos, musgo y hierbas, lo cual se traduce en nuestro modelo en que tendrán un comportamiento parecido al que tendrán los humanos, irán de patch en patch buscando su alimento. Su velocidad máxima es de alrededor de 64Km/h, lo cual es importante para determinar el tiempo que tardan en recorrer un patch. La época de celo ocurre entre noviembre y febrero y la gestación de la cría toma 11 meses, lo cual es importante para determinar cada cuánto se reproducen y cuántos guanacos se añaden al modelo cada vez que esto ocurre. El promedio de vida del guanaco es de entre 20 y 25 años. Los guanacos huyen de los depredadores de la zona, lo cual nos indica que huirán de los humanos dado que los humanos les dan caza. Viven en manadas de unos 50 machos o un sólo macho y de 7 a 9 hembras. Su principal depredador es el puma, a parte de los humanos.

Una vez contamos con esta información podemos llevar a cabo qué requerimientos han de cumplir como agente para ser capaz de aportar información a nuestra simulación:

1) *Movimiento*
   - Los guanacos han de ser capaces de cambiar su posición sobre el terreno.
   - El movimiento ha de ser aleatorio, usando una distribución de cola pesada o Lévy flight que se explicará posteriormente en este documento.
   - Han de evitar entrar en territorios en los que haya agentes del tipo clan.
   - El tiempo que les toma en cambiar de patch ha de ser acorde a su velocidad real.
   - Durante la estación seca han de dirigirse a zonas cercanas a los ríos.
2) *Reproducción*
   - Los guanacos han de ser capaces de reproducirse.
   - La reproducción sólo puede llevarse a cabo si hay familias con hembras vivas.
   - Ha de ocurrir una vez al año.
   - Ha de tomar en cuenta los guanacos que hay dentro del mismo territorio dado que las crías del año anterior pasarán a ser adultas y estas formarán nuevas manadas.
3) *Muerte*
   - Los guanacos han de poder morir, es decir, desaparecer como agente de la simulación.
   - La desaparición por muerte de este agente puede ser provocada por causas naturales como enfermedades o por edad y también aleatoria a causa de los otros depredadores del territorio que no modelamos.
4) *Otros requisitos*
   - Dado que el tamaño de patch puede ser cambiado para mejorar el rendimiento de la simulación, el tiempo que toman para moverse de patch ha de poder ser una constante ajustable para poder calibrar la simulación.

Una vez tenemos esta lista de requisitos pasamos al diseño del agente.

El agente no será un guanaco individual si no la manada dado que para el modelo no nos interesa una granularidad tan fina. Estas manadas pueden ser de dos tipos, manada o familia la diferencia entre las cuales reside en los miembros que conforman la agrupación. La familia contiene un solo macho y un número de crías igual a las hembras que la conforman y que es una constante definida en la definición del modelo, las manadas por otra parte son sólo formadas por machos en un número también definido en la definición del modelo.

Otras constantes del modelo son la probabilidad de supervivencia de los adultos y las crías y las calorías aportadas por un adulto, la cual es útil para futuras ampliaciones del modelo.

Las variables de memoria del agente son: si es familia o manada, sus coordenadas X e Y, las coordenadas objetivo X e Y las cuales serán explicadas posteriormente, el número de guanacos que conforman la manada y cuántos de ellos son adultos y, por último, la estación del año en la que se está.

Se define también en la descripción del modelo la unidad de tiempo que tarda el guanaco en recorrer un patch.

Los estados del agente y sus funciones de transición son:

Estado inicial a estado 1: si ha cambiado la estación entramos en la función en la que inicializamos variables y cambiamos la variable de estación de la memoria de los agentes a la nueva estación; si no, tan solo inicializamos las variables de memoria del agente.

Estado 1 a estado 2: si ha pasado el tiempo para que los guanacos se muevan un patch calcularemos este movimiento; si no, no hacemos nada.

Estado 2 a estado 3: si estamos en la fecha de apareamiento los guanacos se reproducirán; si no, no se hace nada.

Estado 3 a estado final: calculamos si algún guanaco muere.

Otras definiciones son las de los mensajes que se envia-

rán. Por el momento los guanacos se comunicarán únicamente con los agentes patch para indicarles cuántos entran o salen, o cuantos están dispuestos para la reproducción. Este mensaje será necesario para que el patch sepa cuántos guanacos tiene para comunicarlo a otros agentes o para crear nuevas manadas.

Antes de pasar a las implementaciones de las funciones de transición, mencionaremos las simplificaciones que hemos realizado en el modelo con tal de facilitar la implementación o hacer que la implementación tenga mejor rendimiento.

La época de apareamiento es sólo una vez al año, un sólo día que corresponde al inicio de cada año, esto es una vez cada 360 iteraciones. Los guanacos en un patch se reproducirán entre ellos por lo que desaparecerán todas las manadas y se reharán con los nuevos guanacos generados y los que habían anteriormente.

El movimiento de las manadas es realizado por todos los agentes a la vez en la fecha por defecto que hemos definido, la cual es cada 10 días o iteraciones. Este número es aceptable tomando en cuenta la velocidad media que puede tomar un guanaco cuando está en movimiento y dado que su rutina es comer, moverse y descansar.

Debido a la simplificación en la implementación de la reproducción, no se puede tener una variable de memoria en la que se mantenga una edad para los guanacos, por lo que la función de supervivencia se basa en la aleatoriedad con la constante que se ha definido. Esto no es problemático porqué no estamos modelando la vida de un guanaco sino de una manada y podemos asumir que cuando un guanaco adulto muere, muere el mayor de ellos.

Las implementaciones de las funciones de transición de los guanacos son:

Movimiento:
El movimiento en una versión inicial era un simple algoritmo que tomaba un número aleatorio mediante la función *rand()* de C y que tomaba en cuenta la estación del año en la que se estaba para definir hacia dónde se debe mover el agente, ya que si estamos en la estación seca, como hemos indicado anteriormente, el guanaco ha de acercarse a los ríos los cuales en nuestro modelo representan los límites de las coordenadas Y.

Dado que los patches han de saber cuantos guanacos contienen, este agente tendrá que pasar un mensaje tanto al agente patch que abandonan como al que entran.

El pseudocódigo para esta función de movimiento inicial es:

```
función move() devuelve: entero
        entero r → rand();
        entero estación → conseguir_estación();
        mensaje_salir_de_patch();
        si (estación = HÚMEDA) entonces:
                calcular_X(X, r);
                calcular_Y(Y, r);
        sino
                calcular_X(X, r);
                calcular_Y_estación_seca(Y);
        fin si
        mensaje_entrar_en_patch();
        devuelve 0;
fin función
```

La versión posterior para el movimiento usando Lévy flight será explicada en la sección 4.2.

El modelo de patch sufrirá cambios debido a esta función ya que ha de tener una función de transición en la que reciba los mensajes para sumar en su variable de memoria el nuevo número de guanacos que habrá en el patch.

Para el requisito de que los guanacos huyan de los clanes, el agente guanaco recibe información de los agentes patch que están alrededor cada vez que realizan un movimiento. Esta información es procesada por los guanacos para decidir el patch al que ir en última instancia. Si el patch al que van a acceder los guanacos está ocupado por un clan, estos deciden ir a otro patch vecino. Para realizar esto creamos una matriz de 9 posiciones que representa las posiciones a un paso alrededor de la situación actual del agente donde almacenamos la información enviada por mensajes de los agentes patch y si en caso de que nuestro movimiento nos lleva a un patch con clanes, mediante esta matriz podemos decidir una nueva posición de un modo rápido. En caso de que el agente esté completamente rodeado, no se cambiará el patch objetivo inicial.

Reproducción:
Como se ha mencionado anteriormente cuando el tiempo de reproducción llega, los agentes le pasan un mensaje al patch que almacena el número de guanacos adultos y de crías que han crecido hasta tornarse adultas. Tras esto, el agente desaparece de la simulación. Esto no significa que los guanacos desaparezcan, sino que solo desaparece el agente que luego el patch se encargará de volver a generar así como los nuevos agentes.

Así pues, el patch se encarga de contar los guanacos que hay en él, dar aleatoriamente un género a las crías y volver a formar las manadas generando un agente por cada manada formada; si la manada es una familia por que hay hembras en ella, se le asignará una cría por cada hembra.

Muerte:
La muerte en los guanacos se basa en la aleatoriedad. Se genera un número aleatorio que se traduce en base 100 para compararlo con la constante definida en el modelo.

4.2 Lévy flight

La primera implementación del movimiento de los guanacos es un paseo aleatorio; tras posterior investigación [10] descubrimos que el movimiento más cercano al que usualmente realizan los seres vivos cuando buscan algo es aleatorio pero sigue una una distribución un tanto peculiar en la que no siempre se realizan pasos cortos y cercanos buscando recursos sino que, con una cierta probabilidad, en vez de realizar un movimiento corto el sujeto realizará un salto largo, la cual se llama Lévy flight o vuelo

de Lévy. Un vuelo de Lévy es un paseo aleatorio de cola pesada, lo cual quiere decir que con una cierta probabilidad el movimiento que se va a realizar dejará de ser un paso corto para ser un salto a una zona más alejada. El nombre de esta distribución viene del matemático Paul Lévy.

Se decide utilizar esta implementación para el modelo porqué explica el patrón de movimiento de las sociedades cazador-recolector, así como es el patrón observado que siguen los guanacos en la búsqueda de alimento [10].

Nuestra implementación para este patrón de movimiento mantiene la base de la probabilidad de movimientos cortos combinados con movimientos largos pero ha sido simplificada. La simplificación dada al patrón consiste en que calculamos aleatoriamente un número el cual compararemos con nuestra probabilidad de que el movimiento sea largo, tras ello en vez de generar un movimiento largo, alejado de nuestro punto actual, generamos un punto objetivo aleatorio en todo el espacio de simulación. Para implementar esto de un modo sencillo creamos unas variables en los modelos que siguen este patrón de movimiento las cuales son las coordenadas X e Y objetivo.

Así pues, el primer cambio introducido al modelo de las manadas de guanacos fue añadir estas variables y sustituir el movimiento completamente aleatorio anteriormente implementado por este que comprueba si en esta iteración el movimiento ha de ser corto o largo y en cualquier caso decidir hacia donde se realiza.

Una consideración a tener en cuenta, incluso con este patrón, es el hecho de la estación del año, la cual ha de seguir afectando el movimiento de todos los agentes de la simulación. Por este motivo, las manadas de guanacos en estación seca bloquean sus coordenadas objetivo para acercarse a los ríos, donde realizan el mismo patrón pero sin alejarse de estos patches.

Nuestra implementación en pseudocódigo para esta función es:

```
función levyflight (entero: x, y, &targetX, &targetY)
                                        devuelve: entero
    entero r;
    si (x = targetX && y = targetY) entonces:
        r → modulo(rand(), 100);
        si (r  >= MOVIMIENTO_LARGO) entonces:
                calcular_movimiento_largo(targetX);
                calcular_movimiento_largo(targetY);
                devuelve 1;
        sino
                calcular_X(targetX, r);
                calcular_Y(targetY, r);
                devuelve 0;
        fin si
        sino
                devuelve 1;
        fin si
fin función
```

Esta función genera las coordenadas objetivo para el agente además de comprobar si estamos en un movimiento largo dado que devuelve 1 en caso de que hayamos generado una coordenada objetivo aleatoria o en caso de que no hayamos llegado aún a nuestra coordenada objetivo y 0 en caso de que estemos realizando un movimiento corto. Este valor de retorno de la función nos sirve para que en la función de transición podamos alterar alguna coordenada en caso de que la estación sea seca o en caso de que el agente quiera esquivar o entrar en un patch.

4.3 Movimiento de clanes

El movimiento de los clanes ha de cumplir ciertos requisitos que se mencionarán a continuación:

- Los clanes han de poder moverse de patch para poder explotar sus recursos.
- El agente clan ha de intentar estar en patches donde haya agentes del tipo manada de guanacos.
- Los patches han de saber cuantos clanes albergan.
- El tiempo que requiere a los clanes para cambiar de patch ha de ser una constante ajustable en el modelo.
- Durante la estación seca los clanes han de dirigirse a zonas húmedas, estas son tanto los ríos situados en los extremos norte y sur como la costa situada en los patches más orientales.

Una vez tenemos esta lista de requisitos, explicaremos las modificaciones que se han realizado a los agentes que nos venían dados para poder implementar esta funcionalidad.

En primer lugar, en el agente patch hemos añadido en la definición del modelo los mensajes que enviará al agente clan con el número de guanacos que contiene y la estación del año en al que la simulación se encuentra, así como funciones de transición en las que enviarán este mensaje a los clanes y otra función de transición en la que recibirá mensajes de los clanes que entren o salgan del patch.

En el modelo del agente clan hemos añadido las variables de memoria de agente de las coordenadas objetivo X e Y necesarias para implementar el movimiento siguiendo un patrón Lévy flight. Se ha añadido también el tiempo en el que los clanes se moverán de patch como una constante de tiempo que se ha definido en 45 días inicialmente, no obstante se puede modificar para calibrarla. Añadimos también la función de transición en la que se entrará cada vez que se llegue a la iteración correspondiente al tiempo de moverse del clan, así como el mensaje que el agente envía tanto al patch que se abandona como al patch al que se entra.

Cabe decir que, a diferencia del agente guanaco el cual mantiene una variable de memoria para la estación en la que se encuentra, el agente clan no, esto es debido a que el número de agentes manada de guanacos es muy superior al número de agentes clan y por tanto no nos influye apenas en el rendimiento mandar un mensaje con un valor más.

La implementación de la función de transición del mo-

vimiento de los clanes es muy parecida a la implementación de la misma función de los guanacos salvo ciertos aspectos que serán detallados a continuación:

El agente clan sigue a los agentes manada de guanacos cercanos, lo cual quiere decir que cada vez que llegamos a la fecha de movimiento los patch envían la información con los guanacos que contienen y esto será almacenado en una matriz igual que la que los guanacos tienen para evitar a los clanes. Si en estos patch alrededor del clan hay guanacos el movimiento se decide y realiza al patch que más guanacos contiene. En caso de que no haya ninguna manada de guanacos en las posiciones más cercanas al clan, este decidirá un movimiento Lévy flight. Este movimiento Lévy flight es igual al del agente manada de guanacos incluyendo la corrección de coordenadas objetivo durante la estación seca, salvo que en vez de dirigirnos a los ríos miramos si es más próximo un patch costero o no para realizar la corrección a este.

5 RESULTADOS

Con el propósito de generar resultados que podamos comparar hemos aplicado algunas modificaciones en el código que nos han permitido reducir la aleatoriedad en las ejecuciones dado que estamos buscando ver la mejora respecto a la versión secuencial y ver qué mejora aporta el uso de más procesadores. De este modo podemos correr ejecuciones de menos iteraciones ya que sigue un incremento lineal. Estas modificaciones y sus motivos son:

- Se ha eliminado la inicialización de la semilla de la función *rand()* de C. La aleatoriedad en las ejecuciones provocaba que el tiempo de ejecución pudiera variar de entre 3 minutos a agotar límites de tiempo de ejecución de 10 minutos para el mismo tamaño de problema.
- Se ha fijado el número de agentes manada de guanacos, impidiéndoles tanto desaparecer de la simulación así como añadir nuevos. Para realizar esto se ha modificado la constante que calcula su probabilidad de morir a que esta sea nula y se ha eliminado la instrucción que realiza la generación del nuevo agente durante la reproducción. No obstante ambos cálculos se llevan a cabo por lo que el resultado sigue siendo fiel al de una ejecución balanceada. El motivo de esta modificación es mantener estático el número de agentes del tipo manada de guanacos para que en nuestra simulación no desaparezcan todos o hayan demasiados.

Para llevar a cabo las ejecuciones hemos generado los archivos correspondientes a los agentes. El número de agentes utilizados en las simulaciones son 225 agentes patch, 3 agentes clan, 30 agentes individuo que se reparten entre los tres clanes y 17068 agentes manada de guanacos.

Para llevar a cabo las ejecuciones contamos con el cluster del Centro de Investigación Agrigenómica (CRAG). Este cluster nos ha prestado los nodos *Huberman*, *Batman*, *Catwoman*, *Robin* y *Penguin* para realizar nuestras ejecuciones. Las características de estos nodos son:

Huberman: DELL PowerEdge R720
- Memoria: 8x32GB RDIMM, 1333 MHz DDR3.
- Procesador: 2x Intel Xeon Processor E5-2650, 8 cores a 2.00 GHz y 20MB Cache cada uno.

Batman y Catwoman: DELL PowerEdge C6145
- Memoria: 16x4GB, 1600MHz DDR3.
- Procesador: 4x AMD Opteron Processor 6376, 16 cores a 2,3 GHz.

Robin: Supermicro SYS-5017GR-TF
- Memoria: 64GB ECC DIMM, 1600MHz DDR3.
- Procesador: Intel Xeon CPU E5-2630, 6 cores a 2,3 GHz.

Penguin:
- Memoria: 16x8GB, 1600MHz DDR3.
- Procesador: 4x Intel Xeon CPU E5-4620 0, 8 cores a 2,2 GHz.

El tiempo secuencial que nos da la ejecución de 1000 iteraciones del simulador en el nodo *Huberman* es de 49,157 segundos.

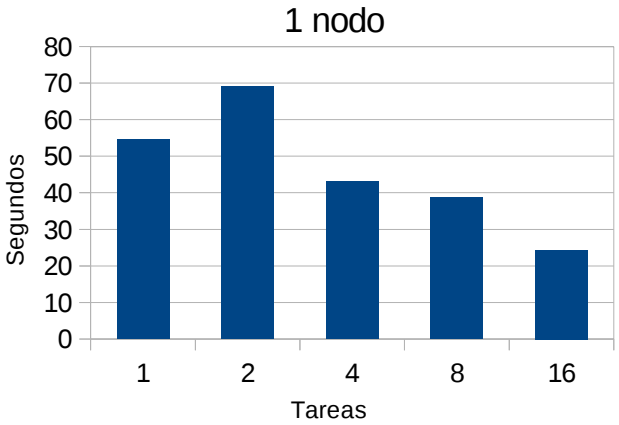

Fig. 2. Ejecuciones para 1000 iteraciones en el nodo Huberman.

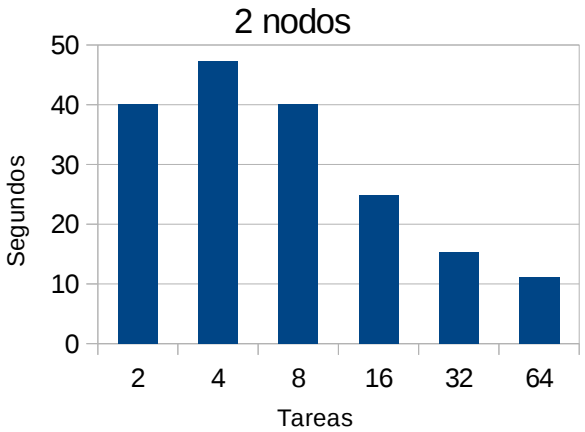

Fig. 3. Ejecuciones para 1000 iteraciones en los nodos Robin y Penguin

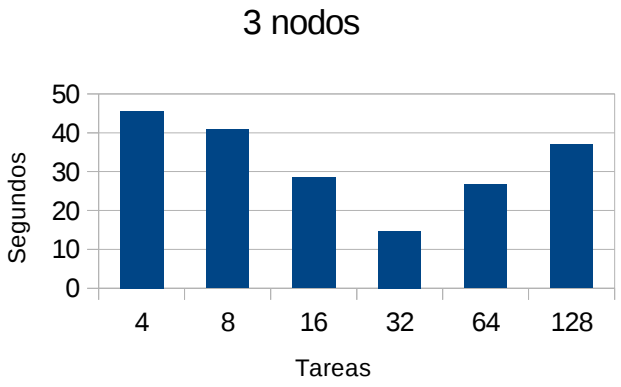

Fig. 4. Ejecuciones para 1000 iteraciones en los nodos Batman, Catwoman y Huberman

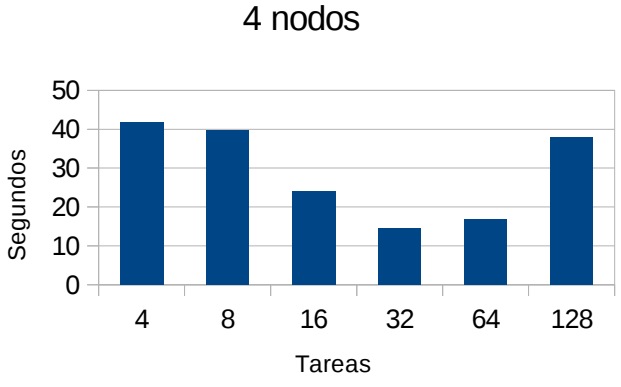

Fig. 5. Ejecuciones para 1000 iteraciones en los nodos Batman, Catwoman, Penguin y Robin.

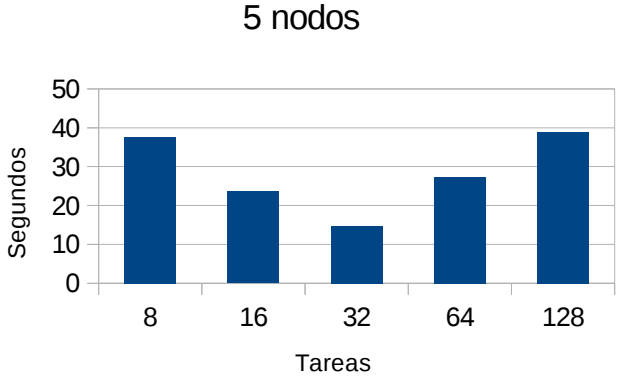

Fig. 6. Ejecuciones para 1000 iteraciones en los nodos Batman, Catwoman, Robin, Penguin y Huberman.

En la fig. 2 podemos observar en la ejecución de la simulación con un solo nodo que la división en tareas para una y dos tareas toma un tiempo de ejecución mayor que la ejecución secuencial, pero que para 4, 8 y 16 este es menor. Para las ejecuciones de la fig. 3, 4, 5 y 6 todos los tiempos son menores que el tiempo de la ejecución secuencial incluso para 4 tareas en 2 nodos.

Podemos observar en la fig. 3 que para dos nodos la separación en tareas disminuye nuestro tiempo de ejecución progresivamente mientras que en las fig. 4, 5 y 6 este aumenta a partir de 32 tareas.

6 Conclusiones

- FLAME es una herramienta potente y fácil de utilizar que nos permite realizar simulaciones paralelas basadas en agentes de un modo sencillo.
- Haber modelado funcionalidades de agentes que simulan ser seres vivos (como el movimiento o la reproducción) ha resultado complejo. Gran parte de esta complejidad reside en el hecho de que el comportamiento se puede simplificar, pero un acercamiento demasiado simple no aportaría gran cantidad de información mientras que un comportamiento muy detallado puede comprometer el rendimiento de la simulación, haciéndola insostenible en tiempo o recursos.
- El hecho del paso de mensajes para paralelizar la simulación añade complejidad dado que, volviendo al punto anterior, si los agentes necesitan más información de los otros agentes el paso de mensajes se eleva, comprometiendo la escalabilidad del modelo por lo que siempre hay que buscar una alternativa equilibrada entre los mensajes y la información necesaria para que la simulación de resultados interpretables.
- Hemos obtenido una mejora considerable en rendimiento al separar en tareas el trabajo realizado en varios nodos.
- No siempre poner más tareas y nodos nos mejora el rendimiento, esto se debe a que, en sus iteraciones de movimiento, todos los agentes manada de guanacos mandan dos mensajes lo cual provoca una caída de rendimiento dado a las comunicaciones.
- Se ha logrado un speed up de hasta 4.42 respecto de la versión secuencial en el mejor de los casos.

Líneas abiertas

- Se puede mejorar el rendimiento de la aplicación optimizando los algoritmos de movimiento, así como realizar las iteraciones de movimiento en momentos de un modo distribuido en varias iteraciones o reduciendo los agentes manadas de guanacos poniendo varias manadas en un solo agente.
- Puede mejorarse el modo en que los agentes nacen o mueren para que los agentes nazcan y mueran en una proporción aproximada con tal de que si ningún agente elimina manadas de guanacos el número de estos sea casi constante.
- Diseñar e implementar la caza en el modelo con este nuevo agente manada de guanacos. Llevar a cabo esta tarea puede resultar muy complejo;

dado que en sí ambas requieren de pasos de mensaje puesto que la comunicación de agentes es algo implícito en estas tareas.
- Investigar otros modelos de movimiento o mejorar el modelo de movimiento actual para que sea más próximo a la realidad.
- Puede mantenerse una memoria de recursos o un grado de atracción para decidir los movimientos de los agentes, basándose en algoritmos como el de la luciérnaga (firefly) [11].
- Una vez completado el simulador, implementar con herramientas existentes su visualización y análisis.

BIBLIOGRAFÍA

[1] Web de SimulPast. *The Emergence of Ethnicity in Hunter-Gatherer Societies. The case of Patagonia.* Visitado el 13 de Octubre del 2014. Disponible en http://www.simulpast.net/index.php/methodology/case-studies/38-cs2

[2] Web de Netlogo. *NetLogo User Manual*. Visitado el 28 de Enero del 2015. Disponible en https://ccl.northwestern.edu/netlogo/faq.html

[3] Hemalatha Vulchi (2014). *Agent Based Social Modelling with FLAME Simulation Framework* (tesis de master). Universitat Autònoma de Barcelona, Catalunya, España.

[4] Alfonso Imbernón. *Simulación Paralela Basada en Agentes de Sociedades Precolombinas: Relaciones entre Clanes.* Febrero de 2015, Pendiente de publicación.

[5] Artículo guanaco en Wikipedia. *Wikipedia: the free encyclopedia*. Consultado el 20 de Octubre del 2014. Disponible en: http://en.wikipedia.org/wiki/Guanaco

[6] Artículo Lévy flight en Wikipedia. *Wikipedia: the free encyclopedia*. Consultado el 3 de Diciembre de 2014. Disponible en: http://en.wikipedia.org/wiki/Lévy_flight

[7] Mariam Kiran, Paul Richmond, Mike Holcombe, Lee Shawn Chin, David Worth y Chris Greenough. *FLAME: Simulating Large Populations of Agents on Parallel Hardware Architectures*. 9Th International Conference on Autonomous Agents and Multiagent Systems, páginas 1633-1636, International Foundation for Autonomous Agents and Multiagent Systems, Toronto (2010).

[8] Web de Message Passage Interface (MPI). *The Message Passing Interface (MPI) standard*. Visitado el 16 de Octubre del 2014. Disponible en: http://www.mcs.anl.gov/research/projects/mpi/

[9] Web de FLAME. *FLAME Overview*. Visitado el 13 de Octubre del 2014. Disponible en: http://www.flame.ac.uk/docs/overview.html

[10] María Pereda García. *Random walks and Lévy flights: a social-science approach*. Presentado en SimulPast project el 25 de Septiembre, 2014 en IMF-CSIC, Barcelona, España.

[11] Donny Sutantyo, Paul Levi, Christoph Möslinger y Mark Read. *Collective-Adaptive Lévy Flight for Underwater Multi-Robot Exploration*. Mechatronics and Automation (ICMA), 2013 IEEE International Conference on 4-7 Agosto 2013, 456 – 462.