Simulación Distribuida de Modelo Orientados al Individuo utilizando MPI[1]

Diego Mostaccio, Remo Suppi, Emilio Luque

Departamento de Informática, Universidad Autónoma de Barcelona
Bellaterra, 08193, España
Diego.Mostaccio@aomail.uab.es, *Remo.Suppi@uab.es*, *Emilio.Luque@uab.es*

Resumen
El presente trabajo muestra los resultados obtenidos en el área de Simulación Distribuida de Eventos Discretos (PDES) para la solución de problemas basados en modelos orientados al individuo (IoM). Estos modelos son ampliamente utilizados en sistemas ecológicos y generan mejores resultados que los modelos convencionales pero, por otro lado, tienen el inconveniente que necesitan grandes capacidades de cómputo para la simulación de sistemas medianos o grandes (centenas o miles de individuos). La simulación distribuida puede ser utilizada como una herramienta útil para su simulación ya que permite emplear agrupaciones (clusters) de estaciones de trabajo de bajo costo como plataforma virtual de cómputo. Esto permite reducir el tiempo de simulación o, dado el incremento de potencia de cálculo, poder simular modelos más grandes y/o complejos. En el presente trabajo se analizan los conceptos teóricos involucrados, se describe el simulador diseñado y desarrollado bajo MPI (Message Passing Interface) y muestran los experimentos realizados para su validación y verificación.

Palabras clave: sistemas distribuidos, procesamiento paralelo, simulación distribuida de eventos discretos, modelos orientados al individuo

Trabajo para ser evaluado dentro del marco del V Workshop de Procesamiento Distribuido y Paralelo.

[1] El presente trabajo ha sido soportado por la CICYT bajo contrato TIC 2001-2592

1. Introducción

Uno de los principales inconvenientes en el uso de modelos orientados al individuo (IoM) es la necesidad de grandes capacidades de cómputo para su simulación. En este tipo de modelos se escoge al individuo como elemento básico a analizar y el ecosistema se describe por propiedades estáticas y dinámicas de cada uno de sus integrantes. Además, en estos ecosistemas el comportamiento de un individuo puede diferir del de otro individuo de la misma u otra especie.

Este tipo de modelo no puede ser resuelto en forma analítica y es necesario utilizar herramientas basadas en técnicas de simulación para obtener el comportamiento dinámico del conjunto. Para sistemas complejos, por ejemplo centenas o miles de individuos o integración de más de una especie en el mismo modelo, es necesario emplear técnicas de simulación avanzadas basadas en la paralelización del simulador o su distribución en agrupaciones de ordenadores (clusters) para brindar un respuesta eficiente.

La simulación de eventos paralela-distribuida –PDES- (*Parallel and Distributed Event Simulation*) es una herramienta útil (e indispensable en algunos casos) para proveer respuesta a problemas complejos en un tiempo aceptable. La principal ventaja de la PDES es poder reducir el tiempo de ejecución de un problema determinado o permitir analizar modelos de mayor tamaño (o con un mayor grado de detalle) ya que se incrementa la potencia de cálculo.

El presente trabajo demuestra el uso de PDES para resolver un tipo de modelo orientado al individuo denominado *Fish Schools*. En este tipo de modelos se analiza el comportamiento de movimientos de agrupaciones de peces. Este tipo de análisis permite a los biólogos estudiar las condiciones de supervivencia, reproducción y movilidad de una especie, teniendo en cuenta el hábitat, los depredadores y otros condicionantes que pueden afectar el comportamiento de la misma.

El trabajo se organiza de la siguiente forma: la próxima sección (2) se describen las principales características del modelo orientado al individuo. La sección 3, analiza las características del modelo de simulación y presenta el desarrollo de FishSchools como modelo de simulación de tipo conservadora en un sistema distribuido. En la sección 4 se describe el entorno experimental utilizado para verificar y validar las bondades de la técnica propuesta, analizando los resultados experimentales y extrayendo conclusiones sobre la misma. Por último en las secciones 5, 6 y 7 se analizan las conclusiones del presente trabajo, las líneas futuras del proyecto, los agradecimientos y las referencias respectivamente.

2. Modelos orientados al individuo -IoM- (Individual-oriented Models)

Un aspecto muy importante para los biólogos y ecologistas es la necesidad de modelar matemáticamente la dinámica de las poblaciones en ecosistemas con el objetivo de predecir su comportamiento. Estos ecosistemas son básicamente entornos de comportamiento realimentado, dinámicos y poseen mecanismos propios de auto-regulación comparables a otros sistemas físicos. Si bien el estudio realizado sobre estos es importante, no se han conseguido los mismos éxitos en su modelado y simulación que en otras disciplinas (control automático, redes de tráfico, etc.). Es por ello que en la bibliografía se proponen alternativas al enfoque clásico de los modelos orientado a población para su modelado y simulación: *modelos orientados al individuo*.

Los modelos más comunes para el desarrollo de sistemas ecológicos se basan en los trabajos de Lotka y Volterra [Smi74, Kre98, Lor95] que representan las interacciones entre una presa y un depredador. Estos modelos, que aún son vigentes, están basados en leyes de acción de masas y describen el comportamiento por medio de ecuaciones diferenciales y con los cuales se obtienen soluciones óptimas para sistemas muy predeterminados, de pequeño tamaño y no pueden ser generalizados para cualquier especie.

Las dos principales objeciones que presentan estos modelos son que la población se modela como cantidades de individuos uniformes (con comportamiento común a todos) y que es de tamaño considerable. Estas consideraciones son debido a que en las ecuaciones diferenciales se utilizan valores medios en los parámetros que caracterizan a una especie y por ello es necesario que la distribución de estos valores sea uniforme. Además, son modelos deterministas ya que para tener una solución es necesario resolver unas ecuaciones (de cierta complejidad) con unas condiciones iniciales (resolución analítica y/o numérica).

Existen diversas alternativas que mejoran los resultados como son los modelos estructurados por edades (*Age-structured models*) o los modelos estocásticos que permiten agregar coeficientes de incerteza en el comportamiento de la evolución a lo largo del tiempo.

Una solución aceptable, pero que difiere radicalmente en la concepción de la idea, son los modelos orientados al individuo (*Individual Oriented Models –IoM-)* los cuales se centran en el comportamiento del individuo y no en la comunidad de la cual forma parte (como se contempla en los modelos tradicionales). Estos modelos intentan modelar con detalle la naturaleza (se puede decir que son bio-inspirados) modelando el individuo a través de un conjunto de reglas simples (naturales) y observando la interacción entre ellos cuando forman parte de un ecosistema.

Dos de las ventajas importantes de estos modelos son: es independiente de la cantidad de individuos (en cuanto al modelo se refiere) y permiten simular ecosistemas muy complejos a partir de una base (individuo) muy simples sin necesidad de describir analíticamente la colectividad.

A diferencia de los modelos tradicionales en los cuales un incremento en el número de individuos implica ecuaciones diferenciales enormemente complejas y de difícil solución (o imposible), los IoM presentan muy buena escalabilidad y siempre es posible (con la potencia de cálculo adecuada) obtener soluciones en tiempos aceptables.

El principal problema de este tipo de modelo radica en los grandes requerimientos de potencia de cálculo que son necesarias para su simulación. En la actualidad, los recientes avances en la tecnología (cómputo y redes de comunicación), el desarrollo de lenguajes de programación orientados a objetos y la posibilidad de abstracción de las comunicaciones a través de librerías como PVM y MPI [Gei94, Cul99] han permitido dar respuesta a este tipo de simulaciones en tiempos aceptables.

2.1. Características de un IoM particular: Fish Schools

Una de las aplicaciones más representativas del IoM es la descripción del movimiento de determinadas especies la cual permite determinar el movimiento de un grupo usando el movimiento de cada miembro del mismo.

En este modelo se utilizan un conjunto de reglas (biológicamente definidas) muy simples que se aplican a cada individuo y con ellas se pueden obtener los movimientos de la colonia. En el caso de las *Fish Scools*, el movimiento del individuo (peces) está gobernado por tres postulados básicos desde el punto de vista de la supervivencia:

- Evitar colisiones (autoprotección).
- Acoplamiento de velocidades (velocidad similar a otros miembros del grupo para no separarse del mismo).
- Obtener una posición en el centro del grupo (para evitar la depredación).

Cada una de estas reglas expresan la necesidad de los individuos por sobrevivir y su instinto de protección (la necesidad de escapar de los depredadores).

Cada pez en el modelo se representado como un punto en el espacio tridimensional con un vector de velocidad asociado y cada individuo cambia de posición y velocidad simultáneamente después de un cierto período Δt. Las acciones que el modelo describe para cada pez son las siguientes:

- Cada pez escoge *X* peces vecinos (*X=4* es suficiente para la mayoría de las especies), los cuales serán los más cercanos y con visión directa. (ver figura 1).
- Cada pez reacciona de acuerdo a la dirección y distancia de cada vecino. Se establecen tres reacciones a partir de tres radios concéntricos (R_1,R_2,R_3):
    - Si el vecino se encuentra dentro del radio menor (R_1), el pez tendrá una reacción de repulsión tomando una dirección opuesta para evitar colisiones.
    - Si el vecino está en el segundo radio de influencia (R_2), el pez adoptará la misma dirección que el vecino y se moverá en el mismo sentido y dirección que él.
    - Si el vecino está en el tercer radio (R_3), el pez se moverá hacia él bajo un instinto de agrupación y protección del grupo.

En función de todas estas decisiones y considerando los *X* vecinos, cada pez calcula su nueva posición y velocidad.

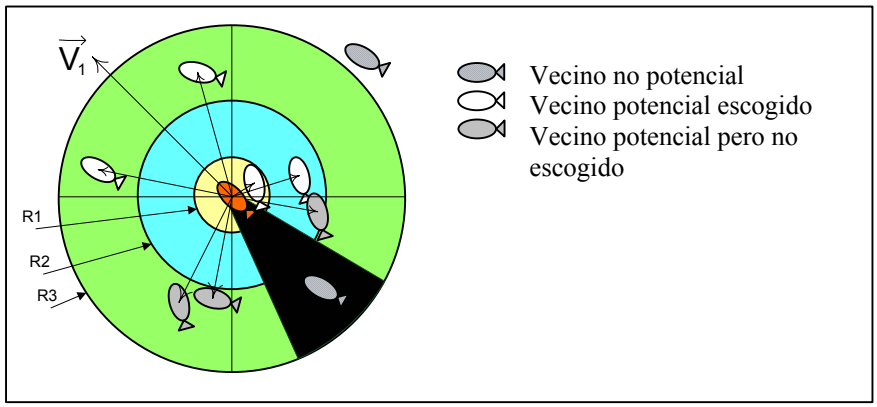

Figura 1. Selección de vecinos en el modelo Fish Schools

Esto genera un modelo muy simple pero permite describir comportamientos muy complejos. Como contraparte, la potencia de cálculo necesaria es muy elevada debido a que la complejidad del algoritmo es de $O(N^2)$, donde N es el número de peces (cada pez intenta buscar el pez vecino inspeccionando todos los otros peces en el sistema). [Lor95, Hus92]

En el modelo, cada pez F_i está definido por su posición $\boldsymbol{p_i}$ y su velocidad $\boldsymbol{v_i}$, y escoge sus potenciales vecinos observando zonas concéntricas de radio incremental hasta encontrar *X* peces como se muestra en la figura anterior. Para calcular la distancia entre dos peces, se utiliza la distancia Eucledania:

$$Dist(p_a, p_b) = \sqrt{(p_{ax} - p_{bx})^2 + (p_{ay} - p_{by})^2 + (p_{az} - p_{bz})^2}$$

Los vecinos potenciales son elegidos usando el algoritmo de *Front Priority*. Este algoritmo calcula todos los ángulos formados por $\boldsymbol{v_i}$ y $\boldsymbol{p_i} - \boldsymbol{p_j}$ (ángulo entre $\boldsymbol{p_i}$ y $\boldsymbol{p_j}$ en caso de colisión) y se seleccionan los *X* vecinos con el menor ángulo. La figura 2 muestra la selección de vecino de $\boldsymbol{p_1}$ usando el algoritmo *Front Priority* donde $\boldsymbol{p_2}$ es el vecino seleccionado dado que el ángulo $\boldsymbol{v_1}$-$\boldsymbol{v_{21}}$ ($\boldsymbol{v_2}$,($\boldsymbol{p_2}$-$\boldsymbol{p_1}$)) es menor que $\boldsymbol{v_1}$-$\boldsymbol{v_{31}}$ ($\boldsymbol{v3}$,($\boldsymbol{p3}$-$\boldsymbol{p_1}$)).

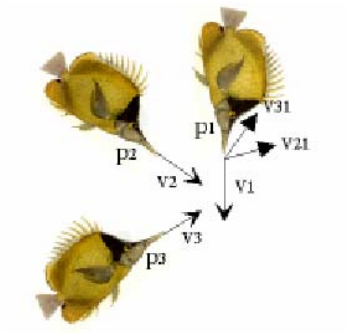

Figura 2. Selección de vecinos mediante el algoritmo Front Priority

Una vez que se han seleccionado los *X* vecinos, se debe determinar para el individuo (F_i) la reacción (rotación del vector $\boldsymbol{v_i}$) con respecto a cada uno de los vecinos (F_j). β_{ij} será la reacción compuesta entre el individuo bajo análisis y cada uno de sus vecinos expresada en coordenadas esféricas. Cada vecino puede ser buscado dentro de uno de tres posibles áreas de influencia (R_1,R_2,R_3):

- Si $Dist(F_j, F_i) \leq R_1$, F_i tiene una reacción de repulsión con respecto a F_j.
- Si $R_1 < Dist(F_j, F_i) \leq R_2$, F_i adopta una posición paralela con respecto a F_j.
- Si $R_2 < Dist(F_j, F_i) \leq R_3$, F_i es guiado hacia F_i.

Finalmente, se obtiene la reacción β como promedio de todos los β_{ij} y se aplica a la rotación del vector $\boldsymbol{v_i}$. La metodología de programación orientada a objetos y técnicas UML fueron utilizadas para la implementación del modelo y en la fase de análisis respectivamente. [Boo99]

3. Modelo de Simulación distribuida (PDES)

El desarrollo de la simulación se ha basado en la definición de un conjunto de procesos lógicos (LP) ubicados en una arquitectura de cómputo distribuida. Cada uno de estos procesos genera y comparte eventos (los cuales se modelan como mensajes en el sistema) con los restantes LPs; los mensajes que se intercambian los LPs incluyen información sobre el tipo de evento y el tiempo de ocurrencia del mismo.

Cada LP modela una sección del sistema a simular y debe intercambiar eventos (mensajes) con los otros LPs, simulando el cambio en las variables de estado en el sistema bajo estudio. Cada LP posee una lista de eventos propia pero que deberán incluir los eventos enviados por otros LPs cuando se modifique un variable de estado del mismo. Es por ello que es necesario mantener la coherencia del sistema global a través de mecanismos de control de la simulación que preserven la causalidad del sistema.

Dada esta necesidad de mantener la causalidad global, los mecanismos PDES pueden ser divididos en dos categorías: *conservativos* y *optimistas*. El método *conservativo* utiliza sincronización en cada LP para evitar errores de causalidad. En estos algoritmos, los eventos son procesados cuando se tiene seguridad que el orden de ejecución es correcto y por lo cual el sistema avanza (como máximo) a la velocidad de generación de eventos externos de LP más lento. Por otro lado, en algoritmos *optimistas*, cada LP procesa los eventos tan pronto como se encuentren disponibles y esta ejecución, en algunos casos, puede significar errores de causalidad pero el algoritmo incluye mecanismos de detección y recuperación de la causalidad. Estos algoritmos son más eficientes desde el punto de vista de simulación pero puede presentar, en determinados modelos, situaciones de inestabilidad por propagación de errores de causalidad además de necesitar más recursos locales para la detección y corrección de errores. [Ser08, Sup00]

3.1 Simulador distribuido y conservador: Diseño y desarrollo

El diseño y desarrollo del simulador conservador Fish Schools distribuido para modelos IoM se ha realizado utilizando las librerías de comunicación MPI. Esta decisión de diseño ha sido tomada en base la experiencia del grupo en temas de simulación de IoM bajo simuladores conservadores y optimistas pero basados en la librería de comunicaciones PVM. La utilización de PVM presenta ciertos problemas en la escalabilidad (tamaño de las colonias) y en la inestabilidad que se genera cuando el ecosistema adquiere dimensiones considerables (miles de individuos), problemas que sido los motivadores de la actual línea de trabajo.

Uno de los aspectos más importantes en un modelo de estas características es la asignación inicial de los individuos dentro de la arquitectura distribuida. Loret [Lor95] esboza una distribución de un modelo IoM asignando una partición estática del grupo de peces a simular en cada procesador y las consultas sobre los posibles vecinos se realizan sobre estructuras de datos centralizadas.

En nuestro caso, la solución adoptada se basa en la dividir el mundo real a simular en secciones, las cuales se asignan a procesos lógicos (LP) que podrán ser ejecutados en diferentes nodos de cómputo. Es decir, a cada LP se le asigna una partición inicial del problema (número de peces) y esta cantidad cambiará dinámicamente durante la simulación. Los LPs tienen una zona física del problema a simular (*spatially explicit simulation*) y el movimiento del pez entre secciones colindantes implicará migraciones de peces entre los LPs.

El simulador Fish Schools incluye dos tipos de mensajes entre LPs: petición de vecinos en una determinada zona y migración. Para reducir la cantidad de información entre los LPs se utiliza un mecanismo de petición de información punto a punto, en la cual sólo se consultan los vecinos de una zona específica de influencia (basadas en los radios de la figura 1). Por ejemplo, si el área simulada es un cubo, solamente los LPs que comparten las caras del cubo se comunicarán tanto para la petición de vecinos como para las migraciones.

El diseño del simulador Fish School propuesto, contempla un proceso *Father* que es quien inicializa los datos en los LPs. Los LPs pueden estar distribuidos uno en cada procesador o bien más de uno por procesador. En la figura 3 se observa un diagrama en bloques del diseño del simulador donde las líneas de comunicación entre el *Father* y los LPs sólo se mantienen en la inicialización mientras que las de comunicación entre los LPs son las utilizadas para el intercambio de eventos.

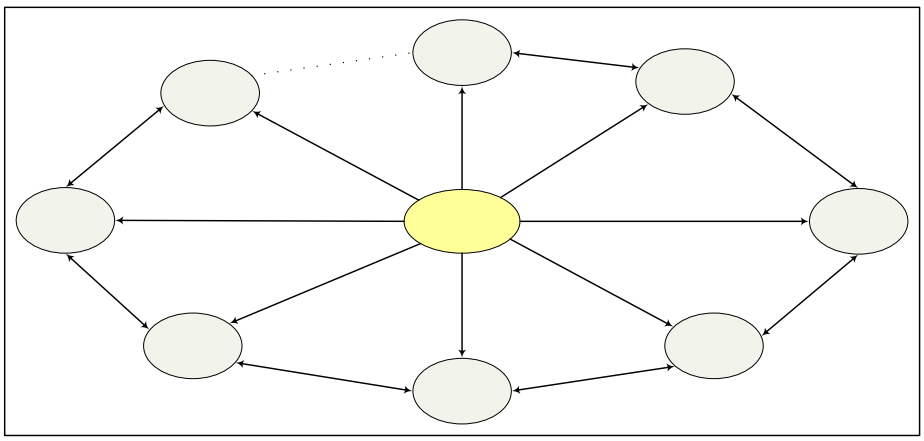

Figura 3.Diagrama en bloques del simulador

Durante el transcurso de la simulación se podrán producir interacciones entre los peces y los cuales pueden o no pertenecer al mismo bloque. En caso de no pertenecer el mismo bloque es necesario un mecanismo para el intercambio de información entre los LPs que en el diseño actual se realiza por medio de tres mensajes diferentes: *EvRequest, EvAnswer, EvMigration.*

Estos mensajes junto con los eventos internos (*NextStep, EvResume*) rigen el funcionamiento de la máquina de estados (figura 4). El funcionamiento de la máquina de estados es, en forma resumida, la siguiente: se inicia la simulación por el consumo del evento *EvNextStep* (el cual se inserta en la lista de eventos cuando se realiza la distribución de los peces en los distintos bloques). Una vez iniciada la simulación, se calculan las nuevas posiciones de los peces para lo cual es necesario disponer de la posición y la velocidad de los potenciales vecinos (que pueden encontrarse tanto en el propio bloque como en el contiguo). La petición de información al bloque vecino (si es necesario) se realiza mediante el envío de un mensaje *EvRequest* pasando posteriormente a un estado de espera de respuesta (*Wait For Answer*) las cuales serán generadas por los LPs receptores de los mensajes *EvRequest*.

Cuando se han recibido todas las respuestas (*EvAnswer*), se reanuda la simulación, a partir del punto donde había quedado, mediante el consumo del evento *EvRessumeStep.* A partir de ese instante, se procesa el evento *EvNextStep* que tiene como posibles resultados algunos de los siguientes eventos: migración de algún(os) individuo(s) (genera de *EvMigration*), cambios en las posiciones de los peces, o requerimiento (*EvRequest*) de posiciones y velocidades de los vecinos para la próxima iteración.

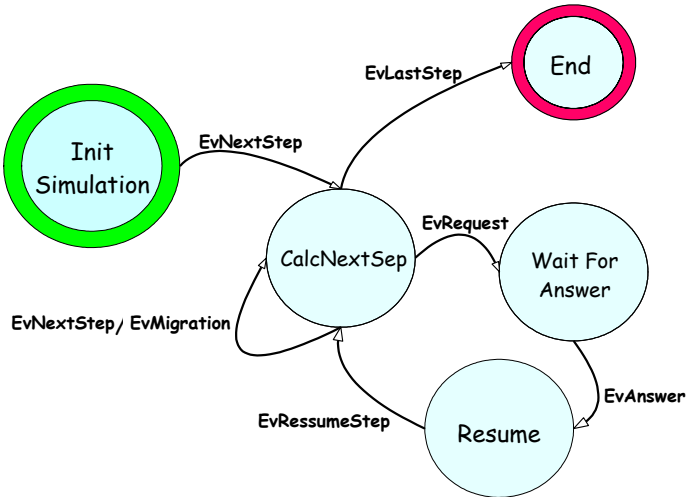

Figura 4. Máquina de estados del simulador distribuido

Para finalizar la simulación se debe generar el evento de *EvLastStep* sobre cada LP cuando en cada uno de ellos se alcance la condición de finalización de la simulación. A continuación se muestra (en pseudocódigo) la estructura básica del simulador:

```
While (Tiempo < Fin) {
        Ev = Extraer_Evento_Con_Menor_TimeStamp (ListaExternos, ListaInternos);
        Switch(Evento_tipo){
                Case EvResumeStep:
                Case EvNextStep:
                        Calcular_Posiciones_y_velocidades();
                        If (Buscar_vecinos)  Enviar_Evento(EvRequest); break;
                        If (Hay_Migraciones)  Enviar_Evento(EvMigration);
                        Incrementar_Tiempo()
                        Agregar_Evento_Interno(EvNextStep)
        Case EvRequest: Enviar_Evento(EvAnswer);

        Case EvAnswer: Agregar_Evento_Interno_(EvResumeStep);

                Case EvMigration: Agregar_Individuo_a_la_Región();
                }
        }
```

4. Estudio experimental

La verificación y validación del modelo y del simulador se han realizado mediante experimentos utilizando un cluster Beowulf bajo Linux SuSE 8 y con una red de interconexión Fast Ethernet.

Para el análisis de escalabilidad y prestaciones se ha generado un simulador serie que implementa el mismo modelo pero su funcionamiento es secuencial (una única lista de eventos) y se ejecuta sobre un monoprocesador. Es necesario destacar que este no es el simulador distribuido ejecutándose sobre un único nodo sino un simulador secuencial que implementa el modelo FishSchools. Como medida de prestaciones se ha seleccionado el tiempo de ventana (frame) –en segundos- que implica el tiempo consumido por el simulador para generar las nuevas posiciones y velocidades de todos los peces del ecosistema bajo estudio.

Las pruebas se han realizado en forma paramétrica, variando la cantidad de peces entre 100 y 25000 y la cantidad de ordenadores del cluster entre 1 y 32 (la densidad de peces por bloque y el territorio de simulación se ha mantenido constante a lo largo de los experimentos).

Como primer paso en la experimentación se ha obtenido el tiempo de *frame* [segundos] utilizando la versión serie del simulador para una población variable (100- 25000) y los resultados se pueden observar en la figura 5 (estos valores son los que posteriormente se han utilizado para realizar la comparación de prestaciones con el simulador distribuido).

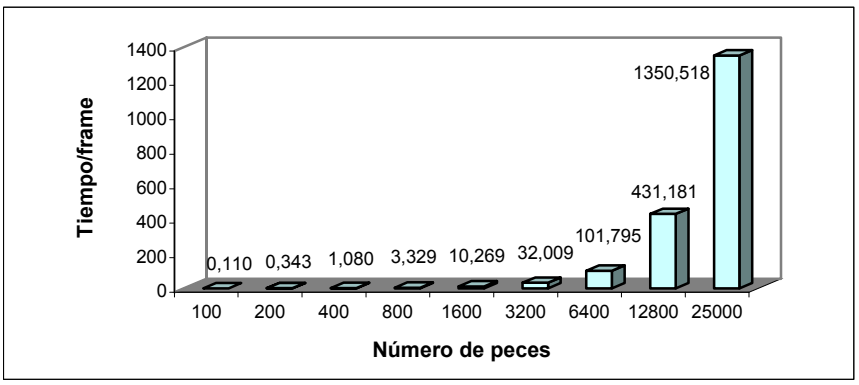

Figura 5. Tiempos de frame para el simulador serie

Como puede observarse, el simulador serie sólo podría hacer una animación en tiempo real aceptable (≈10 *frames*/seg.) para 100 peces mientras que para 400 peces se tendría velocidades aproximadas a un *frame*/segundo.

En el segundo paso de los experimentos, se han realizado las mediciones sobre el simulador distribuido variando la cantidad de peces, (los mismos valores que en el simulador serie) y cantidad de procesadores (de 2 a 32).

En la figura 6 se puede observar la diferencia de tiempo de *frame* existente entre el simulador serie y el distribuido la cual disminuye conforme aumenta el número de procesadores.

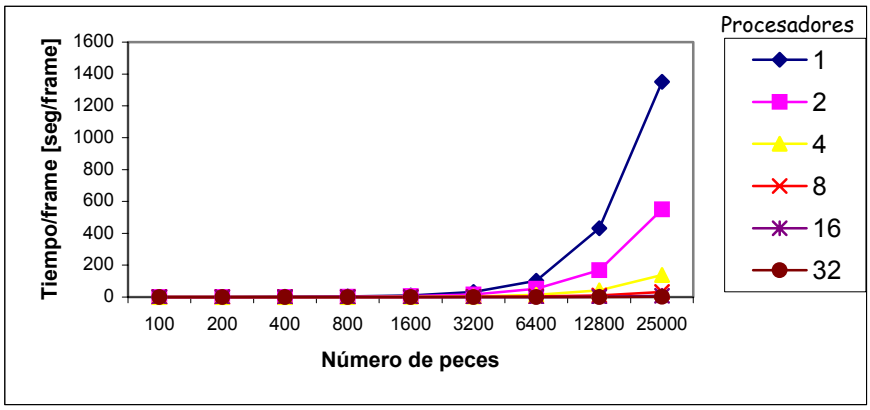

Figura 6. Tiempos de frame para el simulador distribuido

Como comparación entre el simulador distribuido y el simulador serie, se ha calculado el aumento de prestaciones (speedup) los cuales se muestran en la figura 7. Como puede observarse los valores máximos teóricos (linealidad) son superados notoriamente y esto se debe a que la complejidad disminuye con la división del territorio de simulación ya que la cantidad de peces a los cuales se debe consultar para procesar la nueva posición es menor. En el caso del simulador serie se tenia una complejidad de $O(N^2)$ (donde N es el número de peces) mientras que con una división del espacio de simulación en ***n*** LPs, se tiene una complejidad de $O\left(N^2/n\right)$.

Para realizar estas medidas no se ha incluido el tiempo de bloqueo del algoritmo conservador de simulación en la espera de las respuestas de las comunicaciones (tiempo de *barrier*) ya que lo que interesaba era obtener el límite superior del *speedup* posible. La figura 8 muestra los resultados si se contemplan las comunicaciones. Como se puede apreciar, los tiempos de *frame* han aumentado considerablemente, en relación a la figura anterior, pero aún se obtienen mejoras (a partir de determinado número de peces) con respecto al simulador serie.

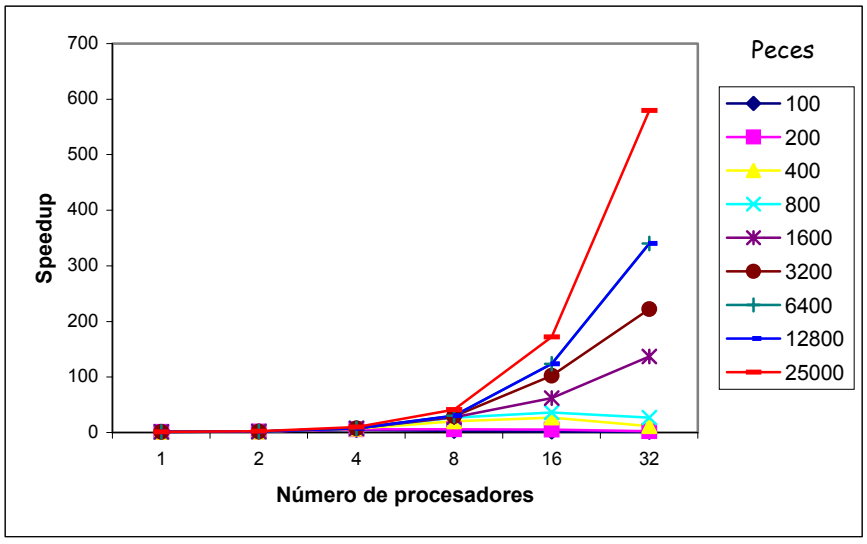

Figura 7. Relación de prestaciones (speedup)

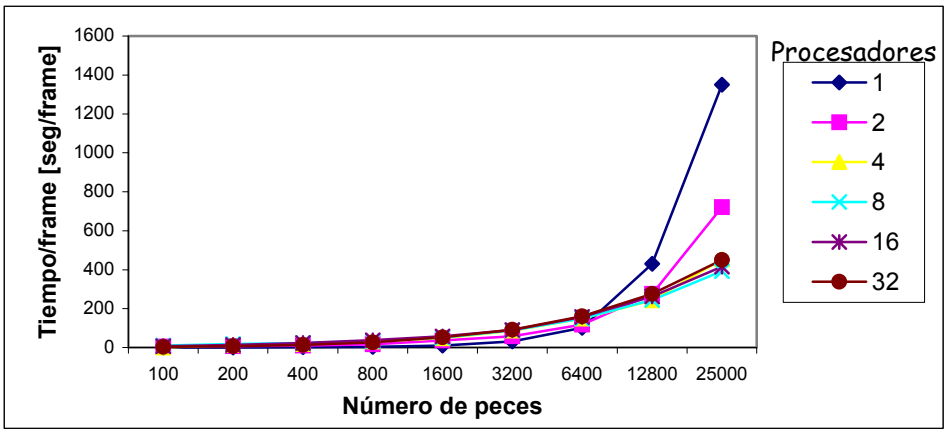

Figura 8. Tiempo de frame incluyendo tiempo de comunicaciones

Es necesario tener en cuenta que para colonias menores a 6400 peces, el tiempo de *frame* del simulador serie es menor que el del simulador distribuido (independientemente del número de procesadores) .

La máxima diferencia de tiempo se consigue para una cantidad de 8 procesadores y 25000 peces. En estas condiciones, el tiempo necesario para realizar una simulación de 50 *frames* se reduce de 18h 45min a 5h 27min como se puede concluir de la figura 9. Se puede observar que en el caso de 2 procesadores y una población de 25000 peces, el *speedup* se encuentra muy cercano al ideal (1,86).

Para analizar la influencia de las comunicaciones y el control de la causalidad del algoritmo conservador, se ha instrumentado el simulador de manera tal que se pueda discriminar la cantidad de tiempo utilizado en comunicaciones y en procesamiento. La figura 10 muestra los resultados obtenidos para la simulación utilizando 8 procesadores para colonias de 6400, 12800 y 25000 peces (zona delimitada por la líneas de puntos en la figura 9). De la gráfica se deduce que el efecto de las comunicaciones sobre el tiempo de simulación es un factor predominante y se debe a los mecanismos de sincronización que es necesario incluir en el algoritmo conservador para mantener la causalidad.

La figura 11 muestra dos imágenes (con y sin texturas) de la interfase gráfica para análisis post-mortem de los resultados de simulación desarrollada por el grupo la cual se ha desarrollado en Linux y OpenGL. [Fra02]

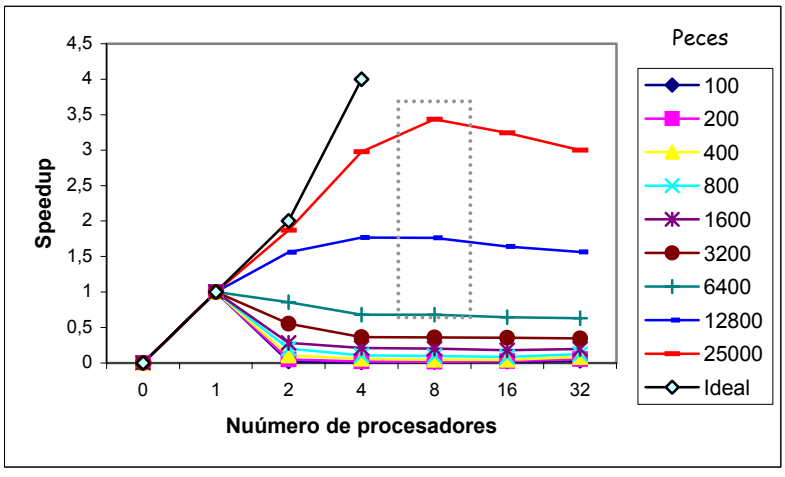

Figura 9. Speedup contemplando las comunicaciones

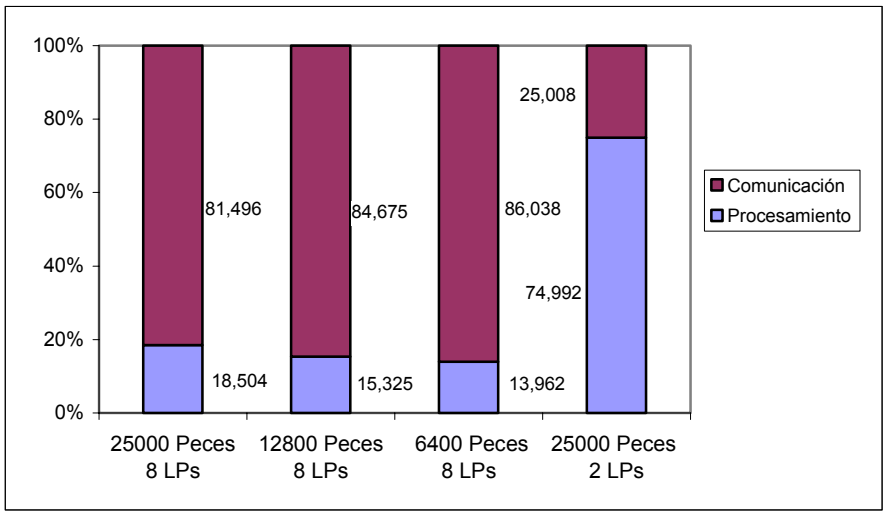

Figura 10. Relación cómputo-comunicación (algoritmo conservador)

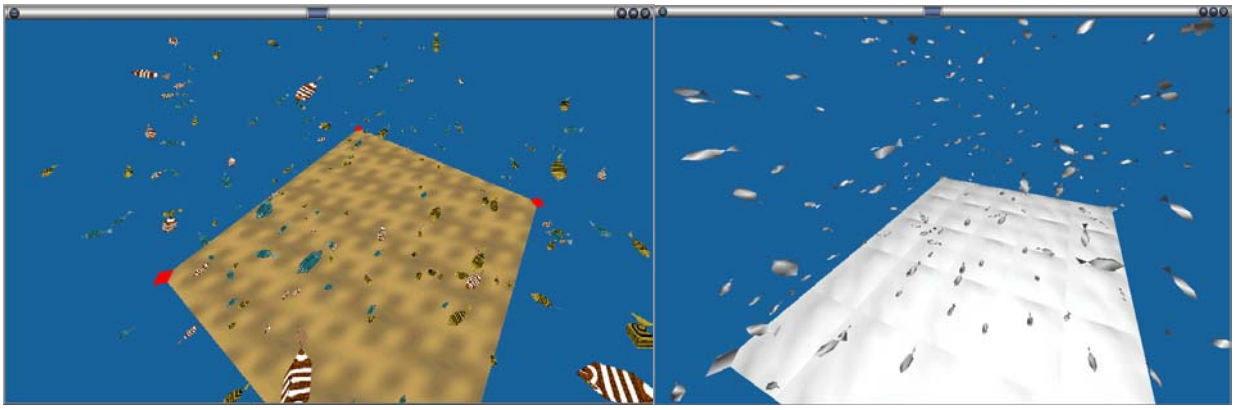

Figura 11. Inferfase gráfica de los resultados del simulador Fish Schools

5. Conlcusiones

La simulación de sistemas ecológicos es un campo que requiere grandes potencias cálculo cuando se utilizan modelos orientados al individuo y la utilización de sistemas de cómputo distribuido es una excelente herramienta para resolver este tipo de problemas.

El presente trabajo analiza la simulación distribuida orientada a eventos en modelos orientados al individuo (concretamente al movimiento de peces) utilizando un algoritmo de simulación de tipo conservador. Por los resultados obtenidos se demuestra que es una opción viable pero que los algoritmos conservadores imponen una fuerte limitación a las potencialidades del método.

Para la comparación de prestaciones se han realizado experimentos en forma paramétrica sobre un simulador distribuido y los resultados se han comparado con los obtenidos en un simulador secuencial (serie) creado especialmente para tal fin. Sin contemplar las esperas en las comunicaciones se han obtenido un aumento de prestaciones (*speeudps)* mucho más allá de la linealidad para grandes cantidades de peces, mientras que si se tienen en cuenta dichas esperas el aumento de prestaciones adquiere valores menores pero muy aceptables.

Por último se han mostrado las ventanas de una animación post-mortem realizadas por una aplicación gráfica desarrollada para visualizar el comportamiento de la simulación basada en OpenGL y ejecutándose sobre Linux.

Las líneas de trabajo futuro se centran en:

- Optimización del algoritmo conservador de simulación con el fin de reducir el impacto de las comunicaciones para mejorar el rendimiento.
- Analizar las posibilidades de los algoritmos optimistas controlado las cadenas de errores mediante mecanismos que permitan controlar el grado de optimismo.
- Generar un entorno de simulación integrado en forma tal que se pueda ejecutar las simulaciones en forma interactiva sobre el cluster visualizando por otro lado las animaciones en tiempo real.
- Incorporar al modelo Fish Schools los efectos de depredadores, obstáculos, mezclas con otras especies, control de la energía y envejecimientos del individuo para que pueda servir a las diferentes necesidades de análisis por parte de biólogos y ecologistas.

6. Agradecimientos

Los autores agradecen la colaboración en el presente trabajo de las personas que han aportado esfuerzo, trabajo y dedicación para que este proyecto fuera posible: **Daniel Ruiz** y **Jordi Valls** en el soporte técnico del cluster, **Daniel Fernández Francos** en el desarrollo de la aplicación gráfica para el análisis por animación post-mortem de los resultados de la simulación y **Pere Munt** por su contribución a la simulación distribuida con modelos orientados al individuo bajo PVM.

7. Bibliografía

[Boo99] Booch, G., Rumbaugh. J., Jacobson, I. *The Unified Modeling Language User Guide*, Adisson-Wesley, 1999.

[Cul94] Culler, D., Pal Singh, J., Gupta, A., *Parallel Computer Architecture: A Hardware/Software Approach,* Morgan Kaufmann, 1999.

[Fra02] Francos, D., *Aplicaciones de simulación y animación distribuidas utilizando cluster de Linux, PVM, OpenGL.* Trabajo de Graduación dirigido por R. Suppi. Universidad Autónoma de Barcelona. España, 2002

[Gei94] Geist, A., Beguelin, A., Dongarra, J, Jiang, W., Manchek, R., Sunderman, V. *Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing.* The Mit Press, 1994.

[Hus92] Husth, A., Wissel, C. *The simulation of movement of fish schools*, Journal of Theoretical Biology, 156, 365:385, 1992.

[Kre98] Kreft, J. Booth, G, Wimpenny, W. T. J. *BacSim, a simulator for individual-based modelling of bacterial colony growth*, Microbiology Journal, 144, pages 3275-3287, 1998.

[Lor95] Lorel, H, Sonnenschein, M., *Using Parallel Computers to simulate individual oriented models: a case study*, Proceedings of the 1995 European Simulation Multiconference (ESM), pages 526-531, June 1995.

[Ser98] Serrano, M., Suppi, R., Luque, E. *Parallel Discrete Event Simulation, State of art anb bibliography.* Unitat d'Arquitectura d'Ordinadors i Sistemes Operatius, departament d'Informàtica, Universitat Autònoma de Barcelona. http://pirdi.uab.es

[Smi74] Smith, M. J. *Models in Ecology*, Cambridge University Press, 1974.

[Sup00] Suppi, R., Cores, F, Luque, E., *Improving Optimistic PDES in PVM Environments*, Lecture Notes in Computer Science ISSN 0302-9743, Springer-Verlag, Vol. 2329(1), pages 107-116, 2002.