Universitat Autònoma
de Barcelona

Escola d'Enginyeria
Departament d'Arquitectura de Computadors i Sistemes Operatius

Particionamiento y Balance de Carga en Simulaciones Distribuidas de Bancos de Peces

Tesis doctoral presentada por Roberto Solar Gallardo para optar al grado de Doctor por la Universitat Autònoma de Barcelona, bajo la dirección del Dr. Remo Suppi Boldrito.

Bellaterra, 31 de mayo de 2012

Particionamiento y balance de carga en simulaciones distribuidas de bancos de peces

Tesis doctoral presentada por Roberto Solar Gallardo para optar al grado de Doctor por la Universitat Autònoma de Barcelona. Trabajo realizado en el Departament d Arquitectura de Computadors i Sistemes Operatius de la Escola d Enginyeria de la Universitat Autònoma de Barcelona, dentro del Programa de Computación de Altas Prestaciones , bajo la dirección del Dr. Remo Suppi Boldrito.

Bellaterra, 31 de mayo de 2012

Director

Dr. Remo Suppi Boldrito

Agradecimientos

A mi hijo Benjamín, cuya aparición en este mundo ha iluminado inmensamente mi vida, te amo mucho.

A mi mujer *Cynthia*, por haberme acompanado incondicionalmente durante este difícil camino y por haberme entregado el regalo más significativo dentro de mi existencia en este planeta. También quisiera agradecer a las integrantes de mi familia, que aunque estén geográficamente alejadas de mi siempre las llevaré dentro de mi corazón: a mi *Abuelita Ana*, que me ha sido un apoyo fundamental durante el desarrollo de mi vida; a mi madre *María Cristina*, por haberme dado la vida y haber confiado en mi; a mi hermana *Ana Karina*, porque siempre me ha comprendido a pesar de mi complejidad y a mi hermana *María Paz*, a la cual he observado desde pequena hasta verla convertirse en la mujer que es ahora. *Las amo mucho a todas.*

También quisiera agradecer a mi director de tesis el Dr. Remo Suppi por haberme dado la posibilidad de desarrollar este trabajo con completa libertad y al Dr. Emilio Luque por haberme dado la oportunidad de formarme como doctor. Finalmente, quisiera agradecer a todos los integrantes del *Departamento de Computadores y Sistemas Operativos* de la *Universidad Autònoma de Barcelona*, que durante esto últimos cuatro anos han sido un aporte primordial para mi desarrollo profesional.

"El individuo ha luchado siempre para no ser absorbido por la tribu. Si lo intentas, a menudo estarás solo, y a veces asustado. Pero ningún precio es demasiado alto por el privilegio de ser uno mismo."

Friedrich Nietzsche

Resumen

El particionamiento y balance de carga son aspectos de gran interés en simulaciones distribuidas basadas en modelos orientados al individuo espacialmente explícitos. La descomposición equitativa del dominio del problema y la distribución eficiente de las particiones sobre las unidades de cómputo de la arquitectura paralela/distribuida son factores cruciales cuando se plantea como objetivo obtener el mejor rendimiento de la aplicación de simulación distribuida. Además, en modelos que exhiben patrones de movimiento es necesario establecer políticas competentes de balance de carga dinámico para continuar satisfaciendo el objetivo de rendimiento a medida que la simulación evoluciona en función del tiempo. En el presente trabajo se han desarrollado métodos de particionamiento y balance de carga para simulaciones distribuidas a gran escala de modelos orientados al individuo espacialmente explícitos que exhiben patrones de movimiento. Se ha utilizado el modelo de *Huth & Wissel*, el cual representa el movimiento coordinado y polarizado de bancos de peces, para validar las estrategias desarrolladas. El método de particionamiento consiste en descomponer el dominio del problema en particiones compactas generadas a partir del *criterio del radio cobertor* y *diagramas de Voronoi*. La distribución de las particiones se realiza por medio de la agrupación de particiones por proximidad en un conjunto *meta-particiones* de cardinalidad igual a la cantidad de unidades de cómputo. La estrategia de balance de carga dinámica consiste en detectar el desbalance por medio de un *algoritmo basado en umbrales* y re-configurar las *meta-particiones* para lograr el re-equilibrio. Finalmente, se ha realizado experimentación exhaustiva con motivo de validar y verificar la viabilidad del simulador distribuido en distintos escenarios.

Prólogo

Durante los últimos anos, la simulación de sistemas complejos cuyos procesos internos independientemente generan comportamientos globales emergentes a partir de la interacción, han despertado gran interés en múltiples ámbitos de aplicación, tales como: *física*, *química*, *biología*, *sociología*, etc. En este tipo de sistemas, la interacción autónoma entre componentes de bajo nivel provoca un flujo de información a través del sistema produciendo patrones colectivos. El modelado orientado al individuo es un paradigma computacional que permite describir forma matemática y/o lógica las relaciones entre componentes. Este tipo de modelos típicamente consisten en: un *ambiente* el cual constituye la ubicación lógica en la cual los individuos residen, actúan e interactúan, y un conjunto de *individuos* definidos en término de sus atributos y reglas de comportamiento. Debido a la gran complejidad que involucra resolver este tipo de modelos analíticamente, es necesaria la utilización de herramientas de simulación computacional para obtener un comportamiento dinámico del sistema.

Los mayores inconvenientes que han acarreado las simulaciones de estos tipos de modelos en arquitecturas secuenciales han sido: 1) *el nivel de complejidad de los modelos*: a medida que los modelos se acercan a la realidad es necesario un gran poder de cómputo para poder evaluar las interacciones individuo-individuo e individuo-ambiente, y 2) *la carga de trabajo asignada a una única unidad de cómputo*: las simulaciones a gran escala requieren un gran espacio de memoria para poder almacenar los componentes de la simulación. La simulación distribuida es una tecnología que ofrece los recursos de *hardware* y *software* para posibilitar la ejecución de simulaciones computacionales de sistemas de alta complejidad y a gran escala reduciendo el costo en términos de tiempo de ejecución que involucra la ejecución en una arquitectura secuencial. Los principales problemas de ejecutar una simulación computacional en múltiples unidades de cómputo son: el *particionamiento*, descomponer eficientemente el dominio del problema y el *balance de carga*, asignar de forma equitativa las particiones sobre las unidades de cómputo de la arquitectura paralela/distribuida (balance de carga estático), y mantener este equilibrio a medida que la simulación progresa en función del tiempo (balance de carga dinámico).

En el presente trabajo se desarrollan estrategias de *particionamiento* y *balance de carga* para simulaciones distribuidas basadas en modelos orientados al individuo espacialmente explícitos que exhiben patrones de movimiento, en particular para simulaciones basadas en el modelo de formación de bancos de peces de *Huth & Wissel*. Por un lado, se propone un método de particionamiento de grano fino basado en *particiones compactas*. Las particiones compactas consisten en delimitar áreas a partir de un elemento de referencia llamado *centro* o *centroide*. La selección de centroides se realiza por medio del *criterio del radio cobertor*, lo

que permite obtener un conjunto de centroides lo suficientemente alejados unos de otros. La delimitación de las áreas se realiza por medio de *diagramas de Voronoi*, lo que permite generar un conjunto áreas de similar tamano y forma. El balance de carga estático (asignación), se realiza por medio de la creación de un conjunto de *meta-particiones*. Cada *meta-partición* es formada a partir del criterio de proximidad entre particiones. La cardinalidad del conjunto de *meta-particiones* es igual a la cantidad de unidades de cómputo de la arquitectura paralela/distribuida. Por otro lado, se establece una política de balance de carga dinámico, que consiste en dinámicamente re-construir las *meta-particiones* a medida que la simulación progresa en función del tiempo, con el objetivo de mantener el equilibrio en la carga de trabajo asignada a las unidades de cómputo. El presente trabajo consiste en los siguiente capítulos:

Capítulo 1: En este capítulo se hará una breve introducción en la que se expondrá una visión global del trabajo realizado en esta investigación. Se comentarán los objetivos imprescindibles para lograr la finalidad global de este trabajo. También se hará una revisión de los trabajos relacionados con esta investigación.

Capítulo 2: En este capítulo se realizará una visión general de los enfoques con los cuales se puede modelar la dinámica de poblaciones. Las herramientas principalmente utilizadas que permiten modelar la dinámica de poblaciones son: el *modelado basado en ecuaciones* y el *modelado orientado al individuo*. Se describirán en detalle ambas aproximaciones y se ejemplificarán con algunos modelos conocidos.

Capítulo 3: En este capítulo se procederá a describir los mecanismos de simulación computacional y simulación distribuida. Por un lado, se describirán los aspectos relevantes y los componentes importantes en la simulación de eventos discretos, y, por otro lado, se revisarán los mecanismos necesarios para ejecutar una simulación computacional en múltiples unidades de cómputo.

Capítulo 4: En este capítulo se hará una revisión de los métodos de particionamiento más conocidos. En particular, dividiremos los métodos de particionamiento en dos conjuntos: *grid-based* - descomposición del espacio de simulación y *cluster-based* - descomposición por agrupación de individuos. También se hará un repaso de los método de balance de carga *estáticos* y *dinámico*, incluyendo la detección del desbalance.

Capítulo 5: En este capítulo se describirán los mecanismos utilizados para la implementación de un simulador distribuido basado en el modelo de *Huth & Wissel*. Primero, se describirá como se ha adaptado el modelo de *Huth &*

Wissel para representar el movimiento de un banco de peces en un ambiente tridimensional. Segundo, se describirá el método de particionamiento y la implementación del algoritmo secuencial de simulación. Finalmente, se describirán en detalle los procedimientos desarrollados para la ejecución de la simulación en una arquitectura distribuida/paralela.

Capítulo 6: En este capítulo se realizará un estudio exhaustivo del desempeno del simulador distribuido desarrollado en el presente trabajo. Se realizará la ejecución del simulador en múltiples escenarios y se hará una evaluación de la ganancia obtenida en términos de *speedup* para la simulación distribuida. Además, se analizará el ajuste de la carga de trabajo realizado por el mecanismo de balance de carga dinámico.

Capítulo 7: Finalmente, se presentarán las conclusiones obtenidas, los trabajos propuestos para el futuro y se expondrán los artículos publicados durante el desarrollo de la presente investigación.

Índice general

Índice de figuras

Índice de algoritmos

Capítulo 1

Introducción

1.1. Antecedentes

Los comportamientos colectivos en grupos de individuos autónomos son un fenómeno común existente en diferentes escalas y niveles de complejidad. En este tipo de sistemas, los fenómenos emergentes surgen a partir de decisiones descentralizadas ejercidas por componentes de bajo nivel del sistema. Estos componentes actúan a partir de cierta información local proveniente de la interacción con otros componentes y el entorno en el cual residen, provocando un flujo de información a través del sistema y, como consecuencia, la aparición de patrones colectivos. Si estos patrones colectivos se despliegan a partir de un fenómeno de organización interna carente de influencias de entidades externas, el proceso es denominado *auto-organización*. Los comportamientos colectivos y la *auto-organización* son fenómenos comunes observados en sistemas complejos existentes en una amplia gama de áreas de investigación, tales como: *física* [68, 14], *química* [24] [50] [66], *sociología* [26, 25], *biología* [28, 29] [55] [1] [54] [30], etc. En biología, este tipo de agregación social es vinculada a distintos términos dependiendo de la especie a la que se refiera, como por ejemplo: *banco* o *cardumen* de peces, *enjambre* de insectos, *manada* de mamíferos, *bandada* de pájaros, etc.

La formación de bancos de peces es un ejemplo pragmático de orden espontáneo que surge a través un proceso de organización interna. Esta agrupación social muestra propiedades emergentes complejas, como por ejemplo: una fuerte *cohesión* de grupo, lo que significa que se mantiene la formación de grupo a través del tiempo y un alto nivel de *sincronización*, lo que significa que los peces mantienen nadando hacia la misma dirección y con la misma rapidez. Además es interesante observar que en este tipo de agregaciones la navegación se realiza en ausencia de líderes o estructura jerárquica. Una de las principales motivaciones que incita a los peces a pertenecer a grupos de muchos individuos es el establecimiento de mecanismos de *anti-depredación*, entre los cuales se pueden destacar: *efecto de*

dilución [67] - pertenecer a un grupo de muchos individuos similares garantiza anonimato y decrementa la posibilidad de ser capturado, *efecto de confusión* [45] - el movimiento coordinado de muchos individuos similares tiende a confundir la capacidad del depredador de enfocarse en un único individuo y capturarlo, *hipótesis de los muchos ojos* [57, 56] - pertenecer a un grupo de muchos individuos incrementa la probabilidad de detectar a un depredador.

Existen dos enfoques con los cuales se puede modelar la dinámica de poblaciones: *modelado basado en ecuaciones* y *modelado orientado al individuo*. Los modelos basados en ecuaciones representan al sistema por medio de un conjunto de variables de estado y su evolución consiste en resolver un conjunto de ecuaciones (generalmente ecuaciones diferenciales). Dependiendo la cantidad de variables independientes incluidas en el modelo, se pueden clasificar en: *ecuaciones diferenciales ordinarias*, en las cuales solamente interviene una única variable independiente, usualmente el *tiempo* (ej. las ecuaciones de *Lotka-Volterra* [39, 72], también conocidas como ecuaciones *presa-depredador*), y *ecuaciones diferenciales en derivadas parciales*, en las cuales es posible incluir más de una variable independiente: *tiempo*, *espacio*, etc (ej. modelos de *invasión* [62] utilizadas para representar la expansión de poblaciones de ratas almizcleras en Europa).

En los modelos orientados al individuo el sistema es descompuesto en dos componentes de bajo nivel: un conjunto de *individuos* los cuales están definidos en término de sus atributos y comportamientos, y un *entorno* o *framework* que representa la ubicación lógica en donde los individuos residen, actúan e interactúan. El modelado orientado al individuo ofrece una gran cantidad de ventajas que permiten experimentar hipotéticamente con sistemas complejos desde una perspectiva más cercana a la realidad, como por ejemplo: modelar las interacciones *individuo-individuo* e *individuo-entorno*, incluir distintas clases de individuos dentro del mismo modelo, representar la heterogeneidad existente entre individuos de la misma clase, describir detalladamente el entorno en el cual los individuos se desenvuelven, establecer mecanismos de aprendizaje, capturar el fenómeno emergente, etc

Algunos modelos orientados al individuo pueden ser espacialmente explícitos, es decir, los individuos están asociados a una posición en el espacio geométrico. Algunos modelos orientados al individuo espacialmente explícitos pueden presentar patrones de movimiento, es decir, los individuos pueden cambiar su ubicación en el espacio geométrico a medida que el tiempo avanza

Sin embargo, es muy difícil resolver analíticamente un modelo orientado al individuo, por lo cual es necesario utilizar herramientas de simulación computacional para obtener una visión global de como las variables de sistema evolucionan a través del tiempo.

La simulación computacional es una herramienta que permite observar dinámicamente la evolución de sistemas complejos a partir de la implementación

de un modelo predefinido. Estos tipo de sistemas se caracterizan por ser de complejidad irreducible, es decir, sistemas compuestos por múltiples unidades que interactúan coordinadamente para desempenar alguna función. Resolver este tipo de modelos analíticamente supone la evaluación exhaustiva de todas unidades que componen el sistema, lo que implica un elevado costo en términos de tiempo de reloj. La simulación computacional permite reducir significativamente el tiempo de resolución de este tipo de problemas por medio de la ejecución del modelo sobre una plataforma computacional. Las grandes ventajas que ofrece la simulación computacional se pueden resumir en: permite experimentar con un sistema sin necesidad de afectar directamente al sistema en cuestión, ofrece resultados con una gran capacidad predictiva, rápida obtención de resultados, permite analizar un sistema en una amplia gama de escenarios, etc.

La implementación de una simulación computacional (simulador) se lleva a cabo por medio de la codificación del modelo mediante la utilización un lenguaje de alto nivel (*C++*, *Java*, etc). Posteriormente, se ejecuta el simulador sobre una plataforma computacional y se recolectan los resultados entregados. Finalmente, se analizan los resultados obtenidos con el objetivo de realizar un proceso de validación y verificación (tanto del modelo como del simulador). Este proceso se repite hasta que los resultados entregados por la simulación satisfagan las expectativas.

La simulación de modelos orientados al individuo ofrece la capacidad de replicar el comportamiento de sistemas complejos con mayor precisión. También permite incluir una infinidad de variables independientes (cada individuo está compuesto por un conjunto de variables independientes), incluir interacción *individuo-individuo* e *individuo-entorno*, incluir variables estocásticas, obtener una visión global de como las variables sistema evolucionan a partir de la interacción entre componentes de bajo nivel, etc.

Durante los últimos anos, la simulación de modelos orientados al individuo en ambientes secuenciales se ha visto principalmente influenciada por dos causas que afectan directamente al rendimiento de la aplicación (en términos de tiempo de ejecución):

El nivel de complejidad de los modelos: La necesidad de obtener resultados más cercanos a la realidad obliga a reducir el nivel de abstracción de los modelos. Esto se traduce en un incremento del nivel de detalle del modelo, por lo que se requiere más capacidad de cómputo para evaluar escenarios más complejos.

La carga de trabajo: A medida que el objetivo de la simulación apunta a replicar el comportamiento de sistemas reales, es necesario incluir dentro de la simulación un número de individuos que satisfaga las condiciones del sistema. Esto se traduce a incluir miles, cientos de miles y hasta millones de

individuos, por lo que se requiere un gran espacio de memoria para poder almacenarlos.

La aparición de arquitecturas de computación paralela/distribuida ofrece los recursos de hardware para resolver problemas de gran magnitud. Una arquitectura paralela/distribuida está compuesta por un conjunto de unidades de cómputo que trabajan simultáneamente para resolver un problema en particular. Las arquitecturas comúnmente implementadas se pueden clasificar en: *computadores paralelos de memoria compartida* los cuales están conformados por un conjunto de unidades de cómputo que comparten memoria por medio de un bus, *computadores paralelos de memoria distribuida* los cuales consisten en conjunto de unidades de cómputo con memoria independiente conectadas a través de una red de interconexión, o *computadores paralelos híbridos* en los cuales existe jerarquía de memoria, es decir, memoria compartida intranodo y memoria distribuida internodo. En la actualidad, la gran mayoría de las arquitecturas paralelas/distribuidas se caracterizan por ser sistemas híbridos.

La simulación distribuida se refiere a la tecnología que envuelve la ejecución de simulaciones computacionales sobre sistemas computacionales formados por múltiples unidades de cómputo. El programa de simulación es descompuesto en un conjunto de procesos independientes que pueden ejecutarse concurrentemente. Cada uno de estos procesos es denominado *proceso lógico*, y está formado por una porción disjunta de las variables de estado del programa de simulación. Debido a la dependencia de datos entre procesos lógicos es necesario implementar algún mecanismo de sincronización de procesos para mantener la consistencia de la simulación.

La simulación distribuida ha despertado gran interés en una infinidad de áreas de investigación debido a su versatilidad y flexibilidad. Las principales ventajas que ofrece la simulación distribuida son:

Ejecución más rápida: Entrega la posibilidad de reducir el tiempo de ejecución de la simulación por medio de su ejecución en múltiples unidades de cómputo.

Escenarios más grandes: Ofrece la posibilidad de trabajar con grandes volúmenes de datos. Una arquitectura paralela/distribuida posee un gran espacio de memoria que posibilita el almacenamiento de tantas variables de estado como sean necesarias.

Integración de simuladores: Permite integrar aplicaciones de simulación que se encuentran geográficamente apartadas.

Integración de modelos heterogéneos: Permite integrar distintos tipos de modelos dentro de la misma simulación.

La simulación distribuida ofrece una serie de beneficios en términos de rendimiento, pero la implementación distribuida de una simulación computacional basada en un modelo orientado al individuo es un problema complejo que acarrea una serie de inconvenientes que pueden afectar el rendimiento global de la aplicación

Particionamiento

El particionamiento de un sistema orientado al individuo consiste en descomponer el dominio del problema en un conjunto de porciones que pueden ejecutase concurrentemente con el objetivo de explotar el paralelismo que ofrece la arquitectura paralela/distribuida. La cantidad de porciones debe ser mayor o igual a la cantidad de unidades de cómputo que posea la arquitectura paralela/distribuida. Existen dos enfoques con los cuales se puede descomponer el dominio de sistemas orientados al individuo: *grid-based* (descomposición espacial) o *cluster-based* (descomposición por agrupación). Los métodos basados en descomposición espacial dividen el espacio de simulación en un conjunto de regiones que contienen a los individuos que residen actualmente en cada una de ellas. Los métodos basados en descomposición por agrupación consisten en formar grupos de individuos asociados por algún criterio. El criterio de agrupación que se seleccione depende de la naturaleza del sistema. Se pueden utilizar criterios de agrupación tales como: proximidad, similitud, familiaridad, etc.

Asignación

El proceso asignación consiste en distribuir el conjunto de particiones obtenidas en el proceso de descomposición del dominio del problema sobre las unidades de cómputo de la arquitectura paralela/distribuida. Este es un problema complejo en el que intervienen múltiples variables y en diferentes niveles. Los principales variables que se deberían tener en cuenta al momento de desarrollar un algoritmo de asignación son: a *nivel de arquitectura* : topología y latencia de la red de interconexión, velocidad de procesamiento de las unidades de cómputo, tamano de la memoria, etc, y a *nivel de aplicación*: patrones de comunicación, granularidad de las particiones, etc. Una buena estrategia de asignación debería cumplir los siguientes objetivos:

- Minimizar la comunicación entre los procesos lógicos asignados unidades de cómputo de la arquitectura paralela/distribuida.
- Equilibrar la carga de trabajo asignada a las unidades de cómputo de la arquitectura paralela/distribuida, es decir, cumplir el objetivo de balance de carga estático.

- Escalabilidad, permitir que el rendimiento de la simulación distribuida se mantenga a medida que se incrementa la cantidad de unidades de cómputo.

El asignación es una tarea fundamental en el desarrollo de algoritmos de paralelos/distribuidos, ya que su implementación afecta directamente al rendimiento global de la aplicación paralela/distribuida.

Balance de carga dinámico

El principal objetivo del balance de carga dinámico es que cada unidad de cómputo posea una parte equitativa del trabajo total a medida que el tiempo avanza. En aplicaciones en las cuales los datos migran desde un proceso a otro a medida que la ejecución del programa progresa, es necesario establecer una política eficiente de balance de carga dinámico. El primer paso es detectar cuando un desbalance ocurre. Luego, es necesario evaluar la situación para decidir que camino tomar. Por un lado, se puede balancear la carga de trabajo migrando datos desde una unidad de cómputo a otra. Si la opción anterior no es factible, será necesario re-particionar y re-asignar el dominio de problema.

1.2. Trabajos relacionados

A continuación se realizará una revisión de los trabajos que se enmarcan dentro del área de investigación del presente trabajo. Esta recopilación representa ejemplos de implementación de modelos orientados a individuo en arquitecturas paralelas/distribuidas. El principal interés se centra en describir los métodos de particionamiento, asignación y balance de carga dinámico (si es que existe).

Debido a que la simulación distribuida de modelos orientados al individuo representa la solución a múltiples problemas en diversos campos de investigación, se dividirán los trabajos relacionados según el ámbito de estudio.

Ámbito de la física:

En [76] se propone un método basado en *micro-celdas* el cual consiste en descomponer el espacio de simulación en pequenas áreas en forma de caja. Estas *micro-celdas* son agrupadas en conjuntos de *micro-celdas* contiguas las cuales son distribuidas sobre la arquitectura paralela/distribuida. El algoritmo de balance de carga consiste en ajustar la carga por medio del intercambio de micro-celdas adyacentes. El artículo estudia simulaciones basadas en partículas.

En [46] se propone un método particionamiento para problemas de simulación de *N-body* utilizando agrupamiento por medio de *k-means*. También introducen heurísticas de balance de carga por medio de escalado iterativo del tamano de los grupos y redistribución adaptativa de los centroides de los grupos.

Ámbito de la biología:

En [63] se realiza de cómo la inclusión de estímulos externos sobre el modelo de bancos de peces de *Huth & Wissel* influye en la escalabilidad de la simulación distribuida. El método de particionamiento implementado consiste en dividir el espacio de simulación en franjas rectangulares. Cada franja es asignada a un recurso de cómputo. La mayor ventaja de este tipo de particionamiento es que la comunicación se realizar sólo con los procesos lógicos vecinos. No se implementan políticas de balance de carga.

En [64] se propone un método de particionamiento basado en el *criterio de radio cobertor* y *diagramas de voronoi*. Este trabajo constituye la etapa preliminar de la investigación propuesta en el presente trabajo. El método consiste en seleccionar un conjunto de centroides por medio del criterio del radio cobertor. Posteriormente, se crea el diagrama de Voronoi asignando el resto de los individuos al centroide más cercano. Los individuos son almacenados en una *lista de clusters*. El criterio de distribución consiste en agrupar los *clusters* por proximidad con el objetivo de crear un conjunto de *meta-clusters*. Cada meta-clusters es asignado a una única unidad de cómputo. En [65] se propone un método de balance de carga basado en proximidad. El método consiste en re-configuración de los *meta-clusters*. No hay paso de mensajes en el proceso de balance de carga, ya que el proceso de migración de individuos es el encargado de realizarlo.

En [17] se implementa el modelo de bancos de peces sobre una arquitectura *GPU* por medio de *CUDA*. El método de particionamiento consiste en dividir el espacio de simulación en celdas cúbicas de igual tamano. Cada celda tienen un único ID y dada la posición de un individuo su *hash* es el ID de la celda.

En [36] se realiza otra implementación del modelo de bancos de peces sobre una arquitectura *GPU*. No hay métodos de particionamiento implementados.

En [77] se realiza una implementación distribuida del modelo de *boids* de *Craig Reynolds*. El método de particionamiento consiste en dividir el espacio de simulación por medio de franjas verticales paralelas al eje y. Cada unidad de cómputo mantiene una única partición. Debido a que los *boids* pueden migrar de una unidad de cómputo a otra, se implementa un mecanismo de balance de carga. La política de balance de carga consiste en re-particionar y re-distribuir el espacio de simulación. Cuando un desbalance ocurre, se recalcula y actualiza la información de la frontera, luego se re-distribuyen los *boids*.

En [10] se realiza una implementación distribuida del modelo de *boids* de *Craig Reynolds*. El método de particionamiento consiste en realizar cortes paralelos a uno de los ejes coordenados (franjas). Cada una de estas franjas es asignada a una unidad de cómputo . El mecanismo de balance de carga consiste en ajustar dinámicamente las fronteras cuando existe un desbalance.

Ámbito de la microbiología:

En [38] se realiza una implementación distribuida del modelo de difusión de partículas. El modelo de difusión consiste en migraciones aleatorias de moléculas o pequenas partículas. El espacio de simulación de particionado en franjas horizontales y cada franja es asignada a una unidad de cómputo. No existen mecanismos de balance de carga implementados.

En [22] se implementa un simulador distribuido para representar el comportamiento de grandes colonias de bacterias.

Ámbito de la sociología:

En [69] se realiza un estudio del comportamiento de multitudes de peatones. El método de particionamiento implementado consiste en dividir el espacio de simulación en regiones de forma irregular. El método consiste en administrar particiones como las envolventes convexas de los puntos que representan las ubicaciones de los agentes en una región en particular. No existen estrategias de balance de carga implementadas.

En [70] se realiza un estudio comparativo entre métodos de particionamiento para multitudes de peatones. En este trabajo se comparan métodos de particionamiento: *rectangular* y el método basado en envolventes convexas propuesto. No existen mecanismos de balance de carga implementados.

En [71] se implementan mecanismos de balance de carga para simulaciones distribuidas de multitudes de personas. El método de particionamiento es descrito en [69, 71]. El mecanismo de balance de carga consiste en migrar agentes desde recursos sobrecargados hasta recursos infrautilizados. Es importante mencionar que la migración de agentes se realiza entre envolventes convexas vecinas.

En [74] se realiza la implementación de un método de particionamiento basado en una adaptación de *k-means*. El artículo estudia el comportamiento a gran escala de multitudes de peatones. El método de particionamiento consiste en agrupar agentes por medio de su posición. Inicialmente, se seleccionan k agentes(centroides) de forma aleatoria y se asignan el resto de los agentes a su centroide más cercano. Finalmente, hay que iterar *k-means* hasta su convergencia.

En [58] se realiza la implementación distribuida del modelo de fuerzas sociales. El método de particionamiento consiste en dividir el espacio de simulación de regiones rectangulares regulares. Cada región es asignada a una unidad de cómputo. Para que la simulación sea consistente, se implementa un mecanismo de celdas fantasma [15] con el objetivo de evaluar la influencia de los peatones que se encuentran ubicados en unidades de cómputo vecinas. No hay mecanismos de balance de carga implementados.

1.3. Objetivos

El objetivo principal del presente trabajo consiste en establecer una alternativa de solución para simulaciones distribuidas a gran escala de modelos orientados al individuo espacialmente explícitos que exhiben patrones de movimiento. Este tipo de simulaciones se caracterizan por su enorme nivel de complejidad por lo que necesitan una gran capacidad de cómputo y grandes espacios de memoria. Además, debido a sus requerimientos computacionales, su ejecución en un entorno secuencial es casi imposible (debido a los elevados tiempos de simulación), por lo que la solución adecuada es la utilización de herramientas de simulación distribuida.

Para lograr el objetivo principal fehacientemente es necesario definir un conjunto de pequenas etapas cuya evolución satisfactoria implica directamente en el cumplimiento de la finalidad de la presente investigación. Los objetivos específicos definidos para el presente trabajo se pueden resumir en:

- Reducir la complejidad del algoritmo de simulación secuencial de $O(n^2)$ a $O(mk)$ donde $m, k < n$. La gran mayoría de los algoritmos de simulación requiere cómputo exhaustivo de todos contra todos con el objetivo de encontrar los vecinos más influyentes.

- Establecer un mecanismo de particionamiento robusto que permitan eliminar evaluaciones espacio-temporales entre individuos que se encuentran demasiado lejos. Para esto, se implementará un método de particionamiento de grano fino consistente en un conjunto micro-particiones las cuales pueden ser intersectadas unas con otras con el objetivos de poder descartar al momento de ejecutar el proceso de búsqueda de vecinos.

- Asignar particiones eficientemente sobre la arquitectura. Debido a que, generalmente, se obtendrán más particiones que unidades de cómputo, se debe establecer un mecanismo eficiente de asignación con el objetivo de reducir el tiempo de ejecución global de la aplicación de simulación distribuida. Se implementará un mecanismo de agrupación de particiones en meta-particiones las cuales serán asociadas por proximidad.

- Establecer mecanismos de balance de carga. Las aplicaciones de simulación distribuidas de modelos orientados al individuo espacialmente explícitos que exhiben patrones de movimiento presentan serios problemas al momento de su ejecución en entornos paralelos/distribuidos. Estos problemas son principalmente causados por la migración de individuos entre procesos lógicos, produciendo la sobrecarga o la infrautilización de recursos de procesamiento.

1.4. Aportaciones del presente trabajo

El principal aporte del presente trabajo consiste en establecer mecanismos eficientes de particionamiento, asignación y balance de carga dinámico para simulaciones distribuidas de bancos de peces. Para poder mejorar el rendimiento de la aplicación distribuida, es necesario mejorar el rendimiento del algoritmo secuencial. Esto ayuda a que la aplicación alcance un *speedup* cercano al lineal, en vez de uno *superlineal*. Esto se produce debido a que el método de particionamiento desarrollado en el presente trabajo posibilita evitar la evaluación exhaustiva de individuos al momento de buscar los vecinos más influyentes. Para esto, se utilizan métodos como los observados algoritmos de *búsqueda por similitud* que permiten descartar a los individuos que se encuentran ubicados demasiado lejos. El problema de encontrar a los vecinos más influyentes se reduce a un problema *KNN* (k-nearest neighbor).

1.5. Organización de la tesis

El presente trabajo está organizado de la siguiente forma: En el capítulo 2 se hará una revisión de los diferentes enfoques utilizados para modelar la dinámica de poblaciones y se hará una breve descripción de los modelos orientados al individuo más relevantes. En el capítulo 3 se hará énfasis en los métodos y técnicas para la ejecución de simulaciones computacionales, tanto en ambientes secuenciales como en paralelos/distribuidos. En el capítulo 4 se realizará una breve descripción de los métodos de particionamiento más conocidos. También se abarcará la descripción de algunos algoritmos de balance de carga estáticos y dinámicos. En capítulo 5 se describirá la implementación del simulador desarrollado para el presente trabajo. Esto abarca tanto como el método de particionamiento, la asignación de particiones a los recursos de cómputo y el balance de carga dinámico. En el capítulo 6 se exhibirán los resultados obtenidos por la aplicación de simulación distribuida desarrollada en el presente trabajo. Además, se hará un estudio exhaustivo con el objetivo de encontrar los mejores parámetros para obtener el mejor rendimiento de la simulación. En el capítulo 7 se presentarán las conclusiones y el trabajo futuro propuesto para el presente trabajo.

Capítulo 2

Modelado de la dinámica de poblaciones

El estudio de la dinámica de poblaciones es un área que ha despertado gran interés en diversos ámbitos académicos. La dinámica de poblaciones ha sido el objetivo fundamental de la *biología matemática* o *biomatemática*, que es la disciplina encargada de desarrollar modelos matemáticos para describir procesos biológicos. También ha tenido un fuerte impacto en la *ecología demográfica* o *demoecología*, que es la disciplina encargada de estudiar la distribución espacial y temporal de los organismos de una misma especie.

La dinámica de poblaciones tiene como principal objetivo analizar y cuantificar las variaciones que sufren ciertas poblaciones a través del tiempo. Una población está compuesta por un conjunto de organismos de la misma especie, que residen en un lugar determinado en el que se desenvuelven. Existen propiedades tales como: las tasas de natalidad y mortalidad, distribución espacial, densidad, etc, que permiten estudiar el comportamiento de una población.

El modelado de la dinámica de poblaciones es un proceso de simplificación el cual consiste en seleccionar un conjunto acotado de características y factores fundamentales del sistema, y definir las relaciones entre componentes suficientes para describir cómo el funciona sistema. Desde un punto de vista global, se puede ver a un sistema como un *todo* compuesto por un conjunto de componentes independientes guiados por un único propósito, en el que este *todo* es mayor que la suma de todos sus componentes (sinergia).

En ecología, el desarrollo de modelos es un proceso de dificultad elevada debido a la complejidad, heterogeneidad y diversidad de los sistemas naturales. Cada sistema es único y difiere de otros sistemas en función de sus características y estructura interna. El diseno de modelos conceptuales en ecología desempena un rol fundamental para comprender cuantitativamente la interoperabilidad inherente existente en sistemas de complejidad irreducible.

Existen dos enfoques con los cuales se puede modelar la dinámica de poblaciones: *modelado basado en ecuaciones* y *modelado orientado a individuos*. En los modelos basados en ecuaciones, las propiedades del sistema son representadas por medio de variables de estado, el sistema se describe por medio de un conjunto de ecuaciones y su evolución a través del tiempo se realiza resolviendo dichas ecuaciones. En los modelos orientados a individuos, el sistema es descompuesto en dos componentes de bajo nivel: *individuos* - un conjunto de entidades autónomas definidas en término de sus atributos y reglas de comportamiento, y *ambiente* - la ubicación lógica en donde los individuos residen, actúan e interactúan. Las propiedades del sistema emergen a partir de la interacción entre individuos y/o el ambiente.

A continuación se revisarán en detalle los distintos enfoques utilizados para modelar la dinámica de poblaciones, abarcando sus principales características.

2.1. Modelado basado en ecuaciones

Los modelos basados en ecuaciones (MbE) capturan las características del sistema por medio de un conjunto de variables de estado y lo describen su funcionamiento por medio de un conjunto de ecuaciones relacionadas con estas variables. El procedimiento de simulación del sistema consiste en resolver o evaluar dichas ecuaciones, que por lo general se refiere a un conjunto de *ecuaciones diferenciales*.

Dentro de la historia de la ecología moderna, las ecuaciones diferenciales han jugado un rol crucial en el desarrollo de modelos analíticos. Originalmente fueron desarrolladas para describir cuantitativamente los cambios de una o más variables en procesos físicos, como por ejemplo: *transferencia de calor*, *transferencia de masa*, *dinámica de fluidos*, *electrodinámica*, etc. Durante las últimas décadas, este enfoque ha sido adaptado para el modelado de procesos ecológicos, en particular, el fenómeno de la dinámica de poblaciones [31].

Según [73], el uso de ecuaciones diferenciales en el modelado de sistemas debe ser utilizado bajo las siguientes hipótesis:

- Las ecuaciones diferenciales trabajan con cantidades homogéneas. No describen la heterogeneidad existente entre miembros de un un sistema.

- Las ecuaciones diferenciales son determinísticas y funcionales. Dada una cierta entrada, la salida siempre será la misma secuencia de estados. Las influencias estocásticas son excluidas.

- Las ecuaciones diferenciales son continuas. Dada cualquier entrada dentro del dominio del problema, siempre existirá un estado de salida.

Dependiendo la cantidad de variables independientes, la ecuaciones diferenciales pueden clasificarse en: *ecuaciones diferenciales ordinarias* y *ecuaciones diferenciales parciales*.

2.1.1. Ecuaciones diferenciales ordinarias

Las ecuaciones diferenciales se denominan *ordinarias* cuando existe una única variable independiente. Generalmente, ecuaciones diferenciales ordinarias (EDO) en ecología son utilizadas para describir crecimiento o densidad de población en el tiempo.

Uno de los principales ejemplos de utilización de EDOs en ecología es el modelo *presa-depredador* desarrollado independientemente por los matemáticos *Alfred J. Lotka* [39] y *Vito Volterra* [72]. Este modelo describe de forma simple la interacción entre una población de presas y una población de depredadores. Las ecuaciones de *Lotka-Volterra* son un par de EDOs *no lineales* de primer orden, en donde la evolución en el tiempo del sistema está dada por las siguiente ecuaciones:

$$\frac{dx}{dt} = \alpha * x - \beta * x * y \qquad (2.1) \qquad \frac{dy}{dt} = \beta * b * x * y - \gamma * y \qquad (2.2)$$

En donde, x representa a la población de presas, y representa a la población de depredadores, $\frac{dx}{dt}$ representa la variación de la población de presas en el tiempo, $\frac{dy}{dt}$ representa la variación de la población de depredadores en el tiempo, α, β, γ y b son constantes positivas cuyos valores típicos son menores que 1,0, α es la tasa de crecimiento por unidad de tiempo para la población de presas, β es el factor de depredación, γ es la tasa de mortalidad por unidad de tiempo para la población de depredadores y b especifica que fracción de la biomasa de las presas capturadas es transformada en biomasa del depredador.

En la ecuación 2.1 las expresiones $(\alpha * x)$ y $(\beta * x * y)$ representan: el crecimiento exponencial de la población de presas y la frecuencia con que un depredador se encuentra con una presa, respectivamente. En la ecuación 2.2 las expresiones $(\beta * b * x * y)$ y $(\gamma * y)$ representan: la cantidad de presas capturadas que son convertidas en biomasa del depredador y disminución exponencial de la población de depredadores, respectivamente.

2.1.2. Ecuaciones diferenciales parciales

Las ecuaciones diferenciales se denominan *parciales* cuando existe más de una variable independiente e involucran derivadas parciales de las variables dependientes. Las ecuaciones diferenciales parciales (EDP) son una de las principales herramientas matemáticas para describir procesos espacio-temporales [27]. Debido a que las EDPs representan de manera más cercana a la realidad a los sistemas modelados, son mucho más difícil de resolver que las EDOs. Las EDPs

permiten a los modeladores incorporar simultáneamente el *tiempo* y *espacio* en las ecuaciones que representan la dinámica de poblaciones.

Uno de los principales ejemplos de utilización de EDPs en ecología son los *modelos de invasión*. En 1951, *John G. Skellam* [62] desarrolló un modelo matemático para modelar invasiones ecológicas por medio de las ecuaciones de *reacción-difusión* para representar la expansión de poblaciones de ratas almizcleras en Europa. Las ecuaciones de *reacción-difusión* describen los cambios de densidad de una población de forma continua en el tiempo y espacio por medio de dos procesos independientes: *reacción* - crecimiento, interacciones, cambios de estado y *difusión* - movimiento local. La forma general de la ecuación de *reacción-difusión* está dada por:

$$\frac{\partial N}{\partial t} = f(N)N + D\left(\frac{\partial^2 N}{x^2} + \frac{\partial^2 N}{y^2}\right) \tag{2.3}$$

En donde x e y corresponden con un punto (x, y) en el espacio, N es la densidad de la población en (x, y), $f(N)$ es una función que describe la tasa de crecimiento per cápita de N y D es el coeficiente de difusión.

El modelo de invasión de *Skellam* considera que la invasión animal es producida con una tasa de crecimiento *malthusiana* (tasa de crecimiento exponencial), por lo tanto la función $f(N)$ es reemplazada por $r\left(1 - \frac{N}{K}\right)$ (*tasa de crecimiento logístico*, tasa crecimiento exponencial hasta que la población se acerca a la *capacidad de carga* del ambiente), en donde r es una constante que representa la tasa de crecimiento de la población y K es la capacidad de carga (cantidad máxima individuos que un área puede soportar hasta consumir todos los recursos disponibles).

En la ecuación 2.3, la expresión $D\left(\frac{\partial^2 N}{x^2} + \frac{\partial^2 N}{y^2}\right)$ representa el término de difusión, en donde, D es el coeficiente de difusión, el cual corresponde con la media del desplazamiento al cuadrado por unidad de tiempo y $\left(\frac{\partial^2 N}{x^2} + \frac{\partial^2 N}{y^2}\right)$ es la expresión de difusión, la cual describe cómo el movimiento afecta a la densidad a una distancia en particular desde el punto de inicial de desplazamiento.

2.2. Modelado orientado a individuos

El modelado orientado al individuo (MoI) es un paradigma computacional que nos permite analizar las propiedades que emergen en un sistema a partir de la interacción autónoma entre componentes de bajo nivel. El requerimiento mínimo para desarrollar un modelo orientado al individuo es poder representar entidades individuales por separado y poderlas distinguir por una o más características [31]. Estas entidades actúan en base a cierta información proveniente de la interacción con otros individuos y/o el ambiente en el que residen. Esta información fluye a

través del sistema produciendo patrones colectivos.

Los modelos orientados al individuos también son conocidos como modelos basados en *individuos/agentes/entidades*. Las principales ventajas de utilizar MoIs en comparación a MbEs son [5] [34] [23]:

Perspectiva más cercana a la realidad: Permiten realizar una descripción natural de sistema, aproximando el modelo a la realidad. También, permiten modelar sistemas formados a partir de otros sistemas. Además, permiten modelar sistemas basados en reglas de comportamiento observadas en sistemas reales.

Emergencia: Permiten capturar el fenómeno emergente, es decir, determinar cuales son los comportamientos del sistema que emergen a partir de rasgos adaptativos de los individuos, y de la interacción con otros individuos y/o el ambiente.

Flexibilidad: Ofrecen la posibilidad de trabajar con una gran cantidad de variables de estado independientes. Cada individuo está caracterizado por un conjunto de variables independientes (atributos), las cuales son modificadas dinámicamente a medida que la simulación avanza. Esto permite adaptar el comportamiento de cada individuo (incrementar o decrementar su complejidad), a partir de su experiencia.

Versatilidad: Permiten incluir componentes determinísticos y estocásticos dentro del mismo modelo. Muchos MoI están compuestos por reglas de comportamiento determinísticas que son influenciadas estocásticamente para incorporar las incertidumbres observadas en sistemas reales.

Observabilidad: Permite observar y evaluar la evolución de las variables a través del tiempo. En otras palabras, ofrece la capacidad de analizar en cada paso de simulación cómo los componentes independientemente evolucionan hasta lograr un estado global consistente del sistema.

2.2.1. Individuos

Un individuo es la abstracción de una entidad perteneciente al *mundo real*. Un individuo es representado por medio de un conjunto de *atributos* y *reglas de comportamientos*. Los *atributos* son variables de estado con la capacidad de distinguir a un individuo de otro. Estos atributos pueden representar: tamano, edad, género, localización espacial, etc. La capacidad de diferenciar a los individuos de una misma especie a partir de sus atributos ofrece un primer nivel de heterogeneidad. Las *reglas de comportamiento* definen la manera de actuar de un individuo dependiendo de su percepción en un momento en particular. Estas reglas de comportamiento operan sobre los atributos del individuo. Por

ejemplo, un individuo puede modificar su ubicación espacial para evitar el ataque de otro individuo que está amenazando su seguridad. La capacidad de poder definir individuos con distintas reglas de comportamiento ofrece un segundo nivel de heterogeneidad.

Las principales características que debe poseer un individuo son [40, 41, 42, 43]:

Modularidad: Un individuo es un ente discreto e identificable, definido por un conjunto de atributos y reglas de comportamiento, y con la capacidad de tomar decisiones.

Autonomía: Un individuo reacciona independientemente a los estímulos provenientes de otros individuos y/o su ambiente.

Sociabilidad: Un individuo es un ente social que interactúa con otros individuos.

Ubicabilidad: Un individuo reside en un ambiente en el cual interactúa con otros individuos.

Pro-actividad: Un individuo puede tener objetivos específicos que cumplir.

Adaptatividad: Un individuo tiene la capacidad de aprender y adaptarse dependiendo de su experiencia.

Individuos pueden representar, por ejemplo: *peces* en *bancos de peces* [28, 29], *pájaros* en *bandadas* [59, 60], *abejas* en *enjambres* [32], etc.

2.2.2. Ambiente o *framework*

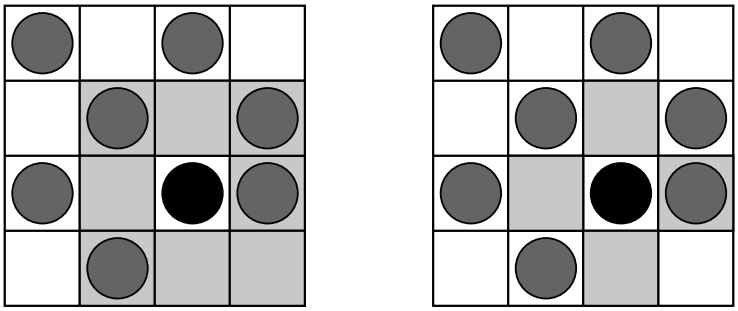
El ambiente o *framework* puede definirse como la ubicación lógica en donde los individuos residen, actúan e interactúan. La naturaleza de ambiente define cómo los individuos deben reaccionar dependiendo la situación que los rodea. Un ambiente puede estar compuesto por un conjunto de objetos, como por ejemplo: *obstáculos* o *fronteras*, que permiten una descripción más cercana a la realidad del entorno en el que los individuos se desenvuelven. La interacción *individuo-ambiente* es un comportamiento común incluido en la formulación de modelos orientados al individuo, ya que los individuos se ven directamente afectados por las fuerzas ejercidas por los objetos pertenecientes al ambiente. Los ambientes o *frameworks* pueden ser clasificados dentro de dos clases: *espacio discreto* y *espacio continuo*.

Modelos de espacio discreto

Los *modelos de espacio discreto* son aquellos en los cuales el espacio es dividido en un conjunto de celdas discretas de igual tamano y forma. Este tipo de modelos

son también conocidos como *autómatas celulares*. En un autómata celular un individuo es almacenado en una única celda y es directamente afectado por los individuos pertenecientes a su vecindario (ver Fig. 2.1). Existen dos tipos de vecindarios comúnmente utilizados en la implementación de autómatas celulares: *vecindario de Von Neumann* (ver Fig. 2.1(b)), el cual considera como vecindario a los indivudos alojados en celdas ubicadas al norte, sur, este y oeste, y *vecindario de Moore* (ver Fig. 2.1(a)), que considera como vecindario a todos los individuos pertenecientes a las celdas adyacentes.

Un autómata celular evoluciona en pasos discretos de tiempo y la actualización de cada individuo se realiza simultáneamente a partir de la información obtenida de los individuos pertenecientes a su vecindario.



(a) Vecindario de *Moore*.

(b) Vecindario de *Von Neumann*.

Figura 2.1: Ejemplos de vecindarios en autómatas celulares.

El *juego de la vida*, desarrollado en 1970 por *John Horton Conway* [20], es un ejemplo fundamental de autómata celular. Las individuos alojados en las celdas de este autómata celular pueden asumir dos estados: *vivo* o *muerto*. Cada individuo es influenciado por los individuos pertenecientes a su vecindario, en particular, el vecindario de *Moore*. El modelo está compuesto por tres reglas determinísticas simples:

Nacimiento: un individuo cambia de estado *muerto* a *vivo* si tiene exactamente 3 individuos vecinos en estado *vivo*.

Muerte: un individuo cambia de estado *vivo* a *muerto* si tiene menos de 2 o más de 3 individuos en estado *vivo*.

Supervivencia un individuo continúa en estado *vivo* si tiene 2 o 3 individuos vecinos en estado *vivo*.

El *juego de la vida* es un ejemplo de sistema en el que un comportamiento emergente complejo surge a partir de la aplicación de un conjunto de reglas de simples.

Modelo de espacio continuo

Los *modelos de espacio continuo* describen la posición de los objetos en el espacio por medio de variables continuas. A diferencia de los modelos de espacio discreto, en los que es espacio de simulación de dividido en una malla de dimensiones $N \times M$ (ej. espacio bidimensional), y un individuo arbitrario i es asociado a una celda $C(2,3)$, en los modelos de espacio continuo un individuo arbitrario i puede residir en cualquier posición del espacio de simulación, como por ejemplo, $x = 1{,}2342, y = 2{,}5315$.

Por un lado, la gran ventaja que ofrecen los modelos de espacio continuo es que ajustan la descripción del sistema modelado a partir de una perspectiva más cercana a la realidad. Ejemplos de sistemas reales, tales como: peces en *bancos de peces*, pájaros en *bandadas*, mamíferos en *manadas* o insectos en *enjambres*, trabajan con cantidades continuas, es decir, los individuos puede residir en cualquier posición en el espacio. Por otro lado, la gran desventaja que entregan los modelos de espacio continuo, en contraste con los modelos de espacio discreto, es que cada individuo debe evaluar exhaustivamente a todos los individuos residentes en el ambiente con el objetivo de encontrar su vecindario.

2.2.3. Modelos espacialmente explícitos

Un modelo orientado al individuo espacialmente explícito es aquel en el que los individuos están asociados a una posición en el espacio de simulación (continuo o discreto). Los MoI espacialmente explícitos permiten enlazar a los individuos a una posición geográfica en el ambiente en el que residen en el *mundo real* decrementando el nivel de abstracción del modelo.

A diferencia de los MoI basados en *redes* o *grafos*, en los cuales los individuos están conectados a través de *enlaces* o *aristas*, y cuya ubicación geográfica no es relevante, en los MoI espacialmente explícitos la ubicación geográfica de los individuos (en particular, la ubicación espacio-temporal), permite extraer conclusiones de real importancia en los sistemas estudiados, como por ejemplo: *estructura de la población*, *densidad de población*, *tasas de migración*, etc.

2.2.4. Modelos con patrones de movimiento

Un modelo orientado al individuo espacialmente explícito puede exhibir patrones de movimiento, es decir, los individuos pueden cambiar su posición en el ambiente a medida que la simulación avanza. Los patrones de movimiento exhibidos por los individuos de un sistema son un comportamiento complejo que representa un fenómeno espacio-temporal. En la gran mayoría de MoI espacialmente explícitos los patrones de movimiento juegan un rol fundamental en el funcionamiento del sistema. Por ejemplo, un banco de peces puede migrar de una región a otra en búsqueda de alimento, aguas más cálidas o un lugar

adecuado para llevar a cabo el proceso de reproducción. Además, de la interacción entre individuos junto con los patrones de movimiento pueden emerger patrones de comportamiento colectivos complejos, tales como: *cohesión* o *sincronización*.

Estos patrones de movimiento pueden ser observados en una amplia gama de organismos en diferentes escalas y niveles de complejidad, como por ejemplo: bacterias moviéndose a través la concentración de ciertas sustancias químicas en su ambiente, migraciones de pájaros o de mamíferos, concentraciones de multitudes de personas, etc.

2.2.5. Eventos

Se definirá como un *evento* al mecanismo de planificación por medio del cual se actualizan los atributos de los individuos. El dinamismo de los sistemas del mundo real es modelado a través de un conjunto de eventos discretizados en el tiempo. Estos eventos pueden representar: *comunicación entre individuos*, *reacción de un individuo a cierta acción*, *realizar una acción cooperativa entre individuos*, etc.

2.3. Ejemplos de modelos orientados al individuo

A continuación se describirán algunos de los modelos orientados al individuo más conocidos dentro de la comunidad científica.

2.3.1. *Fish school*: Modelo de *Huth & Wissel*

El modelo de *Huth & Wissel* [28, 29] es un ejemplo pragmático de auto-organización en sistemas biológicos.

Hipótesis fundamentales

Este modelo fue formulado en base a las siguientes hipótesis:

(1) Cada individuo actúa de acuerdo al mismo modelo de comportamiento. Esto asegura que el grupo de mueva sin líderes.

(2) Cada individuo es independientemente afectado por influencias aleatorias. Por lo tanto, la posición y velocidad de cada individuo son construidas con variables estocásticas.

(3) El movimiento de cada individuo es solamente afectado por sus vecinos más cercanos. Los sentidos de *visión* y *línea lateral* son de vital importancia para la formación del banco de peces.

(4) Los individuos del grupo no son afectados por estímulos externos.

(5) El modelo se caracteriza por su simplicidad. El modelo no está interesado en describir en detalle el comportamiento de cada individuo, pero si en los patrones de comportamientos decisivos para la formación del grupo.

Posición y orientación

Para un pez i, su nueva posición $x_i(t + \Delta t)$ y orientación $v_i^0(t + \Delta t)$ son calculadas a partir de su posición $x_i(t)$ y orientación $v_i^0(t)$, y la de sus vecinos. Dado que el ángulo de giro $\alpha_i(t)$ es calculado por el modelo y la rapidez $v_i(t+\Delta t)$ es obtenida de forma aleatoria, la nueva posición $x_i(t+\Delta t)$ y orientación $v_i^0(t+\Delta t)$ están dadas por:

$$\begin{aligned} x_i(t+\Delta t) &= x_i(t) + \Delta t \cdot v_i(t+\Delta t) \\ v_i^0(t+\Delta t) &= \begin{bmatrix} \cos\alpha_i(t) & -\sin\alpha_i(t) \\ \sin\alpha_i(t) & \cos\alpha_i(t) \end{bmatrix} v_i^0(t) \\ v_i(t+\Delta t) &= v_i(t+\Delta t) \cdot v_i(t+\Delta t) \end{aligned} \tag{2.4}$$

Áreas de influencia

Con el objetivo de determinar adecuadamente que acción realizar dependiendo la situación que rodea a un individuo, el modelo identifica tres áreas de influencia: repulsión - r_1, orientación paralela - r_2, y atracción - r_3 (ver Fig. 2.2).

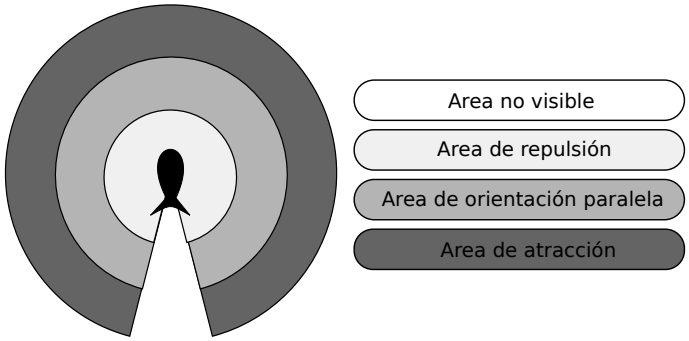


Figura 2.2: Áreas de influencia del modelo de *Huth & Wissel*

En donde, $r_1 < r_2 < r_3$

Reglas de comportamiento

A continuación se describirán las reglas de comportamiento básicas que determinan cómo un pez reacciona a la influencia de un único vecino.

Repulsión: Este comportamiento permite evitar colisiones entre individuos que están demasiado cerca. Si el pez vecino se encuentra en el área de repulsión, el pez gira de forma perpendicular para evitar la colisión (ver Fig. 2.3(a)). Por lo tanto, el ángulo de giro del pez está dado por:

$$\beta_{ij} \quad = \quad \text{mín}\left\{\angle(v_i^0, v_j^0) \pm \tfrac{\pi}{2}\right\} \tag{2.5}$$

Orientación paralela: Este comportamiento permite mantener un grupo altamente sincronizado, es decir, un grupo de peces nadando hacia la misma dirección. Si el pez vecino se encuentra en el área de orientación paralela, el pez nada hacia la misma dirección que su vecino (ver Fig. 2.3(b)). Por lo tanto el ángulo de giro del pez esta dado por:

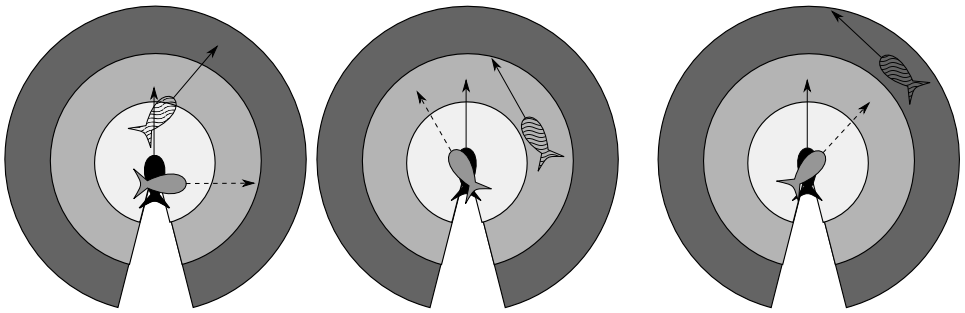
$$\beta_{ij} \quad = \quad \angle(v_i^0, v_j^0) \tag{2.6}$$

Atracción: Este comportamiento permite mantener un grupo fuertemente cohesionado, es decir, prolongar la formación de grupo a través del tiempo. Si el pez vecino se encuentra en el área de atracción, el pez nada hacia su vecino. Por lo tanto, el ángulo de giro está dado por (ver Fig. 2.3(c)):

$$\beta_{ij} \quad = \quad \angle(v_i^0, x_j - x_i) \tag{2.7}$$

Búsqueda de vecinos: Además, si un pez no puede detectar a otros peces dentro de su campo de percepción, el modelo incluye un comportamiento de *búsqueda de vecinos*. Si el pez vecinos se encuentran demasiado lejos o en el *dead angle*, el pez comienza a buscar peces vecinos. Por lo tanto, el ángulo de giro del pez está dado por:

$$\beta_{ij} \quad = \quad chance([-\pi, \pi]) \tag{2.8}$$



(a) Repulsión. (b) Orientación paralela. (c) Atracción.

Figura 2.3: Reglas de comportamiento del modelo de *Huth & Wissel*

Incertidumbre en el ángulo de giro

Como se sabe, un pez detecta la posición y orientación de sus vecinos con cierta incertidumbre, es decir, el pez no gira exactamente un ángulo β_{ij}. Para tomar en cuenta las incertidumbres e influencias aleatorias, el modelo incluye una distribución de probabilidad $\rho(\alpha_i)$ para los ángulos de giro β_{ij}. Para el caso de un único vecino se utiliza $\rho(\alpha_i)$ como la distribución normal con media β_{ij}, tal que:

$$\alpha_i \quad = \quad chance(\rho(\alpha_i)) \tag{2.9}$$

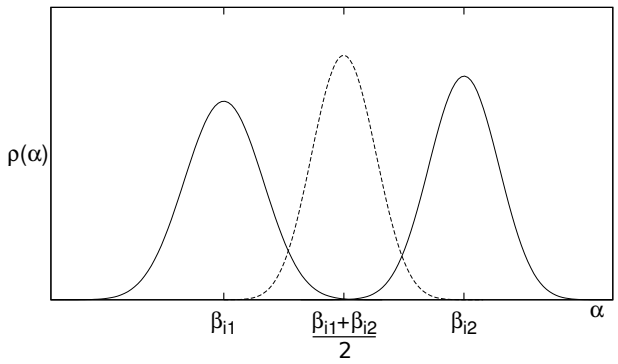


Figura 2.4: Distribución *normal*.

Rapidez

Con el objetivo de simplificar el modelo, la nueva rapidez v_i del pez i es elegida independientemente de otros peces. La rapidez v_i es determinada de forma aleatoria a partir de la distribución *Gamma* (ver Fig. 2.5).

$$\begin{aligned} v_i &= chance(\rho(v)) \\ \rho(v) &= \frac{A^K}{\Gamma(K)} \cdot \exp(-Av) \cdot v^{K-1} \end{aligned} \tag{2.10}$$

En donde, $\Gamma(K)$ es la función Gamma, v es la rapidez, K y A son parámetros, tal que, $K = 4, A = 3{,}3$

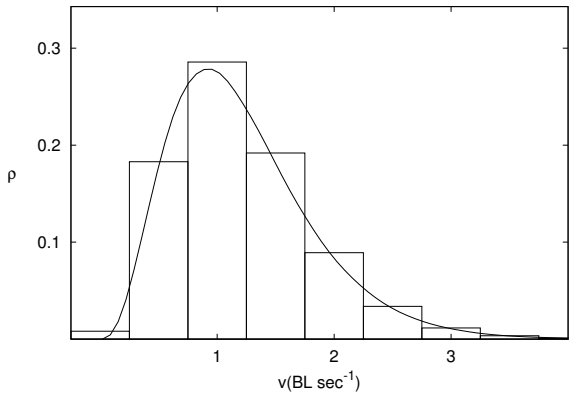


Figura 2.5: Distribución *gamma*.

Mezcla de influencias

Un elemento importante del modelo es cómo mezclar la influencia de varios vecinos. Para esto, el modelo mezcla la influencia de sus vecinos por medio del promedio de los ángulos de influencia de sus vecinos:

$$\overline{\beta_{ij}} \quad = \quad \sum_{j=1}^{nb} \frac{\beta_{ij}}{nb} \tag{2.11}$$

Por lo tanto, la distribución de probabilidad para el ángulo de giro $\rho(\alpha_i)$ queda dada por:

$$\rho(\alpha_i) \quad = \quad \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(\alpha_i - \overline{\beta ij})^2}{2\sigma^2}\right) \tag{2.12}$$

2.3.2. *Boids*: Modelo de *Craig Reynolds*

En 1986, *Craig Reynolds* [59, 60] desarrolló un modelo computacional para simular el movimiento coordinado grupos de animales, tales como: bandadas de pájaros o bancos de peces. *Reynolds* llamó *boids* (*birdoid*, *pajaroide*), a los individuos simulados. El modelo consta de tres *reglas de comportamiento* simples: *cohesion*, *alineación* y *separación*, las cuales describen las maniobras de un único *boid* basado en las posiciones y velocidades de otros *boids* pertenecientes a su vecindario (ver Fig. 2.6)

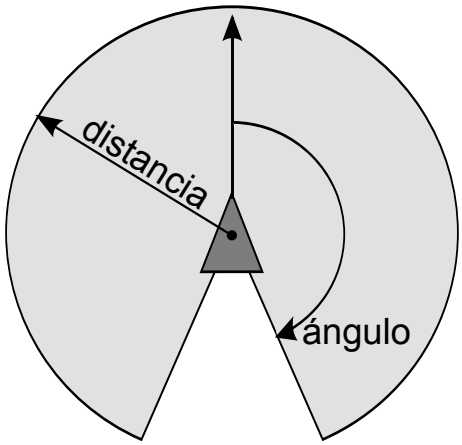
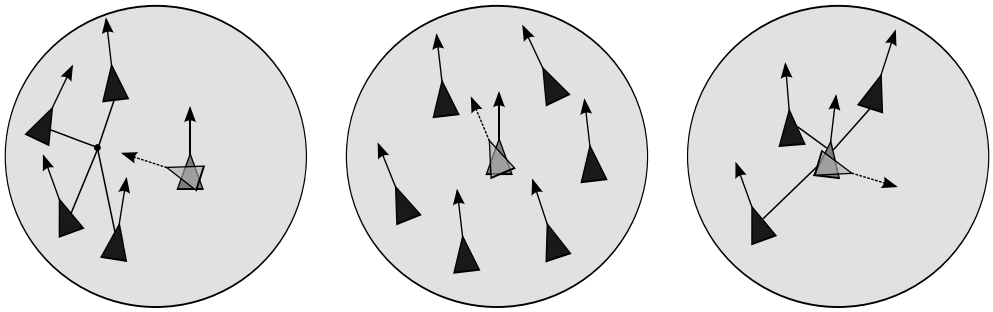


Figura 2.6: Vecindario *boid*.

Cohesion: Intentar de mantenerse cerca de los *boids* vecinos (ver Fig. 2.7(a)).

Alineacion: Igualar la velocidad con los *boids* vecinos (ver Fig. 2.7(b)).

Separacion: Evitar colisiones con otros *boids* y obstáculos existentes en el ambiente (ver Fig. 2.7(c)).



(a) Cohesion. (b) Alineacion. (c) Separacion.

Figura 2.7: Reglas de comportamiento del modelo de *Craig Reynolds*

El modelo de *boids* de *Craig Reynolds* es una fascinante demostración de emergencia y auto-organización, y aunque no representa realmente el comportamiento de un sistema en particular, es la base para la formulación de otros modelos.

2.3.3. *Social force model*: Modelo de *Dirk Helbing*

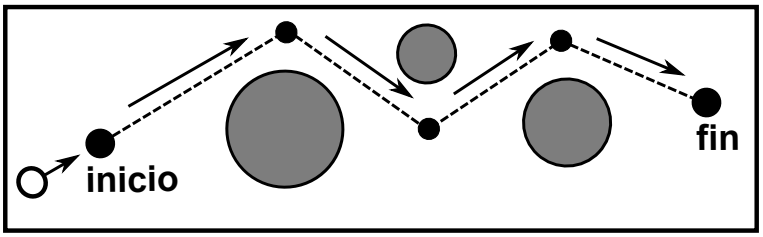
En 1995, *Dirk Helbing* desarrolló un modelo desarrollado para describir el movimiento coordinado de multitudes de peatones. El *modelo de fuerzas sociales*

para dinámica de peatones [26] especifica la influencia de otros peatones y del ambiente sobre el comportamiento individual de un peatón. Los cambios de comportamiento de un peatón son guiados por las llamadas *fuerzas sociales* o *campos sociales* (áreas de interacción). Este modelo consta de tres reglas de comportamiento, las cuales serán descritas a continuación:

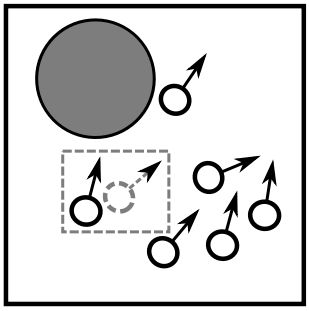
Dirección por objetivos: Usualmente, un peatón desea alcanzar cierto destino de la forma más rápida posible. Este destino generalmente se trata de una puerta o de cierta área en el ambiente. El modelo describe el movimiento de un peatón a través de una ruta de forma poligonal. El peatón mueve recorriendo las aristas del polígono hasta llegar a su meta (ver Fig. 2.8(a)).

Repulsión: El movimiento de un peatón es afectado por otros peatones. El peatón trata de mantenerse a cierta distancia de otros peatones. Cada peatón tiene una *esfera de interés* privada. Si el peatón detecta que otro peatón se encuentra demasiado cerca se produce un *efecto repulsivo*. Además, un peatón trata también de mantenerse lejos de murallas, calles, obstáculos, etc. Este tipo de objetos en el ambiente también producen un efecto repulsivo sobre el peatón (ver Fig. 2.8(b)).

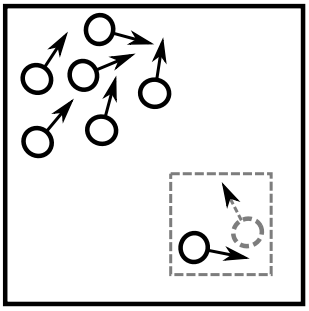
Atracción: En ocasiones, los peatones son atraídos por otros peatones u objetos en el ambiente. Esta atracción es la causa por la cual los peatones forman grupos (ver Fig. 2.8(c)).



(a) Dirección por objetivos.



(b) Repulsión.



(c) Atracción.

Dado que los comportamientos ejecutados por un peatón se realizan en un mismo instante de tiempo, se considera como el efecto total a la suma de todos los efectos. El modelo toma en cuenta ciertas fluctuaciones que son incluidas por medio de variables aleatorias en el comportamiento del peatón.

2.3.4. *Self-propelled particles*: Algoritmo de *Couzin-Vicsek*

El modelo de *Couzin-Vicsek* fue formulado con el objetivo investigar el movimiento emergente *auto-ordenado* en sistemas de partículas con interacción biológicamente motivada.

La única regla del modelo consiste en: *en cada paso de simulación, una partícula de velocidad constante y absoluta asume como su dirección de movimiento el promedio de las velocidades de las partículas de su vecindario de radio r, más alguna perturbación aleatoria* [68] (ver Fig. 2.8).

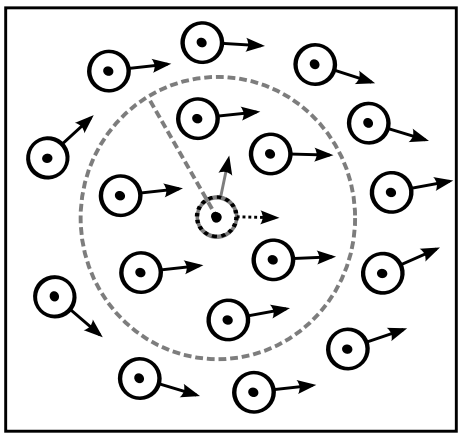


Figura 2.8: Ejemplo del modelo de partículas auto-propulsadas.

A pesar de su simplicidad, el modelo es bastante interesante debido a sus posibles aplicaciones en una amplia gama de sistemas biológicos que involucran agrupación y migración.

Capítulo 3

Simulación y simulación distribuida

Una simulación es un programa computacional, basado en un modelo matemático o lógico, que representa o emula el comportamiento de un sistema. El modelo es una abstracción del *mundo real* que permite describir, de una forma simplificada, las relaciones entre los componentes de un sistema.

Los términos básicos necesarios para comprender el concepto de *sistema* son los siguientes: *estado del sistema*, conjunto de variables necesarias para describir al sistema en cualquier instante de tiempo; *entidad*, abstracción que modela a un componente del *sistema*; *atributo*, propiedad o característica de una *entidad*; *actividad*, proceso que involucra un cambio en el *estado del sistema*; *evento*, ejecución satisfactoria de una *actividad* en un instante específico de tiempo. Un sistema es definido como una colección de *entidades* que actúan e interactúan simultáneamente para satisfacer un objetivo lógico.

Dependiendo en la complejidad de las relaciones entre componentes establecidas en el modelo, se puede optar entre dos estrategias de solución: una *solución analítica*, para modelos de baja complejidad ó una *simulación numérica*, para modelos de alta complejidad. En una *solución analítica* se utilizan métodos matemáticos para obtener información exacta de la pregunta de interés. En una *simulación*, generalmente, se utiliza un computador para evaluar un modelo *numéricamente*, recolectando datos para estimar las características deseadas del sistema [35].

La *simulación computacional* es utilizada como una herramienta de soporte que permite comprender dinámicamente el funcionamiento de sistemas de alto nivel de complejidad. Esta herramienta ha sido difundida a lo largo de una amplia gama de ámbitos de estudios debido a su gran flexibilidad y versatilidad. La simulación computacional permite experimentar hipotéticamente con un sistemas sin alterar el funcionamiento del *sistema físico*. A continuación se detallarán

algunos ejemplos de aplicación en los cuales la simulación computacional ha sido usada como una potente herramienta:

Entretenimiento: La industria de los videojuegos ha sacado gran provecho de la simulación computacional. Hoy en día, la gran mayoría de los videojuegos incluyen módulos de simulación integrados utilizados, por ejemplo, para lograr una interacción *jugador-jugador* y/o *jugador-ambiente* más realista.

Industria: Un sistema de producción industrial, generalmente, consiste en una cadena de producción que involucra múltiples componentes mecánicos. Realizar un cambio en uno de estos componentes puede ser una tarea bastante difícil y, mas aún, un error puede ser catastrófico. La simulación computacional permite manipular este tipo de procesos en un entorno virtual sin las limitaciones que involucran las modificaciones físicas de los componentes. También, permite la creación de sistemas hipotéticos con el objetivo de evaluar la viabilidad de su implementación.

Predicciones meteorológicas: En la actualidad, la predicción meteorológica juega un rol fundamental en el día a día de las personas. Las predicciones meteorológicas consisten en la simulación computacional de un modelo numérico de la atmósfera. Este tipo de modelos describen procesos físicos complejos por medio de ecuaciones. Generalmente, este tipo de ecuaciones son muy difícil de resolver analíticamente, por que la simulación computacional es una herramienta primordial para resolver este tipo de problemas.

Catástrofes naturales: Fenómenos naturales como: *huracanes*, *tsunamis*, *terremotos*, *erupciones volcánicas* etc, se caracterizan por sus devastadores resultados sobre la población. Generalmente, estos desastres naturales extremos ocurren de forma inesperadamente causando la muerte de miles de personas. El modelado y simulación de este tipo fenómenos ha dado un gran paso en el desarrollo de sistemas de prevención que permiten planear estrategias de evacuación con antelación al desastre.

Aplicaciones militares: Los principales usos de la simulación en ámbitos militares se pueden clasificar en:

Decisiones tácticas: Este tipo de simulaciones son conocidas como *war gamming simulations*, y son utilizadas para evaluar diferentes estrategias al momento de atacar o defenderse de una fuerza enemiga.

Entrenamiento: La simulación se utiliza para la integración de pilotos, operadores de tanques u otro personal técnico dentro de un ambiente virtual de entrenamiento realista.

Análisis operacional: La simulación es utilizada como herramienta de soporte para realizar ensayos con armas para su posterior desarrollo o adquisición.

Las simulaciones computacionales han sido utilizadas como una estrategia para la *toma de decisiones* debido a su gran capacidad predictiva y a la rápida obtención de resultados. Los pasos típicos para construir y utilizar un programa de simulación consisten en [49]:

(1) Comprender el comportamiento de los componentes del *sistema físico*.

(2) Capturar las características fundamentales de los componentes del *sistema físico*. Construir un modelo matemático o lógico a partir de dichas características.

(3) Implementar un *programa de simulación* (usualmente llamado *simulador*), basado en el modelo formulado en el paso anterior y ejecutarlo en una plataforma computacional.

(4) Analizar la salida del *simulador* con el objetivo de comprender y predecir el comportamiento del *sistema real*.

Adicionalmente, el modelo y el simulador deben ser validados y verificados durante los pasos (2) y (3).

3.1. Simulación computacional

Una simulación computacional es *programa computacional* que emula el comportamiento de un sistema. El sistema emulado es llamado *sistema físico* [19]. Las simulaciones computacionales definen una colección de *variables de estado* que representan el estado del *sistema físico*. Los cambios de estado del *sistema físico* son realizados por medio de mecanismos implementados en el programa de simulación. El tiempo en el *sistema físico* es representado a través de una abstracción llamada *tiempo de simulación*.

3.1.1. Clasificación de los modelos de simulación

Cada modelo de simulación es una especificación de un *sistema físico* caracterizado por un conjunto de variables de estados y eventos (transición de estados). Una simulación consiste en imitar la ocurrencia de eventos reflejando su comportamiento en las variables de estado en función del tiempo. Dependiendo la naturaleza del sistema, se pueden clasificar los modelos de simulación dentro de las siguientes categorías (ver Fig. 3.1):

Estáticos y dinámicos

Un sistema es considerado *estático* si el comportamiento del sistema no varía en función del tiempo o el tiempo no es un parámetro relevante en la evolución del sistema. Por lo tanto, un modelo de simulación estático puede ser definido como la representación de un sistema en una fracción específica de tiempo. Un típico ejemplo de simulación de sistemas estáticos son las simulaciones *Monte Carlo* (utilizadas para resolver problemas matemáticos de gran complejidad a partir de la generación de números aleatorios).

Por el contrario, un sistema es considerado *dinámico* si el comportamiento del sistema evoluciona en función del tiempo. Por lo tanto, un modelo de simulación dinámico representa la evolución de un sistema a través del tiempo.

Determinísticos y estocásticos

Un sistema exhibe un comportamiento probabilístico o *estocástico* si existe algún elemento de aleatoriedad o incertidumbre. Los sistemas estocáticos incluyen intrínsecamente perturbaciones aleatorias las cuales provocan estados futuros impredecibles. Dada una cierta entrada, la salida del sistema es una variable aleatoria.

Por el contrario, los sistemas *determínisticos* son aquellos en los que no existen influencias aleatorias que afecten al comportamiento del sistema. El funcionamiento del sistema es completamente predecible. Dada una cierta entrada, la salida del sistema siempre será la misma.

Continuos y discretos

Un modelo de simulación es considerado *continuo* si la evolución del sistema ocurre de forma ininterrumpida en el tiempo. Usualmente, los modelos basados en ecuaciones diferenciales describen los cambios de comportamiento de un sistema en función del tiempo.

Un modelo de simulación es considerado *discreto* si la evolución del sistema ocurre de acuerdo a pasos incrementales de tiempo. En este tipo de modelos los cambios en las variables de estado se llevan a cabo instantáneamente en puntos separados de tiempo. El típico ejemplo de un sistema discreto es un *banco* en el que un conjunto de *cajeros* atiende a un conjunto de *clientes*. Cada cliente espera turno hasta que un cajero está disponible, realiza una transacción y finaliza su turno. El tiempo de espera por cliente ocurre en instantes discretos de tiempo de tamano variable.

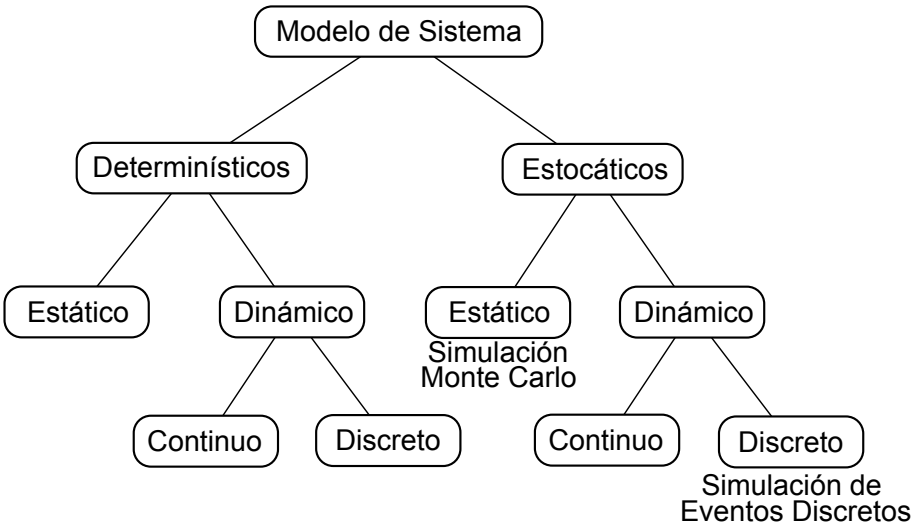


Figura 3.1: Taxonomía de los modelos de simulación.

Los modelos de simulación que se caracterizan por ser: *dinámicos*, *estocásticos* y *discretos* son llamados *modelos de simulación de eventos discretos*.

3.1.2. Eventos

Un evento es una abstracción utilizada en el modelo de simulación para representar una acción instantánea en el *sistema físico* [19]. Los eventos son la idea fundamental en la simulación de eventos discretos. Un evento corresponde con la transición de una o más variables de estado. Un evento posee una marca de tiempo (*timestamp*) que indica en que punto de la simulación sucede dicho evento. El conjunto de eventos de una simulación es almacenado cronológicamente en una estructura llamada *lista de eventos*.

3.1.3. Representación del tiempo

En simulación computacional es fundamentalmente importante distinguir entre las diferentes nociones de tiempo involucradas en el proceso de desarrollo de este tipo de aplicaciones. Dependiendo el ámbito, se puede realizar la siguientes clasificación:

Tiempo físico: Es el tiempo en el *sistema físico*.

Tiempo de simulación: Es una abstracción utilizada por la simulación para modelar el *tiempo físico*.

Tiempo de reloj: Es el tiempo que tarda la ejecución de un programa de simulación.

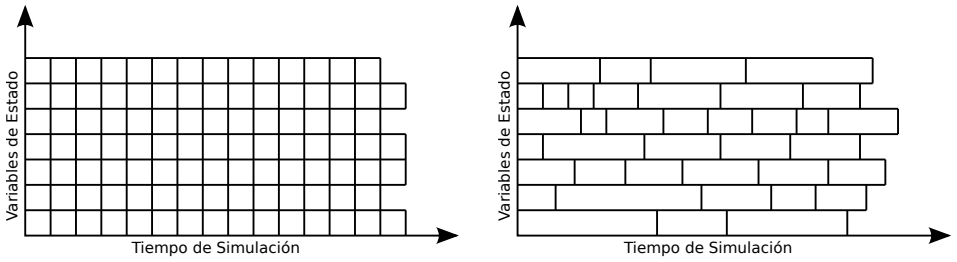
Un componente importante dentro de la simulaciones de eventos discretos es el *reloj de simulación*. Este *reloj de simulación* está fielmente ligado al *tiempo de simulación* y es utilizado para provocar *eventos* en el sistema. El reloj de simulación se incrementa de acuerdo al mecanismo de avance de tiempo implementado.

3.1.4. Progreso del tiempo

En simulación de eventos discretos, existen dos métodos principales con los cuales se puede avanzar el reloj de simulación: *guiado por eventos* y *guiado por tiempo*

Guiado por eventos (*event-driven*): En los mecanismos *guiados por eventos* el reloj de simulación se incrementa cuando sucede un evento. Se extrae el evento con menor *timestamp* de la lista de eventos y se incrementa el reloj de simulación sumando el *timestamp* del evento. El proceso termina cuando la lista de eventos está vacía (ver Fig. 3.2(b)).

Guiado por tiempo (*time-stepped*): En los mecanismos *guiados por tiempo* el reloj de simulación se incrementa en *pasos de tiempo* de tamano constante. Es decir, se divide el tiempo de simulación en intervalos (de igual tamano) Δt, y el *reloj de simulación* avanza desde un Δt hasta el próximo Δt (ver Fig. 3.2(a)).



(a) Guiado por tiempo. (b) Guiado por eventos.

Figura 3.2: Diagramas *tiempo-espacio*.

Los mecanismos *guiados por tiempo* son un caso particular de los mecanismos *guiados por eventos*. Un mecanismo *guiado por tiempo* es un mecanismo *guiado por eventos* en el que la diferencia entre *timestamps* $(t_1, t_2, ..., t_k)$ de los eventos $(e_1, e_2, ..., e_k)$ es constante Δt.

3.2. Simulación distribuida

Durante los últimos anos, el modelado y análisis de sistemas complejos se ha difundido a lo largo de la comunidad científica. La simulación computacional es una herramienta que permite evaluar numéricamente modelos complejos que analíticamente serían casi imposibles de resolver. Por un lado, las simulaciones a gran escala son conocidas por consumir una enorme cantidad de tiempo de ejecución en un entorno secuencial. Por otro lado, la simulaciones de modelos complejos son conocidas por requerir un enorme poder de cómputo para evaluar las relaciones entre componentes. Un enfoque básico para reducir el tiempo de ejecución es por medio de la explotación del paralelismo.

Simulación paralela de eventos discretos (*PDES*, *Parallel Discrete Event Simulation*), también conocida como simulación distribuida, se refiere a la ejecución de un único programa de simulación de eventos discretos sobre un computador paralelo [18]. Este único programa de simulación de eventos discretos es descompuesto en un conjunto de procesos que pueden ejecutarse concurrentemente.

Las principales ventajas de ejecutar un programa de simulación sobre múltiples computadores se pueden resumir en:

Reducir el tiempo de ejecución: Dividir el dominio del problema de una simulación a gran escala en un conjunto de procesos que han de ejecutarse concurrentemente sobre p procesadores puede reducir el tiempo de ejecución de la simulación a $\frac{t}{p}$. Esto no es completamente cierto, ya que la necesidad de intercambio de mensajes entre procesos anade un costo en comunicaciones $\frac{t}{p} + c$. Sin embargo, la ganancia es bastante elevada si se compara con el tiempo de ejecución de la simulación en un entorno secuencial.

Distribución geográfica: Ejecutar el programa de simulación sobre un conjunto de computadores distribuidos geográficamente dispersos permite crear mundos virtuales con múltiples participantes que se encuentran físicamente localizados en diferentes lugares [19].

Heterogeneidad: La simulación distribuida puede simplificar la integración de simuladores que se ejecutan en máquinas de características heterogéneas.

Tolerancia a fallos: La ejecución de la simulación sobre un conjunto de unidades de cómputo ofrece la posibilidad de implementar mecanismos de tolerancia a fallos. Si una unidad de cómputo falla, se tiene la posibilidad de migrar sus tareas hacia otro procesador, permitiendo continuar con la simulación.

3.2.1. Arquitecturas paralelas/distribuidas

La aparición de las arquitecturas paralelas/distribuidas ha marcado una enorme diferencia al momento de resolver problemas complejos y a gran escala. Tratar de resolver este tipo de problemas en entornos secuenciales se ha convertido en una desventaja en términos de costos en tiempos de ejecución. Además, para problemas en los cuales la solución debe obtenerse *lo más rápido posible* la ejecución en un entorno secuencial no es una opción viable.

Las arquitecturas paralelas/distribuidas ofrecen la posibilidad de utilizar múltiples recursos computacionales simultáneamente para resolver un problema computacional. El problema es descompuesto en un conjunto de procesos que pueden ejecutarse concurrentemente. Cada proceso es descompuesto en una serie de tareas que pueden ejecutarse simultáneamente en un conjunto de unidades de cómputo.

La taxonomía de *Flynn* clasifica las arquitectura paralelas (y secuenciales) de acuerdo a dos dimensiones independientes: *Instrucciones* y *Datos*. Según la taxonomía de *Flynn* se pueden clasificar las arquitecturas de computadores en los siguientes grupos:

SISD (*Single Instruction Single Data*): Un único procesador ejecuta un único flujo de instrucciones que opera sobre una único flujo de datos. Es el típico caso de un computador secuencial.

SIMD (*Single Instruction Multiple Data*): Un único procesador (formado por un conjunto de unidades de procesamiento), ejecuta un único flujo de instrucciones sobre diferentes flujos de datos. Ejemplos de este tipo de arquitectura son: *Procesadores vectoriales*, *GPU*, *Procesadores cell*, etc.

MISD (*Multiple Instruction Single Data*): Un conjunto de procesadores ejecuta diferentes flujos de instrucciones sobre un único flujo de datos.

MIMD (*Multiple Instruction Multiple Data*): Un conjunto de procesadores autónomos ejecutan simultáneamente diferentes flujos instrucciones sobre diferentes flujos de datos. Ejemplos de este tipo de arquitecturas son: *supercomputadores*, *sistemas de multiprocesadores simétricos*, *computadores multicore*, *clusters*, *grids*, etc.

En el presente trabajo sólo se hará hincapié en las plataformas *MIMD* y *SIMD*. Según este criterio, las plataformas de hardware pueden ser clasificadas en (ver Fig. 3.3): *computadores paralelos* y *computadores distribuidos*, y su principal diferencia recae en la ubicación física de los componentes de hardware.

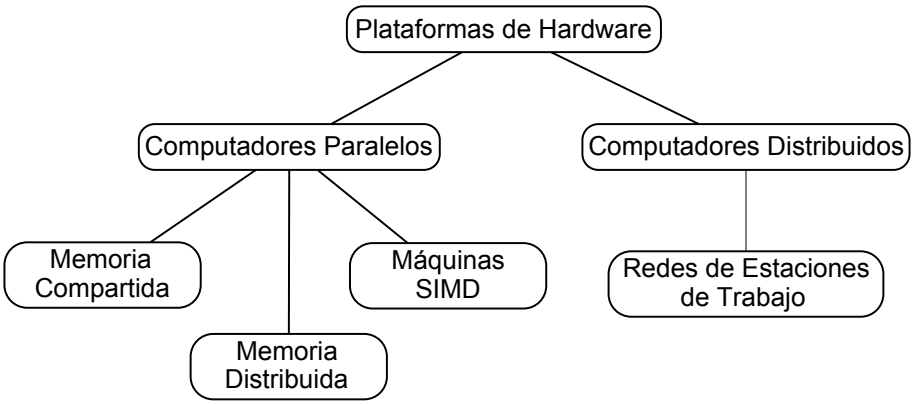


Figura 3.3: Taxonomía de las arquitecturas paralelas/distribuidas más conocidas.

Computadores paralelos

Un computador paralelo está compuesto por un conjunto de procesadores conectados a través de una red de interconexión (o bus), y es capaz de operar de forma cooperativa para resolver un problema computacional. Una de las principales características de los computadores paralelos es que los procesadores se encuentran físicamente cerca y pueden comunicarse muy rápido. Los computadores paralelos son clasificados de acuerdo a como los procesadores acceden a la memoria:

Memoria compartida: Las arquitecturas de memoria compartida se caracterizan por que todos los procesadores pueden acceder a la memoria como un espacio global de direcciones. Múltiples procesadores pueden operar independientemente compartiendo el mismo espacio de memoria. Los cambios realizados en la memoria por un procesador pueden ser vistos por el resto de procesadores (ver Fig. 3.4(a)). Las arquitecturas de memoria compartida pueden ser clasificadas en:

UMA (Uniform Memory Access): Todos los procesadores acceden de forma directa y uniforme a memoria.

NUMA (Non-Uniform Memory Access): Uno a más procesadores poseen su propia memoria a la cual acceden como en un modelo *UMA* y el acceso a memoria de otros procesadores se realiza por medio de un bus de interconexión.

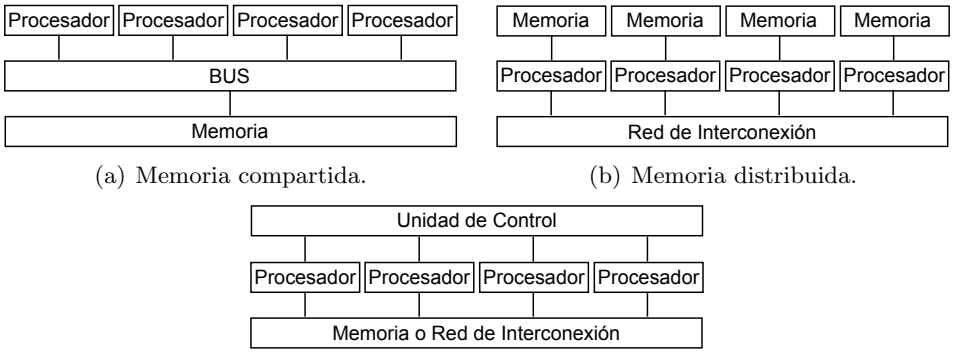
Memoria distribuida: Las arquitecturas de memoria distribuida se caracterizan por que todos los procesadores poseen su propia memoria independiente y se comunican entre ellos a través de una red de interconexión. Es necesario el uso de librerías de pasos de mensajes (*OpenMPI*, *Mpich2*,

LAM-MPI, etc), para la comunicación entre procesadores (ver Fig. 3.4(b)). Algunas de las arquitecturas distribuidas implementadas se pueden resumir en:

Clusters: Un cluster es un conjunto de computadores conectados a través de una red de interconexión. Cada máquina es bastante independiente a nivel de memoria o disco y usan una red de interconexión estándar (o una variación), para la comunicación entre procesadores.

MPP, Massive Parallel Processing: Un *MPP* o supercomputador está compuesto por cientos o miles de procesadores con memoria independiente conectados a través de una red de interconexión de alta velocidad.

Máquinas SIMD: SIMD (*Single Instruction Multiple Data*), este tipo de arquitectura se caracteriza por que una cada procesador ejecuta la misma instrucción sobre distintas secciones de memoria. La instrucción es asignada a cada procesador a través de una *unidad de control* que es la encargada de la sincronización de los procesadores (ver Fig. 3.4(c)).



(a) Memoria compartida.

(b) Memoria distribuida.

(c) SIMD.

Figura 3.4: Estructura de las arquitecturas de computación paralela.

En la actualidad, la gran mayoría de los computadores paralelos implementados se caracterizan por ser arquitecturas híbridas, es decir, *memoria compartida* intranodo y *memoria distribuida* internodo.

Computadores distribuidos

Un computador distribuido se define como un conjunto de computadores heterogéneos geográficamente distanciados conectados a través de una red *WAN*, como por ejemplo, *Internet*. Un computador distribuido puede ser visto como un *super computador virtual* en el que muchos computadores con características de

hardware diferentes e interconectados, actúan en conjunto para resolver trabajos que consumen largos periodos de tiempo. Las implementaciones más comunes de este tipo de sistemas serán descritas a continuación:

Grid: Un *grid* es un sistema a gran escala con máquinas heterogéneas distribuidas a través de múltiples organizaciones y dominios administrativos, conectadas a través de una red *WAN*.

Multiclusters: Un multicluster es un conjunto de clusters heterogéneos independientes, conectados a través de una red *WAN* dedicada, gestionado para su utilización como una plataforma integrada.

3.2.2. Procesos lógicos

Las estrategias simulación distribuida asíncrona típicamente consisten en descomponer el programa de simulación en un conjunto de procesos que pueden ejecutarse concurrentemente. Esta descomposición se realiza con el objetivo de explotar el paralelismo de los componentes del modelo.

Cualquier sistema en el que los componentes interactúan en pasos discretos de tiempo puede ser representado por medio de una red de procesos que se comunican por medio de mensajes. El sistema físico puede ser descompuesto en un conjunto de procesos físicos $(PF_1, PF_2, ..., PF_k)$, que se comunican entre ellos por medio de mensajes [7]. La simulación es construida por medio de un conjunto de procesos lógicos $(PL_1, PL_2, ..., PL_k)$ en el que cada uno de estos representa a un proceso físico $(PF_1, PF_2, ...PF_k)$ (ver Fig. 3.5). La comunicación entre procesos lógicos se realiza por medio de mensajes cronológicamente identificados.

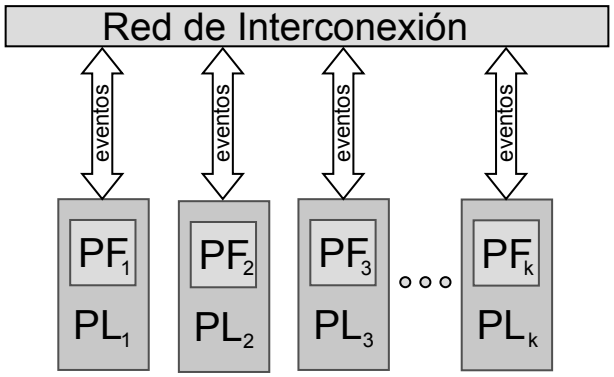


Figura 3.5: Paradigma de procesos lógicos.

La gran ventaja de este mecanismo es particionar las variables de estado del simulador en un conjunto de estados disjuntos y asegurar que los eventos del simulador no accedan directamente a más de un estado [18].

Restricción de la causalidad local

Una simulación de eventos discretos compuesta por un conjunto de procesos lógicos (PL) que interactúan por medio del intercambio de mensajes cronológicamente identificados cumple la restricción de la causalidad local si y sólo si cada PL no ejecuta los eventos cronológicamente ordenados.

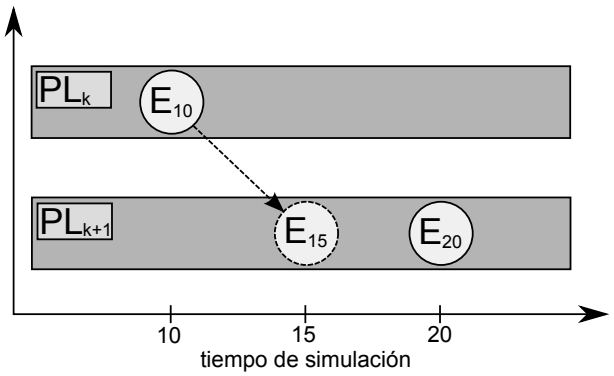


Figura 3.6: Ejemplo de violación de la causalidad local.

En la figura 3.6 se consideran dos eventos E_{10} perteneciente a PL_k con *timestamp* 10 y E_{20} perteneciente a PL_{k+1} con *timestamp* 20. Si E_{10} programa un nuevo evento E_{15} para PL_{k+1} con *timestamp* menor que 20, entonces E_{15} puede afectar a la ejecución de E_{20}. Por ejemplo, si ambos eventos modifican a la variable $x = 2$, E_{15} suma 1 a x y E_{20} divide por 2 a x, entonces, al ejecutar E_{20} antes que E_{15} se obtendría como resultado $x = 2$, pero el resultado correcto sería $x = 1{,}5$. En esta situación es necesaria la ejecución secuencial de los tres eventos. El cumplimiento de la restrición de la causalidad local asegura que la simulación distribuida produzca el mismo resultado que la ejecución secuencial.

3.2.3. Sincronización de procesos

Uno de los principales problemas en la simulación distribuida de eventos discretos es la sincronización de procesos lógicos. Esto se refiere a la ejecución cronológicamente ordenada de eventos y se produce debido a que cada proceso lógico es una unidad de simulación independiente. La ejecución no ordenada cronológicamente de eventos puede causar serios problemas en el resultado de la simulación distribuida, logrando una inconsistencia en los valores de las variables de estado. El principal objetivo de los mecanismos de sincronización de procesos es que la simulación distribuida entregue los mismos resultados que la ejecución secuencial.

Existen dos mecanismos ampliamente conocidos de sincronización de procesos: *Algoritmos conservativos* y *Algoritmos optimistas*. Estos dos mecanismos permi-

ten mantener la consistencia de la simulación distribuida aplicando estrategias para la ejecución de eventos. A continuación se explicarán ambos algoritmos.

Algoritmos conservativos

Históricamente, los algoritmos conservativos han sido los precursores en resolver el problema de la causalidad local en simulación distribuida. Estos mecanismos siempre aseguran la ejecución de eventos de simulación en orden cronológico en cada proceso lógico. Estos algoritmos estrictamente evitan la posibilidad de ocurrencia de cualquier error de causalidad local. Estos algoritmos consisten en cada PL bloquea la ejecución de eventos nuevos hasta que se asegure que en el futuro no se ejecutarán eventos cronológicamente pertenecientes al pasado.

En otras palabras, si un proceso lógico PL_1 recibe un evento E_1 cronológicamente identificado T_1 y es imposible recibir otro evento con cronológicamente menor a T_1, entonces PL_1 ejecuta el evento E_1. Si un otro proceso lógico PL_2 recibe un evento E_2 cronológicamente identificado T_2, y T_2 es mayor que T_1, entonces PL_2 bloquea su ejecución hasta que E_1 sea completado.

A pesar que esta aproximación evita completamente violaciones de la restricción de la causalidad local, es posible la ocurrencia de interbloqueo (*deadlock*). El interbloqueo se produce cuando existe un bloqueo cíclico entre procesos lógicos. A continuación se describirán los algoritmos más conocidos que permiten evitar el interbloqueo:

Mensajes nulos [6]: Un mensaje nulo es uno identificado con *timestamp* T_{null} enviado desde PL_k a PL_{k+1} con el objetivo de asegurar que en el futuro PL_k no enviará eventos a PL_{k+1} con *timestamp* menor a T_{null}. Cada PL envía mensajes nulos al resto de los PLs después de la ejecución de cada evento. Un mensaje nulo no implica la modificación de variables de estado pero si puede entregar información adicional para determinar que eventos son seguros para su ejecución.

El tiempo del próximo evento (TPE): El principal inconveniente de los algoritmo basados en mensajes nulos es que genera una gran cantidad de mensajes, incrementando sustancialmente el costo en comunicaciones. El algoritmo TNE computa, en cada procesador, la mayor cota inferior de los *timestamps* de mensajes de eventos en cada enlace vacío (un enlace vacío es aquel que no contiene mensajes). Se construye un grafo en el que cada nodo representan a un PL y las enlaces corresponden con el *tiempo del próximo evento*. El interbloqueo ocurre cuando existe un ciclo compuesto por enlaces vacíos en el que todos los enlaces poseen el mismo tiempo. El algoritmo TNE explora el grafo dirigido de los PL conectados por enlaces vacíos. El

proceso es ejecutado independientemente por cada PL evitando el costo en comunicaciones que implica la utilización de un algoritmo global

Algoritmos optimistas

Un gran problema en los algoritmos de sincronización conservativos es el *tiempo de espera* que se requiere cuando hay que bloquear la ejecución de uno o más procesos lógicos. A diferencia de los algoritmos conservativos que evitan la ocurrencia de errores de causalidad local, los algoritmos optimistas permiten que este tipo de errores sucedan.

El mecanismo *Time Warp* es uno de los algoritmo optimistas más conocidos. Cuando un PL recibe un evento cronológicamente identificado menor que uno o más eventos ya procesados, se realiza un *roll back*, es decir, se reejecutan los eventos en orden cronológico. El proceso de *roll back* consiste en reestablecer la consistencia de las variables de estado del PL a los valores que poseían antes de ejecutar el evento problemático.

Para ejecutar un *roll back* se deben realizar los siguiente procedimientos: *reestablecer el valor de las variables de estado*, para este proceso se pueden implementar mecanismos de *checkpointing* de estados o un almacenamiento incremental de estados, *cancelación de mensajes*, para este proceso se pueden implementar mecanismos de *anti-mensajes* y aniquilación de mensajes. Un *anti-mensaje* $-msg_k$ es idéntico a un mensaje positivo msg_k excepto por un bit de signo. Los *anti-mensajes* son utilizados para cancelar un mensaje. Cuando un *anti-mensaje* $-msg_k$ se encuentra con un mensaje positivo msg_k ambos se aniquilan.

Capítulo 4

Particionamiento y balance de carga

Uno de los principales problemas en simulación distribuida de modelos orientados al individuos espacialmente explícitos es la descomposición eficiente del dominio del problema y la asignación equitativa del resultado de la descomposición sobre las unidades de cómputo de la arquitectura paralela/distribuida.

Una de las primeras etapas en el desarrollo de simuladores distribuidos basados en modelos orientados al individuo es dividir la simulación en un conjunto de pequenas simulaciones que puedan ejecutarse concurrentemente. Este proceso es llamado *particionamiento* o *descomposición*, y se puede llevar a cabo por medio de dos enfoques distintos: *descomposición del dominio*, que consiste en dividir el dominio del problema en conjuntos disjuntos de individuos que puedan simularse independientemente y *descomposición funcional*, que consiste en dividir el cómputo asociado con la simulación en un conjunto disjunto de tareas que puedan ejecutarse concurrentemente.

Los métodos de particionamiento implementados en el desarrollo de simulaciones distribuidas de sistemas orientados al individuo, por lo general, están basados en la descomposición del dominio del problema. Existen dos clases de métodos de descomposición de dominio aplicables a sistemas orientado al individuo:

***Grid-based* o descomposición espacial:** Consiste en dividir el espacio de simulación en un conjunto de áreas geométricamente definidas, que contienen a los individuos que actualmente residen en cada área.

***Cluster-based* o descomposición por agrupación:** Consiste en agrupar individuos según algún criterio (proximidad, similitud, familiaridad, etc). La elección del criterio de agrupación va a depender directamente de la naturaleza del sistema y de la formulación del modelo.

El nivel de granularidad de un método de particionamiento especifica el grado de concurrencia en términos de procesamiento de particiones. Un método de particionamiento es considerado de *grano fino* si como consecuencia de la descomposición de dominio del problema se obtienen múltiples particiones de pequena magnitud. Un método de particionamiento es considerado de *grano grueso* si como consecuencia de la descomposición del dominio del problema se obtiene una cantidad limitada de particiones de gran magnitud.

Los métodos de particionamiento aplicables a sistemas orientados al individuo pueden ofrecer un doble beneficio en términos de rendimiento. Por un lado, un método de particionamiento eficiente permite explotar al máximo el uso de los recursos de cómputo/comunicación de la arquitectura paralela/distribuida. Por otro lado, un método de particionamiento eficaz permite descartar el cómputo involucrado en la detección de la posición *espacio-temporal* de individuos que se encuentran demasiado lejos del *área de interés* o *área de interacción* de un individuo en particular.

La segunda etapa para desarrollar una aplicación de simulación distribuida basada en un modelo orientado al individuo es establecer una política eficiente de *balance de carga* (cómputo y comunicación). Las políticas de balance de carga consisten en distribuir eficientemente las particiones (generadas en la etapa de descomposición del dominio del problema), sobre las unidades de cómputo de la arquitectura paralela/distribuida con la intención de lograr el mejor rendimiento posible (en términos de escalabilidad, eficiencia, etc).

Las políticas de balance de carga pueden ser clasificadas en *estáticas* y *dinámicas*. Una política de *balance de carga estática* es aquella en la cual la distribución de las particiones se efectúa *offline*, es decir, previa a la ejecución de la simulación distribuida. Este enfoque es útil cuando la carga de trabajo se mantiene constante a través de tiempo y las características de *hardware* de la arquitectura paralela/distribuida son conocidas de antemano. En contraste, en sistemas orientados al individuo en los cuales existe una alta variación con respecto a la carga de trabajo en función del tiempo, es imprescindible el uso de políticas de *balance de carga dinámicas*. Las simulaciones distribuidas de modelos orientado al individuo espacialmente explícitos que exhiben patrones de movimiento pueden presentar graves problemas al momento de ser ejecutadas en una arquitectura paralela/distribuida. Este tipo de simulaciones se caracterizan porque los individuos constantemente están cambiando su posición en el espacio de simulación, provocando migraciones progresivas de individuos entre procesos lógicos, y como consecuencia, la posible sobrecarga o infrautilización de los recursos de cómputo de la arquitectura paralela/distribuida.

Una política de *balance de carga dinámica* es aquella que establece mecanismos eficientes para equilibrar la carga de trabajo durante la ejecución de la simulación. Las políticas de balance de carga dinámico están formadas por dos rutinas:

detección del desbalance y *reajuste de la carga de trabajo*. La *detección del desbalance* puede ser llevada a cabo por medio de algoritmos basados en *umbrales*, *tiempos de ejecución*, *probabilísticos*, etc. El *reajuste de la carga de trabajo* puede ser llevado a cabo por medio de migración de individuos entre procesos lógicos o, en el peor de los casos, reparticionando el dominio del problema y redistribuyendo las particiones.

A continuación se realizará una revisión de las principales estrategias de particionamiento aplicables a sistemas orientados al individuos espacialmente explícitos, clasificadas en dos grupos: *grid-based* y *cluster-based*. También se describirán las políticas de balance de carga que pueden se aplicadas sobre dichos métodos de particionamiento.

4.1. Descomposición espacial, *grid-based*

Esta estrategia consiste en dividir el espacio de simulación en un conjunto de áreas geométricamente definidas que contienen a los individuos que residen actualmente en cada región. La forma y tamano de cada área va a depender directamente de la naturaleza del sistema y de la formulación del modelo.

Las particiones rectangulares representan una alternativa de solución para múltiples problemas computacionales, como por ejemplo: *procesamiento de imágenes*, *indexación en bases de datos no tradicionales*, *multiplicación de matrices de grandes magnitudes*, etc. Este tipo de descomposición consiste en dividir el espacio de simulación por medio de *rectas o planos* de corte ortogonales a los ejes/planos coordenados (dependiendo la dimensionalidad del problema). Algunos de los métodos de descomposición basados en particiones rectangulares más conocidos pueden ser clasificados en: *particiones rectilíneas* [51], *particiones dentadas* y *biparticiones jerárquicas* [61].

4.1.1. Particiones rectilíneas

Los métodos de descomposición basados en particiones rectilíneas consisten en dividir el espacio de simulación en por medio de cortes homogéneos realizado con rectas o planos de cortes ortogonales a los ejes/planos coordenados.

Por ejemplo, si se quisiera particionar un espacio de simulación bidimensional delimitado por $(x_1, x_2), x_1 < x_2$ e $(y_1, y_2), y_1 < y_2$ en 4 áreas homogéneas, el procedimiento sería el siguiente: $x' = \frac{x_2 - x_1}{2}$, $y' = \frac{y_2 - y_1}{2}$, tal que, se obtienen 4 áreas homogéneas constituidas por $a_1 = (x_1, x')(y_1, y'), a_2 = (x_1, x')(y', y_2), a_3 = (x', x_2)(y_1, y'), a_4 = (x', x_2)(y', y_2)$.

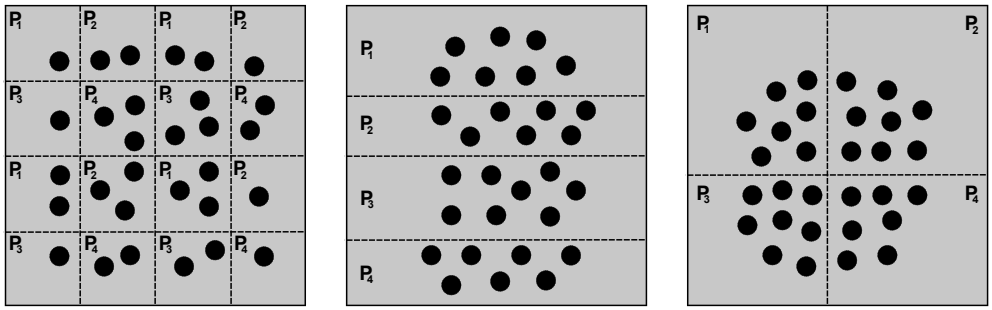
Los métodos de descomposición basados en particionamiento rectilíneo más conocidos se pueden clasificar en:

Particionamiento disperso: El particionamiento disperso [52, 12] es un

tipo de descomposición de grano fino el cual consiste en dividir el espacio de simulación en un conjunto áreas rectangulares del mismo tamaño y forma. En la figura 4.1(a) se muestra un ejemplo de este tipo de particionamiento, en el que el espacio de simulación es dividido en 16 áreas asignadas a las unidades de cómputo: P_1 P_2 P_3 y P_4 las cuales contienen respectivamente 8 7 7 y 8 individuos.

Particionamiento en franjas: El particionamiento en franjas [48, 12] es un tipo de descomposición de grano grueso que consiste en dividir el espacio de simulación en franjas rectangulares con la misma (o similar) cantidad de individuos. Este método consiste en dividir una de las dimensiones del espacio de simulación con el objetivo de obtener tantas franjas como unidades de cómputo. La ventaja de este método es que cada franja comparte fronteras sólo con sus dos franjas vecinas (a excepción de los extremos). En la figura 4.1(b) se muestra un ejemplo de este tipo de particionamiento. Como resultado de la aplicación de este método se obtienen 4 franjas asignadas a las unidades de cómputo: P_1 P_2 P_3 y P_4, las cuales contienen respectivamente 7 7 8 y 7 individuos.

Particionamiento en bloques: El particionamiento en bloques [11] es un tipo de descomposición de grano grueso que consiste en dividir el espacio en bloques rectangulares heterogéneos u homogéneos. Los bloques se construyen por medio de dividir las dimensiones del espacio de simulación al menos una vez. En la figura 4.1(c) se muestra un ejemplo de este tipo de particionamiento, en el que el espacio es dividido una vez a través del eje x y una vez a través de eje y. Como resultado se obtienen 4 bloques asignados a las unidades de cómputo: P_1 P_2 P_3 y P_4, las cuales contienen respectivamente 7 8 8 y 7 individuos



(a) Disperso. (b) Franjas. (c) Bloques.

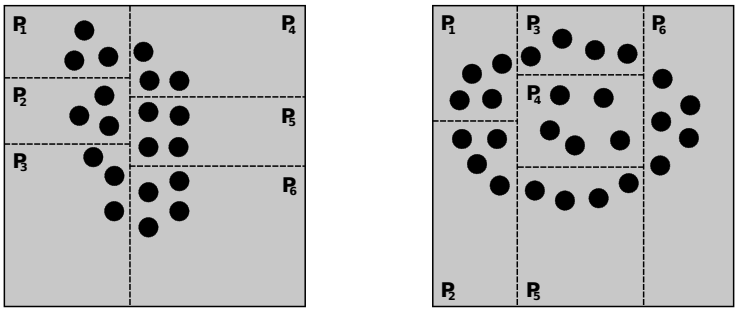
Figura 4.1: Ejemplos de particionamiento rectilíneo.

4.1.2. Particiones dentadas

Los métodos de descomposición basados en particiones dentadas se caracterizan por distinguir una *dimension principal* y una *dimension auxiliar*. La dimensión principal es dividida en P intervalos y cada intervalo es dividido independientemente los otros. Las particiones dentadas pueden ser clasificadas en dos tipos:

Particiones P Q-way: En este tipo de descomposición la *dimension principal* es dividida en P intervalos y cada uno de los P intervalos es dividido independientemente en otros Q intervalos. En la figura 4.2(a) se muestra un ejemplo de este tipo de particionamiento en el que la dimensión principal x es divida en $P = 2$ intervalos, y cada uno de estos P intervalos es dividido en otros $Q = 3$ intervalos. Cada una de las particiones es asignada a las siguientes unidades de cómputo: P_1 P_2 P_3 P_4 P_5 y P_6, las cuales contienen respectivamente 3 3 3 3 4 y 4 individuos.

Particiones *m-way*: En este tipo de descomposición la dimensión principal es dividida en P intervalos y cada P intervalo puede ser dividido independientemente en tantas particiones como sean necesarias. En la figura 4.2(b) se puede ver un ejemplo de este tipo de particionamiento, en el que la dimensión principal x es dividida en $P = 2$ intervalos y cada uno de los P intervalos es dividido en 2 3 y 1 intervalos respectivamente. Cada una de las particiones es asignada a las siguientes unidades de cómputo: P_1 P_2 P_3 P_4 P_5 y P_6, las cuales contienen respectivamente 4 4 4 5 4 y 5 individuos.



(a) Particionamiento del tipo P Q-*way*

(b) Particionamiento del tipo *m-way*

Figura 4.2: Ejemplos de particiones dentadas.

4.1.3. Biparticiones jerarquica

Las técnicas de bipartición jerárquica, basadas en algoritmos de bisección recursiva [4], consisten en obtener particiones que pueden ser recursivamente generadas dividiendo una de las dimensiones en dos intervalos [61]. Las técnicas de bisección recursiva son utilizadas para particionar el dominio un problema en *sub-dominios* que posean aproximadamente el mismo costo computacional mientras se intenta minimizar el costo en comunicaciones. El algoritmo consiste en dividir una de las dimensiones del dominio con el objetivo de obtener dos *sub-dominios*. Luego, se dividen recursivamente a los *sub-dominios* con el objetivo de tener tantos *sub-dominios* como se requieran.

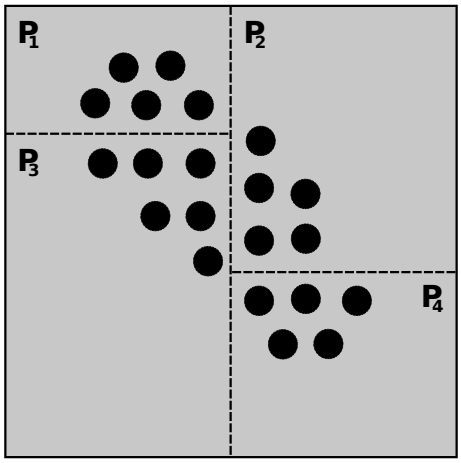


Figura 4.3: Particionamiento recursivo.

En la figura 4.3 se muestra un ejemplo de este tipo de particionamiento, en donde inicialmente se realiza un corte ortogonal al eje y y, posteriormente se un realiza corte en cada *sub-dominio* ortogonal al eje x. Como resultado se obtienen 4 particiones asignadas a las unidades de cómputo: P_1 P_2 P_3 y P_4, las cuales contienen respectivamente 5 5 6 y 5 individuos.

4.2. Descomposicion por agrupacion, *cluster-based*

Otro enfoque con el cual es posible descomponer el dominio de un sistema orientado al individuo espacialmente explícito es por medio de *particiones compactas*. Estos algoritmos consisten en dividir el espacio en un conjunto de zonas definidas a partir de un objeto de referencia llamado *centro* o *centroide*. Estos algoritmos son aplicables sobre espacios vectoriales en donde existe la noción de distancia entre objetos, como es el caso de *espacios metricos*.

Un espacio métrico es un tipo de espacio vectorial en el cual la distancia entre objetos está definida. Dado un conjunto de objetos $\mathbb{X}$ subconjunto de universo de objetos válidos $\mathbb{U}$ y una función de distancia $d : \mathbb{X}^2 \to \mathbb{R}$, $\forall x, y, z \in \mathbb{X}$, tal que:

$$\textbf{Positividad:} \quad d(x, y) > 0,\ d(x, x) = 0 \tag{4.1}$$

$$\textbf{Simetría:} \quad d(x, y) = d(y, x) \tag{4.2}$$

$$\textbf{Desigualdad triangular:} \quad d(x, y) + d(y, z) \geq d(x, z) \tag{4.3}$$

Entonces, se puede decir que $(\mathbb{X}, d)$ es un espacio métrico. Suponiendo que el dominio de un sistema orientado al individuo espacialmente explícito está compuesto por un conjunto de individuos $\mathbb{I}$, asociados a una posición continua en el espacio euclidiano ($2D$ o $3D$) y se utiliza como sensor de visibilidad a la función v (distancia euclidiana), entonces se puede decir que $(\mathbb{I}, v)$ es un espacio métrico.

Existen dos criterios con los cuales se puede descomponer un espacio métrico: *Criterio del radio cobertor* y *Diagramas de Voronoi*. Ambos criterios serán descritos a continuación.

4.2.1. Criterio del radio cobertor

El criterio del radio cobertor consiste en dividir el espacio en un conjunto de esferas que pueden intersectarse entre ellas. Las esferas son construidas a partir de un *radio cobertor* y si la distancia $d(q, c_i)$ entre un objeto arbitrario q y un centroide c_i del *cluster* $[c_i]$ es menor o igual que el radio cobertor r_i de cluster $[c_i]$, entonces q pertenece a $[c_i]$.

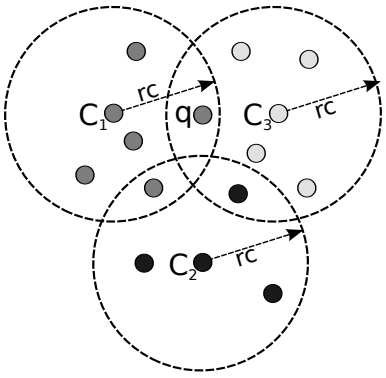


Figura 4.4: Criterio del radio cobertor.

En la figura 4.4 se muestra un ejemplo de aplicación del criterio del radio cobertor, donde son generados tres *clusters* $[c_1], [c_2]$ y $[c_3]$ a partir de los centroides

c_1, c_2 y c_3, respectivamente. Se puede apreciar que el objeto q, el cual se encuentra contenido en los *clusters* $[c_1]$ y $[c_3]$, es asignado al *cluster* $[c_1]$. Esto sucede debido a que las evaluaciones de distancia entre q y el conjunto de *clusters* se realizan de forma ordenada ascendentemente por identificador de *cluster* y la distancia entre q y c_1 cumple la condición para que q pertenezca a $[c_1]$.

4.2.2. Diagramas de Voronoi

Los *diagramas de Voronoi* son una de las estructuras fundamentales en geometría computacional. Dado un cierto número de puntos en el plano, su diagrama de *Voronoi* divide el plano de acuerdo con la regla del *vecino más cercano*: cada punto es asociado con la región del plano más cercana a él [2].

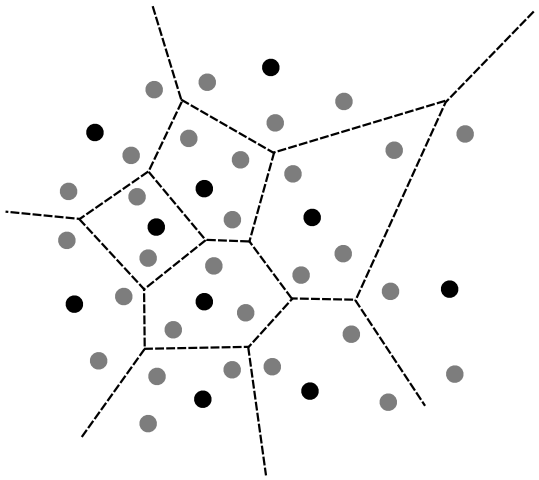


Figura 4.5: Diagrama de *Voronoi*.

En otras palabras, el método consiste en seleccionar un conjunto de puntos en el espacio (subconjunto del dominio del problema), usualmente llamados *semillas* o *puntos generadores*. Cada *semilla* representa el centro de masa de su *celda*. Cada *celda* es un polígono convexo que contiene los puntos más cercanos a su *semilla* (ver Fig. 4.5).

4.2.3. K-means

Uno de los métodos más populares para crear agrupaciones de objetos es el llamado *k-means* [44]. El nombre del algoritmo proviene de que se seleccionan k muestras las cuales serán los centroides de k *clusters* $[c_i]$, se asignan los objetos restantes a cada *cluster* y se calcula la media por *cluster*, se reemplazan los centroides por los más cercanos a la media y se reasignan los objetos restantes.

El principal objetivo de este método es seleccionar como centroides de *cluster* a los objetos más representativos (los que están más en el centro). El resultado de la aplicación de este método es un *diagrama de voronoi*. El algoritmo consiste en los siguientes pasos:

(1) Seleccionar k centroides de *cluster* de forma aleatoria pertenecientes a la dominio del problema.

(2) Asignar el resto de los objetos del dominio del problema a su centroide más cercano.

(3) Recomputar los centroides de *clusters* utilizando los objetos que contiene el *cluster*.

(4) Si la condición de convergencia no se cumple, volver a paso (2).

4.3. Balance de carga

Una aplicación de simulación distribuida basada en un *MoI* está compuesta por una colección de *procesos lógicos*, cada uno de los cuales está formado por un conjunto disjunto de individuos pertenecientes al dominio del problema. Un *proceso lógico* es un elemento de concurrencia dentro de la simulación distribuida responsable del procesamiento de sus individuos sobre una unidad de cómputo, que se comunica con otros procesos lógicos por medio del *intercambio de mensajes*.

En el desarrollo de aplicaciones de simulación distribuida basadas en *MoI* pueden distinguirse dos principales etapas: el *particionamiento del dominio del problema* (descrito en la sección anterior) y *distribución de las particiones sobre las unidades de cómputo de la arquitectura paralela/distribuida*. Cada proceso lógico es asignado a una única unidad de cómputo y está formado por un conjunto de particiones no vacías. El principal *objetivo de optimización* en la fase de distribución de particiones consiste en equilibrar la carga de trabajo entre procesos lógicos minimizando la comunicación entre estos.

En la figura 4.6 se expone la clasificación de los algoritmos de balance de carga. Estos algoritmos pueden clasificarse principalmente en *estáticos* y *dinámicos*. Los algoritmos de *balance de carga estático* son aquellos en los cuales el equilibrio de la carga de trabajo se efectúa por medio de un proceso *offline*, es decir, previa a la ejecución de la simulación distribuida. Estos tipo de algoritmos son aplicables sobre simulaciones de sistemas con características predecibles, es decir, en aquellos en que la carga de trabajo se mantiene constante a medida que la ejecución de la simulación distribuida progresa a través del tiempo.

En contraste, los algoritmos de *balance de carga dinámicos* son aquellos en los cuales la carga de trabajo debe ser ajustada dinámicamente durante la ejecución de la simulación distribuida para cumplir el *objetivo de optimización*. Este tipo de

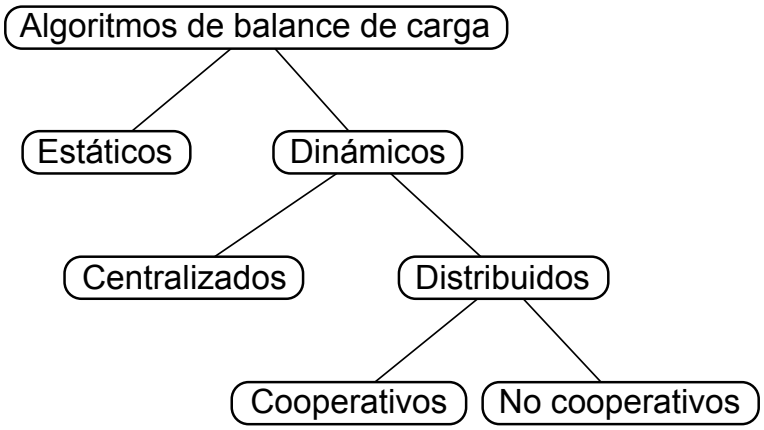


Figura 4.6: Clasificación de los algoritmos de balance de carga.

algoritmos son aplicables sobre simulaciones de sistemas cuyo comportamiento provoca que la cantidad de la carga de trabajo asignada a las unidades de cómputo varíe a medida que la ejecución de la simulación distribuida progresa a través de tiempo. Estas variaciones son las responsables de producir la *sobrecarga* o *infrautilización* de una o más unidades de cómputo, lo que acarrea como consecuencia una progresiva degradación del rendimiento global de la aplicación.

Un ejemplo específico de este tipo de simulaciones son las basadas en *MoI* espacialmente explícitos que exhiben patrones de movimiento. El movimiento intrínseco exhibido en este tipo de sistemas puede provocar migraciones progresivas de individuos entre particiones y/o procesos lógicos, desencadenando un inminentemente desequilibrio. Típicamente, los algoritmos de balance de carga dinámicos constan de dos procesos fundamentales: *detección* del desequilibrio en la carga de trabajo y *re-ajuste* de la carga de trabajo. El proceso de *detección* es la pre-condición que indica cuando debe realizarse el *re-ajuste*. El proceso de *re-ajuste* es encargado de re-establecer el equilibrio de la carga de trabajo asignada a las unidades de cómputo.

4.3.1. Algoritmos de balance de carga estáticos

Los algoritmos de balance de carga estáticos son aquellos en los cuales la distribución de *particiones* se lleva a cabo previa a la ejecución de la simulación distribuida. Estos algoritmos asumen cierto grado de conocimiento acerca de la aplicación y de la arquitectura en donde se ejecuta. Los individuos asignados a una partición siempre serán procesados en la misma unidad de cómputo. El principal objetivo de un mecanismo de balance de carga estático es equilibrar la carga asignada a las unidades de cómputo, y minimizar los tiempos comunicación y de espera. Algunos de los algoritmos de balance de carga más conocidos serán

descritos a continuación:

Algoritmos basados en *round robin*: Estos algoritmos consisten en distribuir las particiones de forma ordenada y cíclica. En otras palabras, la partición i es asignada a la unidad de cómputo ($i \,\%\mathbb{NPROCS}$).

Algoritmos aleatorios: Estos algoritmos consisten en distribuir las particiones de forma aleatoria.

***Simulated annealing*:** *Simulated annealing* [33] es un método meta-heurístico que permite encontrar un óptimo global aproximado de una función objetivo que posee un enorme conjunto de óptimos locales. Este algoritmo es utilizado para encontrar una solución óptima aproximada para la de distribución de múltiples particiones sobre un conjunto de unidades de cómputo teniendo como objetivo equilibrar la carga de trabajo y minimizar las comunicaciones [3].

Algoritmos genéticos: Los algoritmos genéticos son métodos heurísticos adaptativos de búsqueda u optimización basados en mecanismos evolutivos observados en sistemas naturales. Estos algoritmos son utilizados para resolver los problemas de optimización involucrados en la distribución de particiones y el balance de carga [21].

La gran ventaja de los algoritmos de balance de carga estáticos es que no provocan *overhead* durante la ejecución de la simulación distribuida. La utilización de algoritmos de balance de carga estáticos es conveniente cuando se pueden estimar los tiempos de ejecución por *proceso lógico* y los requisitos de comunicación entre estos.

4.3.2. Algoritmos de balance de carga de dinámicos

Los algoritmos de *balance de carga dinámico* son aquellos en los cuales la carga trabajo es ajustada dinámicamente durante la ejecución de la aplicación paralela/distribuida. El principal objetivo de un mecanismo de balance de carga dinámico es que cada unidad de cómputo de la arquitectura paralela/distribuida procese un parte equitativa de la carga total. El ajuste de la carga de trabajo se lleva a cabo por medio de la migración de tareas entre unidades de cómputo *sobrecargadas* y unidades de cómputo *infautilizadas*.

En simulación distribuida de MoI, los métodos de balance de carga dinámico son los encargados de equilibrar el número de individuos asignados a los procesos lógicos en tiempo ejecución. El problema del balance de carga en simulación distribuida de MoI surge cuando las relaciones entre componentes del sistema simulado son lo suficientemente complejas para provocar comportamientos globales impredecibles.

Un ejemplo pragmático de este tipo de simulaciones son aquellas basadas en MoI espacialmente explícitos que exhiben patrones de movimiento. Generalmente, este tipo de modelos están basados en un conjunto de reglas de comportamiento determinísticas influenciadas por componentes aleatorios. En este tipo de sistemas, los cambios de posición de individuos en el espacio de simulación pueden provocar migraciones progresivas de individuos entre *procesos lógicos*, causando un inminente desbalance en la carga de trabajo.

Los algoritmos de balance de carga dinámicos pueden clasificarse en: *centralizados*, en los cuales la decisión de ejecutar el mecanismo de balance de carga es realizada por una única unidad de cómputo o *distribuidos*, en los cuales la decisión de ejecutar el mecanismo de balance de carga es compartida entre varias unidades de cómputo. Las estrategias de balance de carga distribuidas pueden clasificarse en: *cooperativas*, en las cuales todas las unidades de cómputo trabajan en conjunto para lograr un equilibrio global de la carga de trabajo o *no cooperativas*, el las cuales todas las unidades de cómputo trabajan autónomamente para lograr el objetivo de equilibrio de la carga de trabajo.

Algunos de los métodos de balance de carga dinámico más conocidos serán descritos a continuación:

***Gradient model*:** El modelo gradiente [37] es un método de balance de carga localizada en el que cada unidad de cómputo interactúa solamente con sus vecinos inmediatos. El balance de carga global es logrado por medio de propagación y refinamiento sucesivo de la carga de trabajo local.

***Sender initiated diffusion* SID:** SID es un método de balance de carga distribuido en el que cada unidad de cómputo determina, a partir de su carga de trabajo local y la media de la carga de trabajo global, si está siendo sobrecargada o infrautilizada. El balance de carga global se lleva a cabo por medio de la difusión del excedente de la carga de trabajo de unidades de cómputo sobrecargadas a las unidades de cómputo infrautilizadas más cercanas.

***Receiver initiated diffusion* RID:** RID es un método de balance de carga que utiliza una estrategia inversa a la de SID. Las unidades de cómputo infrautilizadas realizan peticiones de carga de trabajo a las unidades de cómputo sobrecargadas más cercanas para equilibrar su carga local.

***Hierarchical balancing method* HBM:** La estrategia HBM [75] consiste en organizar la arquitectura paralela/distribuida en una jerarquía de sub-dominios de tal manera de descentralizar el proceso de balance de carga. Un conjunto de unidades de cómputo son designadas como controladores de operaciones de balance de carga en diferentes niveles de la jerarquía. El balance de carga global se logra recorriendo el árbol de jerarquías (desde las

hojas hasta la raíz), balanceando la carga entre sub-dominio adyacentes en cada nivel de la jerarquía.

***Dimension exchange method* DEM:** La estrategia DEM [13] es muy similar a HBM, ya que ambas se caracterizan por organizar la arquitectura paralela/distribuida en un árbol de jerarquía. La principal diferencia entre DEM y HBM recae en que DEM ejecuta el balance de carga de forma sincronizada en cada nivel del árbol de jerarquía (desde las hojas hasta la raíz).

Los algoritmos de balance de carga dinámicos se caracterizan por estar formados por dos componentes fundamentales: la *detección* del desbalance de carga de trabajo y el *re-ajuste* de la carga de trabajo.

4.3.3. Detección del desbalance

La primera etapa involucrada en la incorporación de mecanismos de *balance de carga dinámicos* en el desarrollo de aplicaciones de simulación distribuida es el proceso de *detección del desbalance* entre la carga de trabajo asignada a las diferentes unidades de cómputo comprometidas en la ejecución de la simulación distribuida. Este proceso es el encargado de determinar en que punto de la ejecución de la simulación distribuida se debe invocar al proceso de *re-ajuste* la carga de trabajo.

Este proceso de *detección* es el encargado de identificar cuando una unidad de cómputo está siendo *sobrecargada* o *infrautilizada*, y cuantificar el desequilibrio de la carga de trabajo. La detección del desbalance no es un proceso simple ya que se debe determinar con precisión cuando el desbalance influye efectivamente en el desempeno global de la simulación distribuida. Además, se debe tener en cuenta que como resultado de este proceso se invocarán procesos de *re-ajuste* de la carga de trabajo, los cuales involucran cómputo y comunicaciones que pueden influir en el desempeno global de la simulación distribuida en un punto especifico de su ejecución.

Detección basada en umbrales

Una forma simple de determinar cuando una unidad de cómputo está siendo *sobrecargada* o *infrautilizada* es por medio de *umbrales* [9, 16]. De esta manera, se fijan los rangos *mínimo* y *máximo* de individuos que una única unidad de cómputo debe alojar como condición para continuar con la ejecución normal de la simulación distribuida.

En otras palabras, si se tiene un total de N individuos distribuidos sobre P unidades de cómputo, tal que cada unidad de cómputo debe poseer una cantidad de individuos cercana a la media $\frac{N}{P}$, y se define como umbral de desbalance $ud\,\%$,

tal que, $min = np - (np * ud\,\%), max = np + (np * ud\,\%)$, entonces, la cantidad de individuos por unidad de cómputo np debe mantenerse entre $min \geq np \leq min$, de lo contrario, debe ejecutarse un *re-ajuste*.

Lo más difícil de establecer un método de detección basado en umbrales es determinar el tamano del umbral. Esto puede depender de muchas variables independientes, tales como: velocidad del procesador, tamano de la memoria, tamano del problema, la naturaleza de la aplicación, etc.

Detección basada en probabilidades

A diferencia de los métodos basado en umbrales, los cuales ejecutan una evaluación de la carga de trabajo en cada iteración de la simulación distribuida, los métodos basados en probabilidades determinan cual es la posibilidad de que un desbalance de carga ocurra, es decir, realizan una predicción del comportamiento de la simulación distribuida basada en las características del modelo. Especificar la función de probabilidad es una tarea bastante difícil, ya que depende directamente de la naturaleza del modelo de simulación. Además, la función de probabilidad puede tener la capacidad ajustase dinámicamente a medida que la ejecución de la simulación distribuida avanza con el objetivo de lograr resultados de predicción más precisos.

Detección basada en tiempos de ejecución

Otro método que permite detectar el desbalance de la carga de trabajo es por medio del análisis de los tiempos de ejecución entregados por la simulación distribuida. Estos métodos permiten detectar cuando un desbalance de carga de trabajo ocurre dependiendo de los cambios en los tiempos de ejecución en cada unidad de cómputo. Para implementar este tipo de métodos de necesita recopilar información de ejecuciones previas para poder alimentar a la base de datos de conocimiento. Adicionalmente, existe la posibilidad de alimentar dinámicamente a la base de datos de tiempos de ejecución a medida que progresa la simulación distribuida para lograr resultados más precisos

4.3.4. Re-ajuste de la carga de trabajo

A continuación se describirán las estrategias aplicables sobre los métodos de particionamiento basados en descomposición espacial y por agrupación para lograr balance de carga dinámico en simulaciones distribuidas de MoI espacialmente explícitos que exhiben patrones de movimiento.

Particiones *grid-based*

Los métodos de particionamiento basados descomposición espacial se caracterizan por ser generados a partir de *rectas o planos de corte* ortogonales a los ejes/planos coordenados. A medida que la ejecución de la simulación distribuida avanza en el tiempo, pueden producirse migraciones progresivas de individuos entre procesos lógicos y en consecuencia surge un desequilibrio en la carga asignada a las unidades de cómputo. Estas migraciones de individuos son causadas, por ejemplo, por el movimiento de *individuos* o *agentes* a través del ambiente en una simulación.

La solución en particiones rectilíneas de *grano grueso* es mover los límites de las particiones, es decir, establecer nuevas *rectas o planos de corte* que satisfagan las condiciones de equilibrio en la carga de trabajo. En otras palabras, ofrecer un grado de dinamismo en términos de límites de particiones a medida que progresa la ejecución de la simulación. Volver a establecer los límites de las particiones implica la migración de individuos entre procesos lógicos con el objetivo de mantener la coherencia en la ejecución de la simulación distribuida.

En particiones rectilíneas de *grano fino*, como es el caso del *particionamiento disperso*, reestablecer los límites no es una solución viable, ya que condicionalmente cada celda debe tener el mismo tamano y forma. En este caso, la solución consiste en la migración de particiones (*celdas*) entre procesos lógicos con el objetivo de equilibrar la carga de trabajo asignada a las unidades de cómputo. Cabe mencionar que en este tipo de particionamiento no es factible el uso de mecanismos de balance de carga estáticos, tales como: *round-robin* o *métodos de distribución aleatorios*, ya que su utilización acarrearía un inmenso costo en términos de comunicación. Este costo es causado debido a que cada individuo debe evaluar a los individuos pertenecientes a su vecindario y la utilización de este tipo de algoritmos dispersaría de forma fragmentada a los vecindarios sobre las unidades de cómputo de la arquitectura paralela/distribuida. Una aproximación viable consiste en *agrupar* (o *re-agrupar* en balance de carga dinámico), celdas con un criterio de adyacencia con el objetivo de mantener grupos aglomerados de pequenas particiones.

Particiones *cluster-based*

Cuando se particiona el dominio de un problema en base a un criterio de *clustering*, por lo general, se obtienen más particiones que unidades de cómputo. En consecuencia, se puede clasificar como una descomposición de *grano fino*. La idea central de los algoritmos de *clustering* es tener cierta noción de proximidad o similitud entre particiones con el objetivo de descartar rápidamente una zona al momento de hacer una consulta. El criterio de proximidad o cercanía es definido por medio de una función de distancia.

El resultado de este tipo de descomposición debe almacenar en una estructura de alto nivel que permita administrar dinámicamente los individuos. Este tipo de estructuras, por lo general, utilizan una tabla de distancia en la cual se almacenan las distancias entre centroides. Distribuir este tipo de estructuras debe ser lo suficientemente eficiente con el objetivo de representar la proximidad o similitud inherente en el método de particionamiento.

La solución típica para este tipo de problemas es siempre explotar la proximidad. En otras palabras, muchas particiones pequenas pueden agruparse por proximidad en conjunto de P *meta-particiones*, en donde P es la cantidad de unidades de cómputo. De esta manera se establece una política eficiente de *balance carga estático*.

Por otro lado, a medida que la ejecución de la simulación distribuida avanza, pueden provocarse migraciones de individuos entre *clusters* locales o, peor aún, entre *clusters* locales y remotos. El resultado de estas migraciones es el responsable de ocasionar problemas de desequilibrio en carga de trabajo asignada a las unidades de cómputo de la arquitectura paralela/distribuida, provocando un decremento en el desempeno global de la aplicación. La solución a este tipo de problemas recae, nuevamente, en el concepto de proximidad. Una vez detectado el problema, se debe realizar la *re-agrupación* de las múltiples *particiones* en P *meta-particiones* que satisfagan la condición de equilibrio en la carga de trabajo. Esta *re-agrupación* acarrea comunicaciones en términos de migración de *particiones* entre unidades de cómputo, para lograr continuar consistentemente la ejecución de la simulación distribuida.

Capítulo 5

Simulador distribuido de bancos de peces

En el presente capítulo se describirán en detalle los mecanismos utilizados para la implementación de un simulador distribuido de bancos de peces basado en el modelo orientado al individuo de *Huth & Wissel*. El simulador implementa un mecanismo de progreso de tiempo basado en *time-stepped*, debido a que las actualizaciones de las posiciones y orientaciones de todos los individuos se debe realizar coordinadamente en la misma fracción de tiempo. Además, se implementa un mecanismo de sincronización *conservativo* de procesos, debido a que cada proceso lógico necesita conocer las posiciones y orientaciones actualizadas de los individuos pertenecientes a las fronteras de procesos lógicos adyacentes antes de ejecutar un nuevo paso de simulación.

La etapa preliminar en el desarrollo del simulador consiste en una adaptación del modelo de *Huth & Wissel* con el objetivo de representar un banco de peces en un ambiente tridimensional. Esto implica decrementar el nivel de abstracción del modelo, acercándolo un poco más a la realidad, y anadiendo el costo computacional que involucra incluir una tercera dimensión.

La gran mayoría de los modelos orientados al individuo describen la reacción de un único individuo influenciado por otros individuos pertenecientes a cierta *área de visibilidad* o *área de interés*. Los algoritmos tradicionales de simulación de este tipo de modelos efectúan cómputo exhaustivo de *todos contra todos* con el objetivo de encontrar a los vecinos más influyentes. En el presente trabajo se ha propuesto desarrollar un método de particionamiento que permita descartar la evaluación de la posición *espacio-temporal* de los individuos que se encuentran demasiado lejos. Para esto se ha introducido el concepto de *similitud* o *proximidad* asociado a *espacios métricos*. Pero, Cómo se puede modelar la similitud en un banco de peces?. La pregunta no es fácil de responder, mas bien, depende de como esté formulado el modelo.

El modelo de *Huth & Wissel* es un modelo de espacio continuo en el que los individuos están asociados a una posición en el espacio euclidiano tridimensional. Estos individuos representan al conjunto de puntos que constituyen el dominio del problema. Por otro lado, el sensor de visibilidad de un pez es modelado con la distancia euclidiana, la cual cumple las propiedades de: positividad, simetría y desigualdad triangular. Por lo tanto, el conjunto de peces más el sensor de visibilidad forman une espacio métrico.

El método de particionamiento desarrollado en el presente trabajo consiste en formar pequenas aglomeraciones de peces en función del máximo *rango de visibilidad* y que pueden o no intersectarse unas con otras. Este método consiste en formar un conjunto de *particiones compactas* generadas a partir de un criterio híbrido de descomposición basado en el *criterio del radio cobertor* y *diagramas de voronoi*. Estas particiones son denominadas *clusters* y son formadas a partir de un elemento de referencia llamado *centro* o *centroide* más los elementos que pertenecen a su zona. Esta estrategia de *clustering* ha permitido reducir la complejidad del algoritmo secuencial de $O(n^2)$ a $O(mk)$, donde k es la cantidad de *clusters* y m la cantidad de individuos por *cluster*.

La *lista de clusters* es una estructura de datos utilizada para la búsqueda por similitud en espacios métricos. Esta estructura está formada por una lista en enlazada de *clusters*. Gracias a la simplicidad de esta estructura es posible administrar eficientemente los *clusters* obtenidos en el proceso de particionamiento. Esta *lista de clusters* permite almacenar cierta información relevante que permitirá descartar rápidamente *clusters* que se encuentran demasiado lejos durante el proceso de simulación, como por ejemplo: una *tabla de distancias*, que almacena las distancias desde un centroide al resto y/o el *radio cobertor*, que corresponde con la distancia desde un centroide al elemento más alejado perteneciente a su zona.

Distribuir este tipo de simulaciones debe ser lo suficientemente eficiente para minimizar las comunicaciones involucradas entre procesos lógicos y asignar equitativamente la carga de trabajo sobre las unidades de cómputo. En particiones de grano fino, como es el caso del método de particionamiento propuesto en el presente trabajo, una distribución inadecuada de las particiones puede incrementar las comunicaciones de forma explosiva. El método de distribución de las particiones sobre las unidades de cómputo propuesto en el presente trabajo consiste en la formación de un conjunto de *meta-clusters*, es decir, aglomeraciones de *clusters* asociados por proximidad. Este proceso consiste en la construcción de $\mathbb{NPROCS}$ *meta-clusters* con una cantidad de individuos cercana a la media $\frac{individuos}{\mathbb{NPROCS}}$. Cada uno de estos *meta-clusters* es asignado a una unidad de cómputo.

En simulaciones de modelos orientados al individuos espacialmente explícitos que exhiben patrones de movimiento ejecutadas en arquitecturas paralelas/distribuidas, es común que a medida que la simulación progrese se produzcan problemas de desbalance de carga. Estos problemas son causados principalmente

por la migración de individuos entre procesos lógicos. La solución de este tipo de inconvenientes se lleva a cabo introduciendo mecanismos de *balance de carga dinámico*. En el presente trabajo se ha desarrollado una política de balance de carga dinámico basada en la re-configuración de los *meta-clusters*. El proceso de detección de desbalance de la carga de trabajo se realiza por medio de un mecanismo de detección *basado en umbrales*, es decir, se definen cierto rangos en los cuales la aplicación distribuida puede continuar su ejecución normalmente. Si se detecta un desbalance se invoca al método de balance de carga dinámico. El proceso de re-configuración de los *meta-clusters* se efectúa antes del proceso de *migración*, que es el encargado de enviar a los individuos almacenados localmente que pertenecen a *clusters* remotos a donde realmente pertenecen.

A continuación se describirá detalladamente la metodología seguida para la implementación del simulador distribuido desarrollado en el presente trabajo.

5.1. Adaptación 3D del modelo de *Huth & Wissel*

El modelo de *Huth & Wissel* consiste en una serie de reglas de comportamiento disenadas para representar el movimiento de un banco de peces en un ambiente bidimensional. Con el objetivo de reducir la abstracción del modelo, es decir, acercar el modelo a la realidad se ha modificado el modelo para poder representar el movimiento de un banco de peces en un ambiente tridimensional.

5.1.1. Reglas de comportamiento $3D$

La adaptación de las reglas de comportamiento de un modelo bidimensional a uno tridimensional puede ser un proceso bastante complejo. Esta adaptación puede transformar el conjunto inicial de reglas de comportamiento en un conjunto de reglas de comportamiento completamente distintas. Por ejemplo, rotar un vector en el espacio euclidiano bidimensional es un proceso bastante simple que se lleva a cabo por medio de la multiplicación de un vector por una matriz de rotación. En el caso del espacio euclidiano tridimensional es un bastante proceso complejo, y casi imposible. A continuación se presentarán las fórmulas matemáticas que representan la adaptación del modelo de *Huth & Wissel* para la simulación del movimiento de un banco de peces en un ambiente tridimensional.

Repulsión

Sean $\vec{v}_i$ y $\vec{v}_j$ las orientaciones del *i-ésimo* y *j-ésimo* pez, $\alpha = \angle(\vec{v}_i, \vec{v}_j)$ el ángulo entre $\vec{v}_i$ y $\vec{v}_j$, y n y m dos números reales, tal que:

$$\begin{gathered} \vec{w} = n * \vec{v}_i - m * \vec{v}_j \\ \vec{w} \perp \vec{v}_i \end{gathered} \tag{5.1}$$

Aplicando el teorema del seno, se tiene lo siguiente:

$$\frac{|\vec{w}|}{\sin(\alpha)} = \frac{|n * \vec{v}_i|}{\sin(\frac{\pi}{2} - \alpha)} = \frac{|m * \vec{v}_j|}{\sin(\frac{\pi}{2})} \Rightarrow \frac{|\vec{w}|}{\sin(\alpha)} = \frac{n * |\vec{v}_i|}{\sin(\frac{\pi}{2} - \alpha)} = \frac{m * |\vec{v}_j|}{\sin(\frac{\pi}{2})} \tag{5.2}$$

Ya que $\vec{v}_i$, $\vec{v}_j$ y $\vec{w}$ son vectores unitarios, se puede decir que:

$$\frac{1}{\sin(\alpha)} = \frac{n}{\sin(\frac{\pi}{2} - \alpha)} = \frac{m}{\sin(\frac{\pi}{2})} \Rightarrow n = \frac{\sin(\frac{\pi}{2} - \alpha)}{\sin(\alpha)}, m = \frac{\sin(\frac{\pi}{2})}{\sin(\alpha)} \tag{5.3}$$

Una vez obtenidos los valores de n y m, el vector de repulsión es representado por:

$$\text{mín}(\angle(\vec{v}_j, \vec{w}), \angle(\vec{v}_j, -\vec{w})) \tag{5.4}$$

Orientación paralela

Sean $\vec{v}_i$ y $\vec{v}_j$ las orientaciones del *i-ésimo* y *j-ésimo* pez. El cambio de orientación del pez i respecto al pez j es dada por:

$$\vec{v}_i = \vec{v}_j \tag{5.5}$$

Atracción

Sean $\vec{v}_i$, p_i y $\vec{v}_j$, p_j las orientaciones y posiciones del *i-ésimo* y *j-ésimo* pez. El cambio de orientación del pez i respecto al pez j es dada por:

$$\begin{aligned} \vec{w}_i &= p_j \vec{-} p_i \\ \vec{v}_i &= \vec{w} \end{aligned} \tag{5.6}$$

5.1.2. Incertidumbre en el ángulo de giro

El incertidumbre en el ángulo de giro en el modelo bidimensional está determinado por una matriz de rotación de la forma:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.7}$$

La ecuación (5.7) permite rotar un vector en función a los ejes coordenados del plano euclidiano. Pero debido a que la intención es rotar un vector en el espacio euclidiano tridimensional y respecto a un vector arbitrario, el uso de la ecuación (5.7) no es viable.

En su lugar, se utilizará la fórmula de rotación de *euler-rodrigues* [47], la cual permite rotar un vector respecto a un vector normal.

Sean $\vec{v}_i(\Delta t)$ y $\vec{v}_i(\Delta t + 1)$ las orientaciones en el instante Δt y $\Delta t + 1$ del *i-ésimo* pez. $\vec{v}_i(\Delta t + 1)$ es obtenida de forma determínistica a partir de la mezcla

de influencias de los vecinos más cercanos del pez i. Sea $\vec{n}$, un vector normal a $\vec{v}(\Delta t)$ y $\vec{v}(\Delta t + 1)$, tal que:

$$\vec{n} = \vec{v}(\Delta t) \times \vec{v}(\Delta t + 1) \tag{5.8}$$

Sea α, un número con distribución normal dado por la ecuación (2.9) que representa la incertidumbre en el ángulo de giro de $\vec{v}(\Delta t + 1)$:

$$[R] = \cos\alpha \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + (1 - \cos\alpha) \begin{bmatrix} n_x n_x & n_x n_y & n_x n_z \\ n_y n_x & n_y n_y & n_y n_z \\ n_z n_x & n_z n_y & n_z n_z \end{bmatrix} + \sin\alpha \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} \tag{5.9}$$

La ecuación (5.9) representa la fórmula de rotación de *euler-rodrigues* en términos del ángulo α y eje de rotación $\vec{n}$. En consecuencia, el nuevo vector de orientación $\vec{v}\,'(\Delta t + 1)$ queda dado por:

$$\vec{v}\,'(\Delta t + 1) = [R]\ \vec{v}(\Delta t + 1) \tag{5.10}$$

5.2. Método de particionamiento

El método de descomposición del dominio del problema propuesto en el presente trabajo consiste en una estrategia de particionamiento de grano fino basada en *particiones compactas*. Una partición compacta, también llamada *cluster*, es un área generada a partir de un individuo de referencia llamado *centro* o *centroide*, y los individuos pertenecientes a dicha área. El principal objetivo de este tipo de descomposición es dividir el dominio del problema en pequenas zonas que pueden ser intersectadas o no. A partir de esta intersección se puede determinar si una zona se encuentra cerca o lejos otra, permitiendo descartar la evaluación de individuos que residen en zonas demasiado alejadas en el proceso de *búsqueda de vecinos más influyentes*.

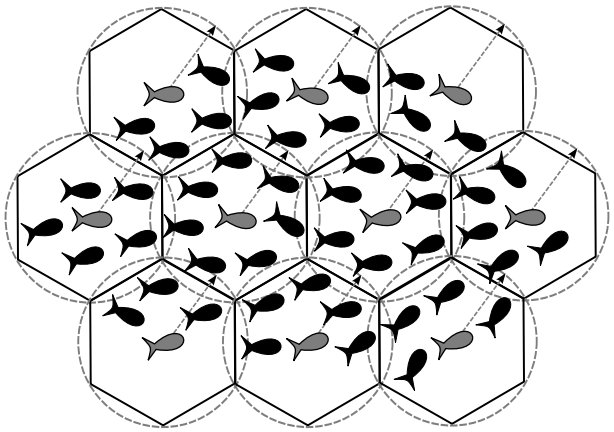


Figura 5.1: Particionamiento basado en criterio del radio cobertor/diagramas de voronoi.

Este es un método de particionamiento de grano fino el cual consiste en descomponer el dominio en un conjunto de particiones compactas generadas por medio de dos criterios: *criterio del radio cobertor* y *diagrama de Voronoi*. El método de descomposición consta de dos etapas:

Selección de centroides: Primero, es necesario establecer un conjunto de centroides lo suficientemente alejados unos de otros. Para esto se utiliza el criterio del radio cobertor. Se fija el *radio del cluster* en función del máximo rango de visión del pez R_{MAX}. De esta manera se asegura que los individuos de un *cluster* arbitrario sólo interactúen con los individuos pertenecientes a otros *clusters* adyacentes. Por medio del criterio del radio cobertor se obtiene un conjunto de centroides con una distancia mínima de $R_{max} = \eta R_{MAX}$, donde $\eta >= 1$ es el factor amplificación del rango de visión.

Descomposición del dominio: Una vez seleccionado el conjunto de centroides, se procede a asociar al resto de los individuos su partición correspondiente. Para esto se utiliza un tipo de descomposición llamado *diagrama de voronoi*. La descomposición consiste en asignar los individuos restantes al área determinada por el centroide más cercano. De esta manera se obtiene un conjunto de particiones de similar forma y tamano.

La cantidad de centroides y individuos por área está estrictamente asociada al factor de amplificación η. Asumiendo que los individuos en un banco de peces tienden a alinearse, es decir, la distancia media $\overline{d}$ entre individuos se encuentra entre $[R_1, R_2]$ (R_1 - radio de repulsión, R_2 - radio de orientación paralela), se puede decir que la cantidad de *clusters* k está dada por:

$$k \approx \frac{Volumen\ banco\ de\ peces}{Volumen\ cluster} \tag{5.11}$$

5.3. Estructura de datos

Una vez definido el método de particionamiento, es necesario definir una estructura de datos robusta en la cual se puedan almacenar y manipular a los individuos a medida que la simulación avanza. En el presente trabajo se utiliza una estructura de datos empleada para búsqueda por similitud en espacios métricos llamada *lista de clusters* [8]. Realmente, debido a al criterio híbrido de descomposición del dominio se utiliza algo más parecido a lo propuesto en [53].

Una *lista de clusters* es una lista enlazada de *grupos de datos* (*clusters*), asociados por un criterio predefinido y, que además, almacena algún tipo de información que permite descartar rápidamente una zona durante el proceso de búsqueda (usualmente una tabla de distancias y el radio cobertor de cada *cluster*). En la Figura 5.2 se puede ver un ejemplo de la adaptación de la *lista de clusters* utilizada en el presente trabajo.

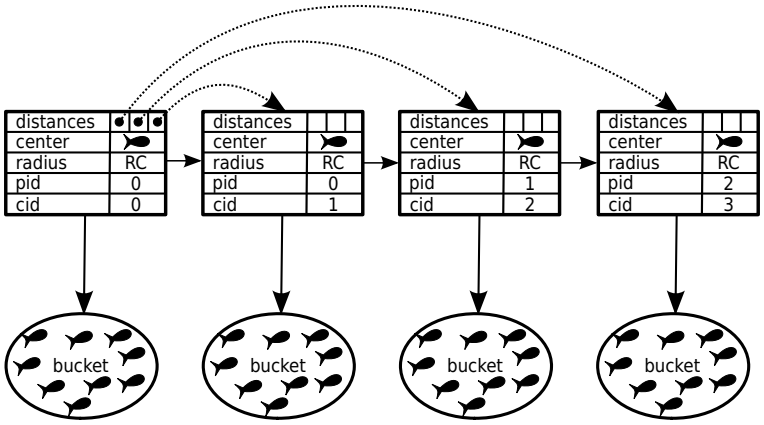


Figura 5.2: Adaptación de la lista de *clusters* para simulación distribuida.

En la adaptación de la *lista de clusters* se incluyen ciertos datos que son útiles al momento de la simulación y su distribución sobre la arquitectura paralela/distribuida. La modificación de la *lista de clusters* utilizada en el presente trabajo consta de los siguientes datos:

Centroide: Es el individuo más representativo del grupo, es decir, el centro de masa de cada *cluster*.

PID: Identificador de unidad de cómputo, indica en que unidad de cómputo está almacenado el *cluster*.

CID: Identificador de *cluster*, indica en que posición de la lista es almacenado cada *cluster*.

Radio cobertor: La distancia entre el centroide y el individuo más alejado del *cluster*. Utilizado, junto con la tabla de distancias, para descartar rápidamente un *cluster* en el proceso de búsqueda de los vecinos más influyentes..

Tabla de distancias: Almacena la distancia de un centroide al resto de los centroides. Es utilizada para descartar rápidamente un *cluster* que se encuentra demasiado lejos y, además, para asociar *clusters* en *meta-clusters*.

Se ha seleccionado este tipo de estructura de datos debido a su simplicidad y fácil manipulación. Además, se debe tener en cuenta que los datos almacenados en la lista van cambiando a medida que el programa de simulación avanza, y utilizar una estructura de datos más compleja puede acarrear serios problemas.

5.3.1. Construcción

La construcción de la *lista de clusters* consiste en la inserción iterativa de individuos sobre la estructura. Inicialmente, si la lista está vacía, se selecciona el primer individuo como centroide. Posteriormente, por cada individuo restante, se calcula la distancia al conjunto de centroides. Si la distancia es menor que R_{max} se inserta en el *bucket* del *cluster*, de lo contrario, se selecciona como un nuevo centroide.

A medida que se van insertando los individuos en la lista se va actualizando el radio cobertor de cada *cluster*. Los identificadores de *cluster* van desde cero hasta el largo de la lista menos uno. Finalmente, cuando ya se tiene la *lista de clusters* construida, se actualiza la tabla de distancia de cada *cluster*.

5.4. Algoritmo secuencial

Uno de los principales problemas que se presentan en este tipo de simulaciones es la evaluación de *todos contra todos* los individuos en el proceso de *búsqueda de individuos más influyentes*. La utilización de la *lista de clusters* evita este problema, ya que por medio de la tabla de distancias y el radio cobertor de cada *cluster* se puede obtener su vecindario, es decir, el conjunto de *clusters* adyacentes. La determinación del vecindario de un *cluster* está dada por la intersección entre *clusters*. Sean $[c_i]$, $[c_j]$ dos *clusters* de centroides c_i y c_y respectivamente, r_i y r_j los radios cobertores de cada *cluster*, $d_i(j)$ la distancia entre los centroides c_i y c_j almacenada en la tabla de distancias d_i del *cluster* $[c_i]$ y R_{MAX} el máximo rango de visión de un pez. Por lo tanto, la intersección entre los *clusters*. $[c_i]$ y $[c_j]$ queda dada por:

$$d_i(j) - (r_i + r_j) \leq R_{MAX}, i \neq j \tag{5.12}$$

Si la condición expresada en la ecuación (5.12) se cumple, significa que los individuos más alejados de los *clusters* $[c_i]$ y $[c_j]$ pueden verse. En otras palabras, se determina si los individuos de las fronteras de dos *clusters* adyacentes están dentro de máximo rango de visión R_{MAX}.

Por un lado, se define como vecindario de un *cluster* al conjunto de *clusters* adyacentes a este. Por otro lado, se definirá como vecindario efectivo de un *cluster* a todos los *clusters* adyacentes que solamente contienen a los individuos que se encuentran dentro del rango de visión del individuo más alejado del *cluster* en cuestión. Esto ayuda a eliminar cómputo innecesario al momento de elegir a los *candidatos a vecinos más influyentes*. Sea $d(c_i, [c_j]_k)$ la distancia entre el centroide c_i perteneciente al *cluster* $[c_i]$ y el *k-ésimo* individuo contenido en el *bucket* del *cluster* $[c_j]$, tal que:

$$d(c_i, [c_j]_k) - r_i \leq R_{MAX} \tag{5.13}$$

Con la ecuación (5.13) se determina si el *k-ésimo* individuo contenido en el *bucket* del *cluster* $[c_j]$ pertenece al vecindario efectivo $\mathbb{N}\{[c_i]\}$ del *cluster* $[c_i]$.

Sea $\mathbb{N}\{[c_i]\}$ el vecindario efectivo del *cluster* $[c_i]$, j representa a un individuo perteneciente al *bucket* del *cluster* $[c_i]$ y $\mathbb{LC}$ la *lista de clusters*. Por lo tanto, un paso de simulación queda dado por el siguiente algoritmo:

Algoritmo 1 *simulation*(Δt)

```
for all [c] ∈ LC do
  update c by using N{[c]})
end for
for all [c] ∈ LC do
  for all j ∈ [c] do
    update j by using N{[c]})
    reinsert j in LC
  end for
end for
```

El algoritmo (1) consiste en dos etapas: **actualizar las orientaciones y posiciones de los centroides de cada *cluster*** - se utiliza el vecindario efectivo de cada *cluster* para encontrar los *vecinos más influyentes* del centroide. Luego, se aplican las reglas del modelo para obtener la nueva orientación y la nueva posición, y **actualizar las orientaciones y posiciones de los individuos contenidos en los *buckets de cada cluster*** - se utiliza el vecindario efectivo de cada *cluster* para obtener los *vecinos más influyentes* de cada individuo del *bucket*. Se aplican las reglas del modelo para obtener la nueva orientación y la nueva posición. Luego, se re-insertan los individuos sobre la cabecera de la lista (buscando el centroide más cercano). La re-inserción se realiza sobre toda la cabecera de la lista debido a que los individuos pertenecientes a los *buckets* pueden cambiar de *cluster* a

medida que la simulación avanza. Finalmente, si un individuo queda fuera de la lista, se crea un nuevo *cluster*.

5.4.1. Eliminación del solapamiento de *clusters*

Uno de los problemas que causan los patrones de movimiento exhibidos por el modelo es el solapamiento entre *clusters*. El uso de la estrategia de particionamiento propuesta en el presente trabajo no soluciona el problema de solapamiento, mas bien, no es tan importante si un *cluster* está solapado con otro (dos *cluster* solapados son adyacentes). El problema recae cuando el centroide de un *cluster* está contenido dentro de otro *cluster*. Esta situación no debería suceder, ya que un centroide es individuo representativo de un *cluster* y debe ser único. A continuación se presentará un método de eliminación de solapamiento basado en grafos de intersección. El principal objetivo de este método es minimizar la eliminación de *clusters*, es decir, eliminar la menor cantidad de estos debido al costo que presenta la re-inserción de los individuos contenidos en su *bucket*.

Sean c_i, c_j centroides de los *clusters* $[c_i], [c_j]$ y k el número de *clusters*.

$$d(c_i, c_j) < R_{min}, i \neq j \tag{5.14}$$

Si la condición (5.14) se cumple, se traza una arista entre $[c_i]$ y $[c_j]$. Sea $G = (V, E)$ un grafo donde $V(G)$ es el conjunto de vértices and $E(G)$ es el conjunto de aristas y $N_G(v)$ es el vecindario de un vértice v en el grafo G.

Algoritmo 2 *overlapping_deletion*($\mathbb{L}$)

while $|E(G)| > 0$ **do**
  $v' \leftarrow \text{máx}_v\,(deg(v)), \forall v \in V(G)$
  delete v' and $v'N_G(v')$
  reinsert the data associated to v' within the list of clusters.
end while

El algoritmo (2) consiste en eliminar iterativamente los vértices con mayor grado. Esto garantiza eliminar la menor cantidad de *clusters*. El algoritmo se realiza iterativamente hasta que el grafo sea de grado 0. Es importante recalcar que los individuos contenidos en el *bucket* de un *cluster* eliminado deben ser re-insertados sobre la lista y que esto significa un alto costo computacional.

5.5. Simulación distribuida

La simulación de modelos orientados al individuo puede obtener un gran provecho de los entornos de computación de altas prestaciones. El objetivo principal es reducir el tiempo de ejecución de este tipo de aplicaciones. Como ya se

ha comentado anteriormente, la simulación de modelos complejos y a gran escala produce enormes desventajas al tratar de ejecutarlos en un entorno secuencial.

En esta sección se describirán los métodos utilizados para distribuir la simulación descrita en la sección anterior sobre una arquitectura paralela/distribuida. Hay que recordar que se ha hecho un cambio de enfoque al utilizar técnicas de proximidad dentro de la simulación de bancos de peces.

5.5.1. Distribución de la estructura

Distribuir este tipo de aplicaciones sobre una arquitectura distribuida no es una cuestión simple. Cabe mencionar que se ha implementado un método de particionamiento de grano fino que establece una alternativa de solución para simulaciones orientadas al individuo espacialmente explícitas que presentan patrones de movimiento. Una distribución inadecuada podría producir serios problema que afectarían al rendimiento global de la aplicación distribuida. Por ejemplo, elegir una política de distribución como *round robin* podría afectar fuertemente a las comunicaciones ya que los vecindarios estarían dispersos sobre la arquitectura paralela/distribuida.

A continuación se presentará una política de distribución basada en proximidad. El principal objetivo de este método es asociar *clusters* por proximidad en conjuntos de *clusters* que se denominarán *meta-cluster*. La implementación es bastante simple debido a que de antemano se conocen las distancias entre *clusters* (contenidas en las tablas de distancias de cada *cluster*).

El método consiste en formar tantos *meta-clusters* como unidades de cómputo se posean. La tabla de distancias es una estructura simple formada por un conjunto de tuplas de la forma $(index, distance)$, en donde $index$ es el identificador de *cluster* (*cid*) y *distance* la distancia al centroide del *cluster*. Esta tabla de distancias tiene tantas tuplas como *clusters* tiene la lista y está ordenada por distancias. Inicialmente, los identificadores de unidad de cómputo $[c_i]_{pid}$ de cada *cluster* $[c_i]$ están inicializados a -1. Se denotará $[c_i]_d$ a la tabla de distancia del *cluster* $[c_i]$ y $mean = \frac{individuals}{processors}$ como la media de individuos.

El algoritmo (3) consiste en asignar el identificador de unidad de cómputo a cada *cluster* de la lista $\mathbb{LC}$. Se ha denotado el acceso a los *clusters* de la lista por medio de $\mathbb{L}[index]$, en donde $index$ es el identificador de *cluster* (posición en la lista, *cid*). Además, se denotan como $\mathbb{LC}[index]_{size}$ a la cantidad de individuos del *cluster* que se encuentra en la posición $index$ de la lista.

Primero, se recorre la lista en búsqueda de *clusters* disponibles (cuyo *pid* es -1). Cuando se encuentra uno, se recorre su tabla de distancias para encontrar los *clusters* más cercanos a el (ya que la tabla de distancia está ordenada ascendentemente por distancias). Si se encuentra un *cluster* disponible, se le asigna el identificador de unidad de cómputo y se incrementa n. La variable n es un contador que indica la cantidad total de individuos por unidad de cómputo. Si se

Algoritmo 3 *assignment*($\mathbb{LC}$)

```
for pid ← 0 to NPROCS do
  for all [c] ∈ LC do
    if [c]_pid = (−1) then
      n ← 0
      for all (index, distance) ∈ [c]_d do
        if LC[index]_pid = −1 and (|mean−(n+LC[index]_size)| < |mean−n|)
        then
          LC[index]_pid ← pid
          n ← n + LC[index]_size
        else
          break
        end if
      end for
    end if
  end for
end for
```

quiere agregar un nuevo *cluster* disponible, cuya cantidad de individuos deja al total de individuos por unidad de cómputo n más alejados de la media, se rompe el ciclo y se continúa con la asignación del siguiente identificador de unidad de cómputo. El objetivo de esta asignación es establecer un estado inicial de balance de carga estático.

Posteriormente, se procede a la distribución de la estructura de datos sobre la arquitectura distribuida. Este es un proceso simple, ya que una vez inicializados los identificadores de unidad de cómputo sólo es necesario enviar a donde pertenece todo lo que tiene un identificador de unidad de cómputo distinto al local. El proceso es descrito en el siguiente algoritmo:

Algoritmo 4 *distribution*($\mathbb{LC}$)

```
for pid → 0 to NPROCS do
  LCout_pid → LC
  for all [c] ∈ LCout_pid do
    if [c]_pid ≠ pid then
      clear [c_i]
    end if
  end for
  send LCout_pid to PROC_pid
end for
```

El algoritmo (4) consiste en hacer una copia de la lista original $\mathbb{LC}$ a una

lista específica de salida $\mathbb{LC}out_{pid}$ por unidad de cómputo. Luego, limpiar (*clear*) $\mathbb{LC}out_{pid}$, es decir, eliminar todos los individuos pertenecientes a los *buckets* de los *clusters* de *pid* distinto al de la lista de salida. Esto asegura de que cada unidad de cómputo tendrá toda la cabecera de la lista junto con los individuos asignados con el algoritmo (3). Finalmente, se envía la lista $\mathbb{LC}out_{pid}$ a su destino.

5.5.2. Ejecución de la simulación

En este punto, cada unidad de cómputo tiene su propia lista de *clusters* $\mathbb{LC}_{pid}$ la cual contiene solamente una porción de los individuos del dominio de problema. Ahora el principal interés es ejecutar la simulación distribuida manteniendo la consistencia de la *lista de clusters*. Las etapas para la ejecución de un paso de simulación distribuida son los siguientes:

Intercambio de vecinos

Este proceso consiste en el intercambio de vecinos definidos a partir de la intersección entre *clusters* remotos y *clusters* locales. En el proceso de *selección de vecinos* cabe la posibilidad que la visión de un individuos se extienda hasta individuos alojados en unidades de cómputo remoto. Debido a que cada unidad de cómputo posee la cabecera de la lista por completo (centroides, tablas de distancias y radios cobertores), se pueden intersectar cada *cluster* local con *clusters* remotos por medio de la ecuación 5.12. La diferencia entre *clusters* remotos y *clusters* locales recae en el valor del *pid*.

Algoritmo 5 $neighbors\_exchange(\mathbb{LC}_{pid})$

for all $[c] \in \mathbb{LC}_{pid}$ **do**
  if $[c]_{pid} = \mathbb{PID}$ **then**
    for all $(index, distance) \in [c]_d$ **do**
      if $\mathbb{LC}[index]_{pid} \neq \mathbb{PID}$ **then**
        if $intersection([c], \mathbb{LC}[index]) = true$ **then**
          $insert\ [c]\ in\ \mathbb{LC}out_{(\mathbb{LC}[index]_{pid})}$
        end if
      end if
    end for
  end if
end for
for $pid \rightarrow 0\ to\ \mathbb{NPROCS}$ **do**
  $send\ \mathbb{LC}out_{pid}\ to\ \mathbb{PROC}_{pid}$
end for

Entonces, se crean $\mathbb{NPROCS}$ listas de salida $\mathbb{LC}out_{pid}$, en donde *pid* indica el

destino de la lista. Se evalúa la intersección entre *cluster* local $[c_i]$ y un *cluster* remoto $[c_j]$ y si la condición se cumple se almacena $[c_i]$ en $\mathbb{LC}out_{([c_j]_{pid})}$.

Al momento en el que cada unidad de cómputo recibe la listas de salida desde otras unidades de cómputo se convierten el listas de entrada $\mathbb{LC}in_{pid}$. Los *clusters* de cada lista de entrada $\mathbb{LC}in_{pid}$ son insertados en la lista local $\mathbb{LC}_{pid}$. De esta manera se pueden realizar los cálculos necesarios para simular a los individuos local como si se tratase de una simulación secuencial.

Actualización de los centroides

Este proceso consiste en actualizar las posiciones y orientaciones de los centroides de *clusters* locales. Para esto se utiliza el vecindario efectivo de cada *cluster* $N\{[c]\}$ considerado precalculado. El algoritmo es el siguiente:

Algoritmo 6 *update_centroids*($\mathbb{LC}_{pid}$)

```
for all [c] ∈ LC_pid do
  if [c]_pid = PID then
    update c_i by using N{[c]}
  end if
end for
```

Una vez terminado el proceso de actualización de centroides se verifica se cumplen la condiciones de solapamiento, es decir, la distancia mínima entre *clusters* debe ser mayor que R_{min}. Para esto se aplica el algoritmo 2. Cabe mencionar que si existe solapamiento, la re-inserción de los centroides conflictivos se realizará sobre la cabecera completa de la lista, posibilitando la inserción de individuos en *clusters* remotos.

Actualización de los *buckets*

Una vez actualizada la cabecera de la lista, se procede a actualizar a los individuos contenidos en el *bucket* de cada *cluster*. Para esto, se utiliza el vecindario efectivo $\mathbb{N}\{[c]\}$ de cada *cluster* $[c]$. El proceso consiste en actualizar las orientaciones y posiciones de cada individuo, y re-insertarlos sobre la cabecera de la lista. Cada individuo se asocia con el centroide más cercano a el. El proceso es similar al la actualización de *bucket* descrita en el algoritmo secuencial y su principal diferencia recae en que al re-insertar un individuo puede ser asociado a cualquier *cluster* de la lista, inclusive si es un *cluster* remoto. Esto representa un problema importante que influye directamente en la consistencia de la *lista de clusters*.

Migración

Como se ha mencionado anteriormente, los procesos de *actualización de centroides* y *actualización de buckets* pueden provocar la inserción de individuos sobre toda cabecera de la lista, incluyendo *clusters* remotos. Esto entrega un grado de flexibilidad en la estructura de datos debido a que permite insertar individuos sin condiciones. El problema recae en que al finalizar los procesos de re-inserción la consistencia de la lista enlazada está deteriorada. Es imposible continuar con la simulación debido a que cada unidad de cómputo es solamente encargada de actualizar los individuos pertenecientes a *clusters* locales y, terminada la re-inserción posee individuos pertenecientes a unidades de cómputo remotas.

El proceso de *migración* es el encargado de enviar a los individuos que residen localmente y pertenecientes a unidades de cómputo remotas a donde corresponden. Este proceso es sumamente importante, ya que es el encargado de re-establecer la consistencia de la lista de clusters. El proceso consiste en que cada unidad de cómputo crea $\mathbb{NPROCS}$ listas de salida $\mathbb{LC}out_{pid}$, en donde *pid* corresponde con al unidad de cómputo de destino. Cada unidad de cómputo recorre su lista local de *clusters* en búsqueda de individuos pertenecientes a *clusters* remotos, los almacena en su respectiva lista de salida y los elimina permanentemente. El algoritmo de migración es el siguiente:

Algoritmo 7 $migration(\mathbb{LC}_{pid})$

```
for all [c] ∈ LC_pid do
  if [c]_pid ≠ PID then
    insert [c] in LCout_([c]_pid)
    clear [c]
  end if
end for
for pid → 0 to NPROCS do
  send LCout_pid to PROC_pid
end for
```

Una vez finalizado el proceso de envío, cada unidad de cómputo recibe $\mathbb{NPROCS}$ listas entrada provenientes desde unidades de cómputo remoto y asocia los individuos pertenecientes a la los *buckets* de los *clusters* recibidos a donde realmente pertenecen, es decir, a la lista de *clusters* local $\mathbb{LC}_{pid}$.

Difusión de los radios cobertores

Los radios cobertores son un dato de suma importancia en este tipo de simulación. Estos permiten descartar rápidamente (en conjunto con la tabla de distancia), por medio del proceso de intersección entre *clusters*. A partir del

proceso de re-inserción, los valores de los radios cobertores van cambiando local y remotamente, ya que se van actualizando a medida que se van asignando individuos a los *clusters*. El proceso de difusión de radios cobertores es un proceso simple pero bastante importante. Este proceso consiste en que cada unidad de cómputo difunde los valores de los radios cobertores pertenecientes a *clusters* locales. Este proceso permite continuar con la simulación de forma consistente y continuar descartando zonas al momento de determinar que datos enviar en el proceso de *intercambio de vecinos*.

5.5.3. Creación cooperativa de *clusters*

Otro problema importante sucede cuando se intenta re-insertar un individuo sobre la lista de *clusters* y este queda fuera de alcance de cualquier *cluster*. Desde un punto de vista sencillo, la solución consistiría en crear un nuevo *cluster* y asignar este individuo como centroide. La solución no es tan simple como lo propuesto anteriormente, ya que al finalizar la simulación cada unidad de cómputo tendría listas de *clusters* distintas, produciendo un problema de coherencia. Desde un punto de vista centralizado, se podrían enviar a todos los individuos que queden fuera de alcance de los *clusters* de la lista a una única unidad de cómputo que administre a los individuos en este tipo de situaciones. Pero la centralización de procesos siempre acarrea problemas de comunicaciones y rendimiento, debido a que la unidad de cómputo encargada se transforma en un cuello de botella.

En el presente trabajo, se propone un estrategia de creación de *clusters* descentralizada y cooperativa. El procesos consiste en que cada unidad de cómputo crea su propia extensión de lista de *cluster* $\mathbb{LC}ext_{pid}$ la que manipula independientemente insertando a los individuos que quedan fuera de la lista global de *clusters* $\mathbb{LC}$ (compartida entre unidades de cómputo). Finalizada la simulación, cada unidad de cómputo comparte la cabecera de su extensión de lista de *clusters* con el resto de unidades de cómputo, las mezclan en una única extensión de *lista de clusters*, eliminan posibles intersecciones, y la agregan a la *lista de clusters* local $\mathbb{LC}_{pid}$. El algoritmo de creación cooperativa de *clusters* es el siguiente:

Algoritmo 8 *cooperative_creation*($\mathbb{LC}ext_{pid}$)

bcast $header :: \mathbb{LC}ext_{pid}$
recv $header :: \mathbb{LC}ext_{(0,\ldots,\mathbb{NPROCS}-1)}$ *from others*
$\mathbb{LC}ext \leftarrow$**merge**($header :: \mathbb{LC}ext(0, \ldots, \mathbb{NPROCS} - 1)$)
overlapping_deletion($\mathbb{LC}ext$)
join $\mathbb{LC}$ *and* $\mathbb{LC}ext$

Este proceso se debe realizar estrictamente antes del proceso de *migración*, ya que a partir de la eliminación del solapamiento pueden producirse inserciones en *clusters* remotos.

5.5.4. Balance de carga dinámico por proximidad

Uno de los principales inconvenientes en simulaciones distribuidas de modelos orientados al individuo espacialmente explícitos con patrones de movimiento es que los individuos pueden cambiar su posición en el ambiente a medida que la simulación avanza. El movimiento de un individuo no tiene limitaciones espaciales, es decir, puede moverse por todo el espacio de simulación sin importar si un cambio de posición implica la migración a una unidad de cómputo remota. Estas migraciones pueden causar problemas de sobrecarga o infrautilización de recursos de cómputo y como consecuencia un decremento en el rendimiento global de la aplicación de simulación distribuida.

En la simulación distribuida basada en un mecanismo de particionamiento adaptativo (presentada en este trabajo), también pueden suceder este tipo de problemas. Tanto la creación de nuevos *clusters* como la migración de individuos entre *clusters* pueden causar un desequilibrio en la carga de trabajo asignada a las unidades de cómputo. Al presentarse este tipo de problemas se ha tomado la decisión de continuar explotando la proximidad inherente en el método de particionamiento con el objetivo de mantener dinámicamente equilibrada la carga de trabajo.

El primer paso para antes de ejecutar un mecanismo de balance de carga dinámico es detectar cuando existe un desequilibrio. En el presente trabajo, se ha implementado un método de detección basado en umbrales. En esta clase de métodos se fija un umbral de desequilibrio que define las condiciones mínimas y máximas para que una unidad de cómputo pueda continuar trabajando normalmente. Si es que la carga de trabajo se encuentra fuera de los rangos, se invoca al método de balance de carga.

Una vez detectado el desequilibrio, se ejecuta el algoritmo de balance de carga. En este caso el algoritmo consiste en asignar grupos contiguos de *clusters*, es decir, aglomerar *clusters* en *meta-clusters*. Este proceso es similar al proceso de asignación expuesto en el algoritmo 3 y su principal diferencia recae en que es necesario difundir los tamanos de todos los *clusters* (inclusive de los remotos), con el objetivo de obtener el tamano real de cada *cluster* (debido a que este proceso se realiza antes de la migración). Una vez obtenidos los tamanos reales de cada *cluster* se aplica el algoritmo 3, en el cual se forman *meta-clusters* por proximidad y de tamano cercano a la media. No hay paso de mensajes explícito en este proceso debido a que se realiza antes del proceso de migración. Posteriormente, el proceso de migración se encargará de hacer la distribución de los individuos del tal manera que se volverá a lograr el objetivo de balance de carga.

Capítulo 6

Resultados experimentales

En el presente capítulo se presentarán los resultados experimentales para la simulación distribuida basada en *clustering* de bancos de peces a gran escala. El principal objetivo de este capítulo es validar y verificar los resultados obtenidos a partir de la ejecución del simulador distribuido de bancos de peces en múltiples escenarios de configuración. Se hará un estudio exhaustivo de la influencia de la magnitud del *máximo radio cobertor* $R_{max} = \eta R_{MAX}$, que delimita el tamano de las particiones dentro de la simulación. Las simulaciones han sido ejecutadas usando tres cargas de trabajo distintas: 131072 (2^{17}), 262144 (2^{18}) y 524288 (2^{19}) individuos, las cuales conforman un único banco de peces. Las cargas de trabajos han sido pre-procesadas con el objetivo de obtener la forma de un banco de peces *real* a partir de una carga de trabajo generada sintéticamente. Los resultados fueron obtenidos a partir de la ejecución de 250 pasos de simulación. Además, se han fijado tres umbrales de desbalance: 10 %, 20 % y 30 % de carga de trabajo local asignada a cada core, con el objetivo de identificar cual es el umbral de desbalance adecuado para cada escenario. La ejecución fue realizada en un *Cluster* IBM que posee las siguientes características:

Nodos	32 Nodos IBM xSerie con doble core
Procesador	Dual-Core Intel(R) Xeon(R) CPU 5160 @ 3.00GHz 4MB L2 (2x2)
Memoria	12 GB Fully Buffered DIMM 667 MHz
Almacenamiento	Hot-swap SAS Controller 160GB SATA Disk
Red de Interconexión	Integrated dual Gigabit Ethernet

Cuadro 6.1: Características Cluster IMB.

Los parámetros típicos utilizados en este tipo de simulaciones y que además aseguran la cohesión del banco de peces a través del tiempo son los siguientes:

R_3	10,0BL	Radio de atracción	R_{MAX}
R_2	5,0BL	Radio de orientación paralela	
R_1	1,0BL	Radio de repulsión	
ω	$\frac{\pi}{3}$		*dead angle*
vecinos	10		*front prioritized*
speed	*distribución* Γ		$K = 4, A = 3{,}3$
ω	$\frac{\pi}{12}$	Desviación estándar	

Cuadro 6.2: Parámetros típicos en simulación de bancos de peces.

6.1. Simulación de 131072 Individuos

A continuación se analizarán los tiempos de ejecución obtenidos para la simulación distribuida de 131072 individuos en un conjunto de diferentes de escenarios. Estos escenarios están compuestos por distintos factores que influyen en la simulación, tales como: *cantidad de cores*, *tamaño de los radios cobertores* y *umbrales de desbalance*. Las simulaciones fueron ejecutadas utilizando $1, 2, 4, 8, 16, 32, 64$ y 128 *cores*. Los límites de los radios cobertores son presentados en el cuadro 6.3. Se han utilizado tres umbrales de desbalance $10\,\%, 20\,\%$ y $30\,\%$ de la carga de trabajo local. Los resultados presentados a continuación fueron obtenidos por medio del promedio de 250 pasos de simulación.

	R_{min}	R_{max}
$\mathbf{conf}_1$	1,00R_{MAX}	1,05R_{MAX}
$\mathbf{conf}_2$	1,05R_{MAX}	1,10R_{MAX}
$\mathbf{conf}_3$	1,10R_{MAX}	1,15R_{MAX}
$\mathbf{conf}_4$	1,15R_{MAX}	1,20R_{MAX}
$\mathbf{conf}_5$	1,20R_{MAX}	1,25R_{MAX}

Cuadro 6.3: Radios cobertores para la simulación de 131072 individuos.

6.1.1. Tiempos de ejecución

En la figura 6.1 se muestran los resultados obtenidos para la simulación de 131072 individuos utilizando 1 core (simulación secuencial). Claramente se observa que a medida que se incrementan el tamano de los límites de los radios cobertores, se incrementa el tiempo de ejecución. Esto indica que en este caso en particular la configuración $\mathbf{conf}_1$ es la elección adecuada para la ejecución de la simulación, es decir, la utilización de radios cobertores más pequenos permite la obtención de mejores resultados en términos de tiempo de ejecución.

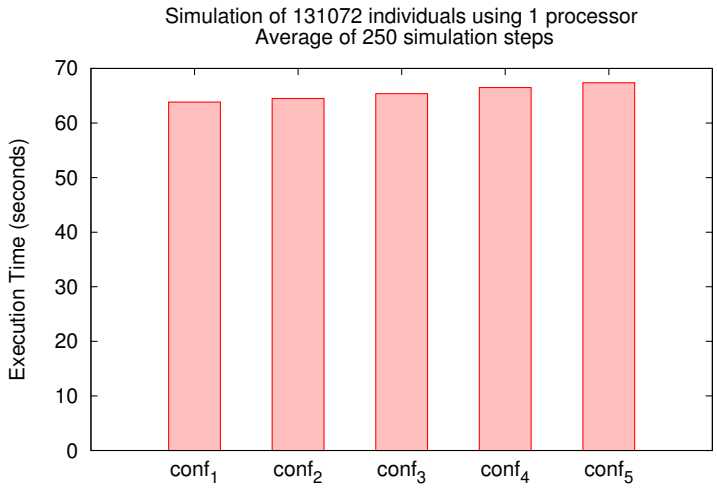


Figura 6.1: Simulación de 131072 individuos usando 1 core.

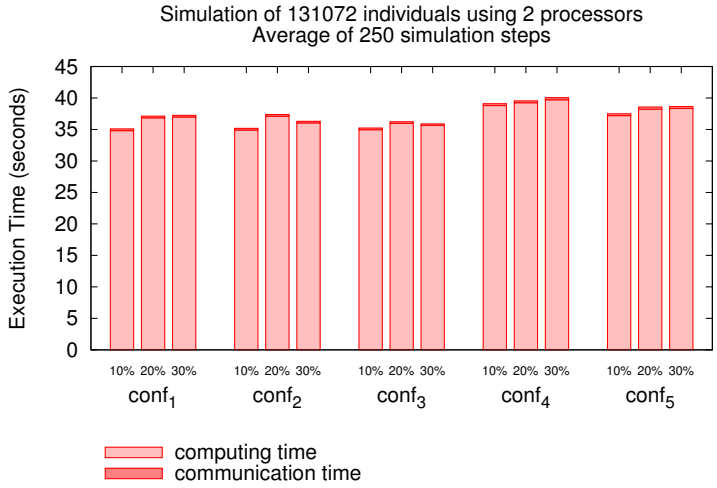


Figura 6.2: Simulación de 131072 individuos usando 2 cores.

En la figura 6.2 se muestran los resultados obtenidos para la simulación distribuida de 131072 individuos utilizando 2 cores. Se observa que los mejores resultados se han obtenido por medio de la utilización de las configuraciones: $\mathbf{conf}_1$, $\mathbf{conf}_2$ y $\mathbf{conf}_3$. Estas configuraciones representan radios de cobertura relativamente pequenos. Además, se aprecia una tendencia a obtener mejores resultados de simulación utilizando un umbral de desbalance de 10 %. Los tiempos

de comunicación son insignificantes en comparación a los tiempos de ejecución obtenidos.

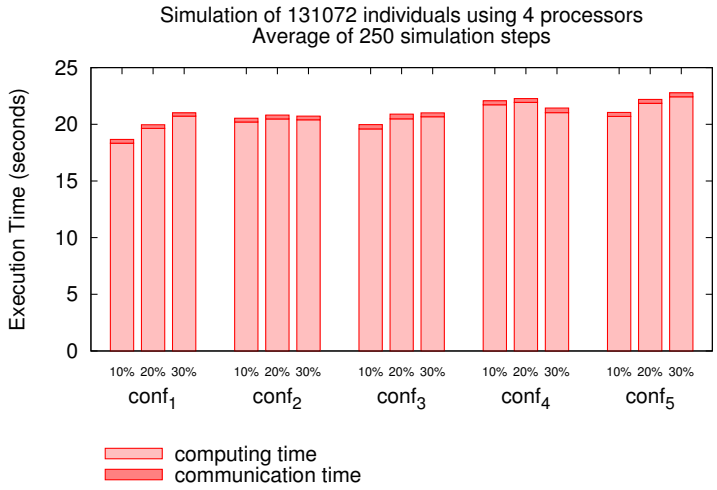


Figura 6.3: Simulación de 131072 individuos usando 4 cores.

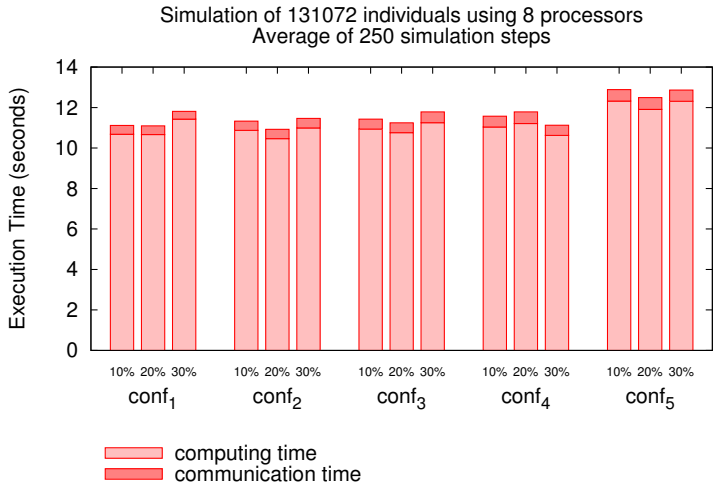


Figura 6.4: Simulación de 131072 individuos usando 8 cores.

En la figura 6.3 se muestran los resultados obtenidos para la simulación distribuida de 131072 individuos utilizando 4 cores. Se observa una tendencia similar al caso anterior (ver Fig. 6.2), es decir, los mejores resultados fueron obtenidos

a partir de las configuraciones: $\mathbf{conf}_1$, $\mathbf{conf}_2$ y $\mathbf{conf}_3$, utilizando un umbral de desbalance de 10 %. Los tiempos de comunicación se mantienen relativamente similares a los casos anteriores, es decir, insignificantes en comparación a los tiempos de ejecución.

En la figura 6.4 se muestran los resultados obtenidos para la simulación distribuida de 131072 individuos utilizando 8 cores. En este caso se puede observar que por medio de la utilización de las configuraciones: $\mathbf{conf}_1$, $\mathbf{conf}_2$, $\mathbf{conf}_3$ y $\mathbf{conf}_4$, se obtienen resultados muy similares en términos de tiempos de ejecución. También se puede apreciar que dependiendo la configuración utilizada el umbral de desbalance varía en su magnitud. Los mejores resultados obtenidos para este caso son los siguientes: $\mathbf{conf}_2$ con un umbral de desbalance del 20 % y la $\mathbf{conf}_3$ con un umbral de desbalance del 30 %. Los tiempos de comunicación muestran un incremento en comparación a los casos expuestos previamente, no obstante, aún son insignificantes en comparación a los tiempos de ejecución obtenidos.

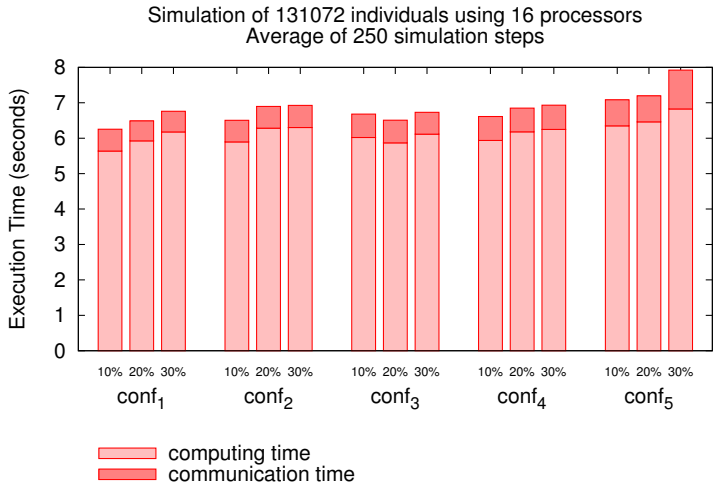


Figura 6.5: Simulación de 131072 individuos usando 16 cores.

En la figura 6.5 se muestran los resultados obtenidos para la simulación distribuida de 131072 individuos utilizando 16 cores. En este caso se puede apreciar que los mejores resultados fueron obtenidos a partir de la utilización de las configuraciones: $\mathbf{conf}_1$, $\mathbf{conf}_2$, $\mathbf{conf}_3$ y $\mathbf{conf}_4$. En términos de optimización, se puede apreciar que existe una tendencia a obtener un mejor rendimiento a partir de la utilización las configuraciones: $\mathbf{conf}_1$ utilizando un umbral de desbalance del 10 %, $\mathbf{conf}_2$ utilizando un umbral de desbalance del 10 % y $\mathbf{conf}_3$ utilizando un umbral de desbalance del 20 %. Los tiempos de comunicación muestran un incremento en comparación a los casos expuestos anteriormente, no obstante,

todavía representan una pequena porción en comparación a los tiempos de ejecución obtenidos.

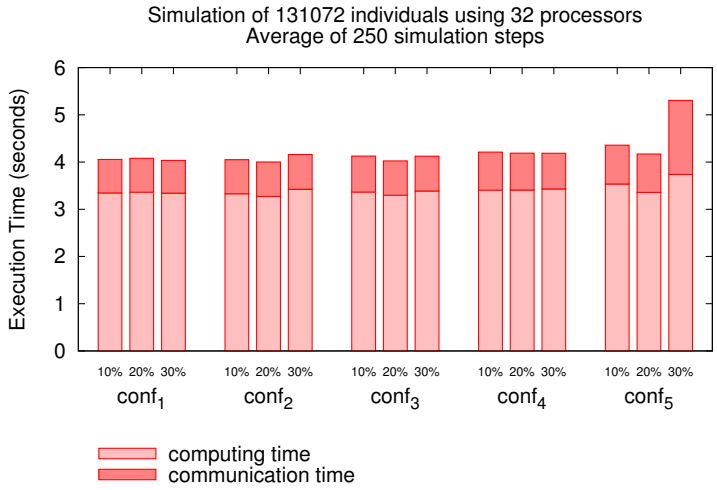


Figura 6.6: Simulación de 131072 individuos usando 32 cores.

En la figura 6.6 se muestran los resultados obtenidos para la simulación distribuida de 131072 individuos utilizando 32 cores. En este caso se observa una tendencia muy similar en términos de tiempos de ejecución utilizando todas las configuraciones, a excepción de $\mathbf{conf}_5$ con un umbral de desbalance del 30 %. En términos de optimización, se puede apreciar una leve ganancia por medio de al utilización de las configuraciones: $\mathbf{conf}_2$ y $\mathbf{conf}_3$, ambas con umbrales de desbalance del 20 %. Los tiempos de comunicación se mantiene relativamente similares en todos los casos, a excepción de la configuración $\mathbf{conf}_5$ con un umbral de desbalance del 30 %, en el que los tiempos de comunicación se incrementan explosívamente. Los tiempos de comunicación se han incrementado en comparación a los casos expuestos anteriormente debido que a medida que la cantidad de cores se incrementa los tiempos de comunicación se incrementan de forma acoplada.

En la figura 6.7 se muestran los resultados obtenidos para la simulación distribuida de 131072 individuos utilizando 64 cores. Se puede apreciar que en este caso todas las configuraciones presentar resultados muy similares. También se puede apreciar que los tiempo de comunicación se han incrementado sustancialmente en comparación al caso anterior (ver Fig. 6.6). En términos de optimización, se puede apreciar que los mejores resultados fueron obtenidos por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ y $\mathbf{conf}_2$, ambas con umbrales de desbalance del 30 %.

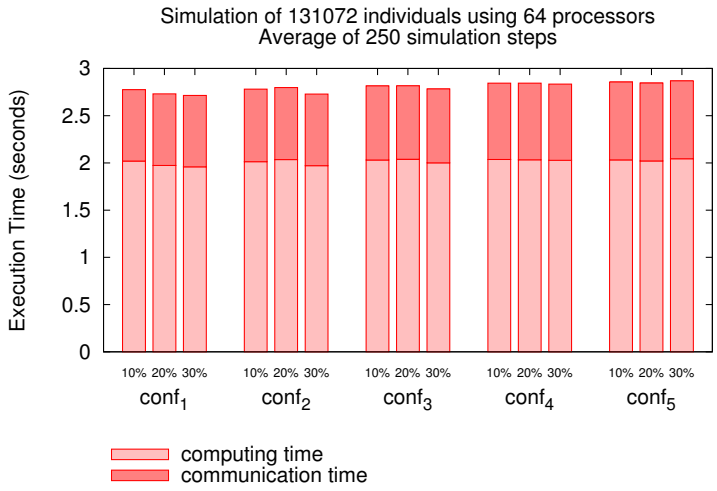


Figura 6.7: Simulación de 131072 individuos usando 64 cores.

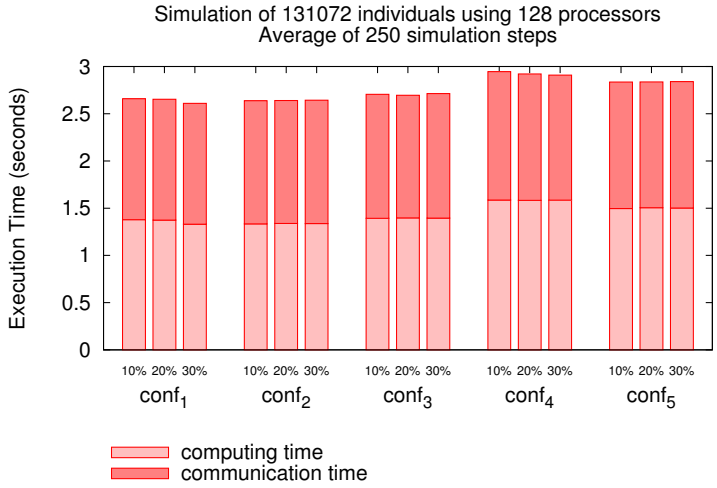


Figura 6.8: Simulación de 131072 individuos usando 128 cores.

En la figura 6.8 se muestran los resultados obtenidos para la simulación distribuida de 131072 individuos utilizando 128 cores. Es este caso se puede apreciar que la utilización las configuraciones $\mathbf{conf}_4$ y $\mathbf{conf}_5$ degradan el rendimiento de la simulación distribuida en términos de tiempos de ejecución. En términos de optimización, se puede apreciar que por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ con un umbral de desbalance de 20 %, $\mathbf{conf}_2$ con

un umbral de desbalance del 10 % y $\mathbf{conf}_3$ con un umbral de desbalance del 20 %, se ha obtenido una leve ganancia. Los tiempo de comunicación muestran un gran incremento en comparación a los casos expuestos anteriormente, siendo proporcionalmente similares a los tiempos de ejecución. Este comportamiento puede ser causado debido a que la carga de trabajo es demasiado pequena para la ejecución de la simulación distribuida por medio de la utilización de 128 cores.

cores	**conf**	$\mathbf{T}_{computo}$	$\mathbf{T}_{comunicacion}$	**umbral de desbalance**	*clusters*
1	$\mathbf{conf}_1$	63,8381	-	-	997
2	$\mathbf{conf}_1$	34,817596	0,276004	10 %	995
4	$\mathbf{conf}_1$	18,333093	0,333607	10 %	998
8	$\mathbf{conf}_2$	10,461	0,465	20 %	885
16	$\mathbf{conf}_1$	5,638824	0,614996	10 %	1001
32	$\mathbf{conf}_2$	3,272503	0,729337	20 %	886
64	$\mathbf{conf}_1$	1,958493	0,756157	30 %	999
128	$\mathbf{conf}_1$	1,33077	1,27963	30 %	1001

Cuadro 6.4: Sumario de los mejores resultados obtenidos para la simulación de 131072 individuos.

En el cuadro 6.4 se presentan un resumen de los mejores resultados obtenidos para la simulación distribuida de 131072 individuos. Estos resultados han sido obtenidos a partir del promedio de 250 pasos de simulación. Se puede apreciar que existe una tendencia a obtener resultados deseables por medio de la utilización de radios cobertores muy pequenos, cercanos al máximo radio de visibilidad R_{MAX}. También se puede observar que en los casos en los cuales se utilizan $2, 4, 8, 16$ y 32 cores, se obtienen mejores resultados por medio de la utilización de umbrales de desbalance pequenos (10 % y 20 %). Por otro lado, en los casos en los cuales se utilizan 64 y 128 cores, la utilización de umbrales de desbalance más grandes ofrecen un mejor desempeno.

También, se puede observar que a medida que se incrementa la cantidad de cores hasta 32, los tiempos de comunicación se incrementan levemente. En el caso particular de 128 se puede observar que los tiempos de comunicación casi duplican los obtenidos en el caso anterior (64 cores). Esto puede ser generado por tres causas principales: el tamano de la carga de trabajo es demasiado pequeno para ejecutar la simulación con 128 cores, la distribución de dominio del problema se vuelve ineficiente cuando se incrementa la cantidades de cores o los patrones de comunicación se tornan impredecibles cuando se incrementa la cantidad de cores.

6.1.2. Speedup

En la figura 6.9 se expone la ganancia obtenida en términos de *speedup* para la simulación distribuida de 131072 individuos. Esta gráfica fue obtenida a partir

de los valores mostrados en el cuadro 6.4. Se puede observar que los casos en los cuales se utilizan $2, 4, 8, 16$ cores, la simulación distribuida muestra un ganancia bastante aceptable y muy cercana a la lineal. Por otro lado, se puede observar que el *speedup* comienza a decrementarse cuando se utilizan $32, 64, 128$ cores. Este comportamiento puede ser principalmente causado por que la comunicación se incrementa irregularmente a medida que se incrementa la cantidad de cores (ver cuadro 6.4). Esto puede ser provocado por que la carga de trabajo es demasiado pequena para ejecutar la simulación distribuida con esas magnitudes de cores o por que a medida que la simulación avanza, el mecanismo de balance de carga dinámica no ajusta adecuadamente la carga de trabajo.

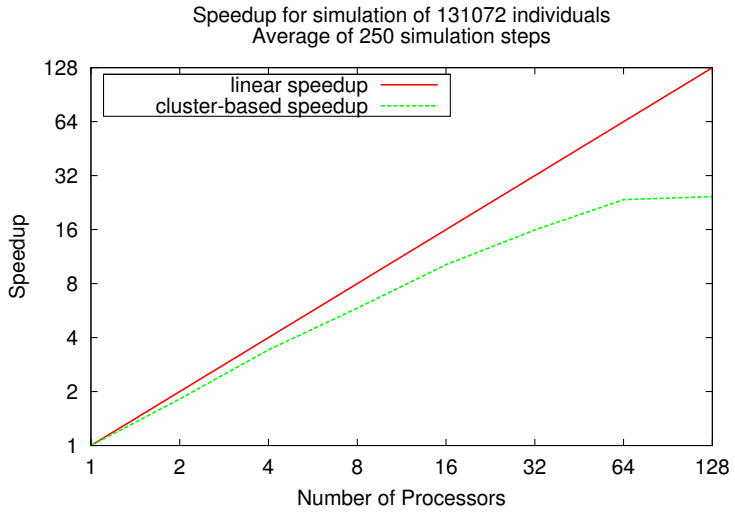


Figura 6.9: Speedup para la simulación de 131072 individuos.

Particularmente, en la simulación distribuida de 131072 individuos utilizando 128 cores, la cantidad de *clusters* (por medio de la utilización de la configuración $\mathbf{conf}_1$), ronda valores cercanos a los 1000 (ver Cuadro 6.4), por lo que cada *cluster* tiene una cantidad de individuos cercana a los 131 y cada core tiene una cantidad de *clusters* cercana a los 7 ($\approx$ 1024 individuos por core). Debido a la limitada cantidad de *clusters* por core se incrementa la probabilidad de ocurrencia de migraciones progresivas entre PLs, lo que acarrea un incremento en los tiempos de cómputo y comunicación y como consecuencia surge un decremento en el rendimiento de la simulación distribuida (como se puede observar en la Fig 6.9).

6.1.3. Ajuste dinámico de la carga de trabajo

En la figura 6.10 se muestra el progreso de la carga de trabajo en función del tiempo para la simulación distribuida de 131072 individuos utilizando 128

cores sobre 250 pasos de simulación con un 10 % de umbral de desbalance local, lo que significa que la carga de trabajo de cada core debe mantenerse dentro de [922 : 1126] individuos.

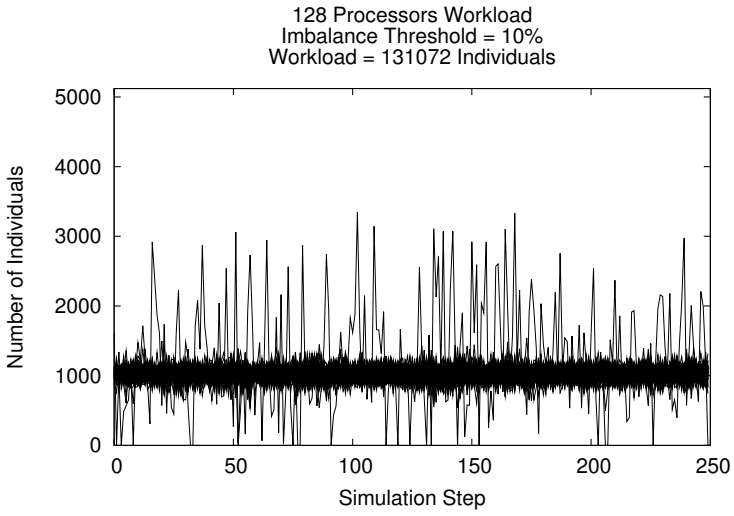


Figura 6.10: Carga de trabajo para la simulación 131072 individuos usando 128 cores con un 10 % de umbral de desbalance.

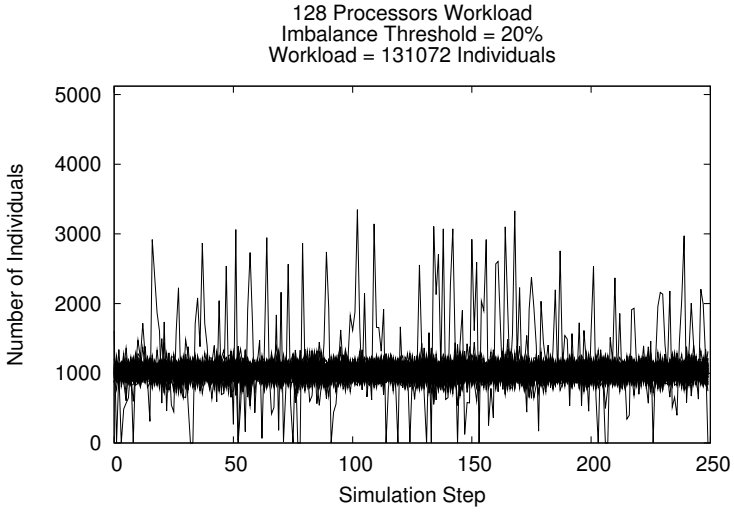


Figura 6.11: Carga de trabajo para la simulación 131072 individuos usando 128 cores con un 20 % de umbral de desbalance.

En la figura 6.10 se puede apreciar la existencia de múltiples picos a medida que progresa la simulación distribuida en función del tiempo. Este comportamiento es principalmente causado debido a que la carga de trabajo es demasiado pequena para ser ejecutada sobre 128 cores. Esto provoca que sucedan migraciones progresivas entre PLs ocasionando la sobrecarga e infrautilización de recursos de cómputo, forzando al mecanismo de balance de carga dinámico a intentar equilibrar la carga. Este último punto desencadena una degradación en la simulación distribuida que se refleja en el *speedup* obtenido.

En la figura 6.11 se muestra la evolución de la carga de trabajo para la simulación distribuida de 131072 individuos utilizando 128 cores sobre 250 pasos de simulación con un 20 % de umbral de desbalance local, lo que significa que la carga de trabajo de cada core debe mantenerse dentro de [820 : 1288]. El comportamiento de la carga de trabajo en función del tiempo es similar al caso anterior (ver Fig 6.10) y las causas son análogas.

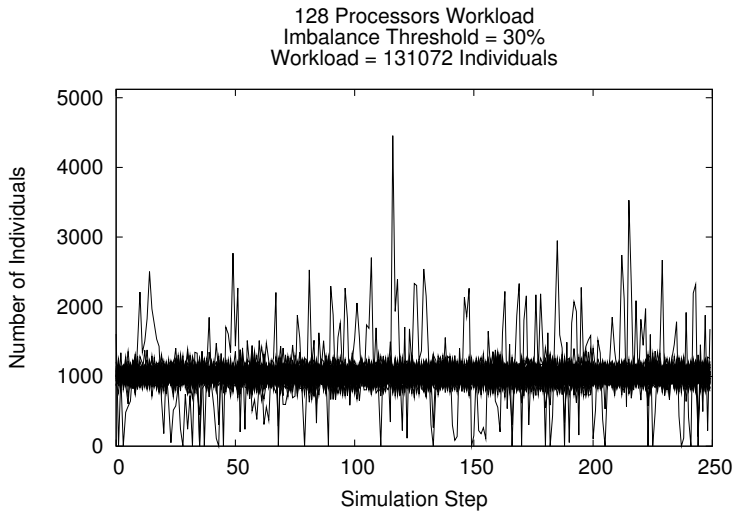


Figura 6.12: Carga de trabajo para la simulación 131072 individuos usando 128 cores con un 30 % de umbral de desbalance.

En la figura 6.12 se muestra el progreso de la carga de trabajo para la simulación distribuida de 131072 individuos utilizando 128 cores sobre 250 pasos de simulación con un 30 % de umbral de desbalance local, lo que significa que la carga de trabajo de cada core debe mantenerse dentro de [717 : 1331]. El comportamiento de la carga de trabajo en función del tiempo es similar a ambos casos expuestos anteriormente.

6.2. Simulación de 262144 Individuos

A continuación se presentarán los resultados obtenidos para la simulación distribuida de 262144 individuos en distintos escenarios. Los escenarios están compuestos por los distintos factores que influyen en el rendimiento de la simulación distribuida, tales como: *cantidad de cores*, *tamaño de los radios cobertores* y *umbrales de desbalance*. Las simulaciones fueron ejecutadas utilizando $1, 2, 4, 8, 16, 32, 64$ y 128 cores. Los rangos de los radios cobertores son expuestos en el cuadro 6.5. Se han utilizado tres distintos umbrales de desbalance, $10\,\%, 20\,\%$ y $30\,\%$ de la carga de trabajo local asignada a cada core. Los resultados han sido obtenidos a partir del promedio de tiempos ejecución de 250 pasos de simulación.

	R_{min}	R_{max}
$\mathbf{conf}_1$	$1{,}10R_{MAX}$	$1{,}15R_{MAX}$
$\mathbf{conf}_2$	$1{,}15R_{MAX}$	$1{,}20R_{MAX}$
$\mathbf{conf}_3$	$1{,}20R_{MAX}$	$1{,}25R_{MAX}$
$\mathbf{conf}_4$	$1{,}25R_{MAX}$	$1{,}30R_{MAX}$
$\mathbf{conf}_5$	$1{,}30R_{MAX}$	$1{,}35R_{MAX}$

Cuadro 6.5: Radios cobertores para la simulación de 262144 individuos.

6.2.1. Tiempos de ejecución

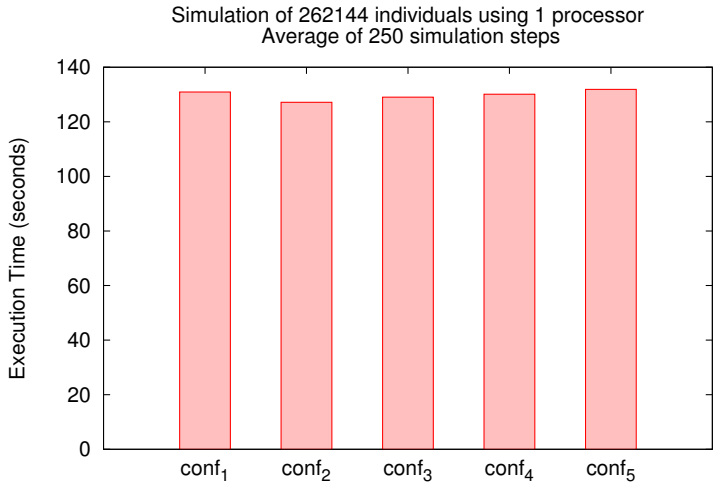


Figura 6.13: Simulación de 262144 individuos usando 1 core.

En la figura 6.13 se muestran los resultados obtenidos para la simulación de 262144 individuos utilizando un único core (ejecución secuencial). Se puede observar que el rendimiento es muy similar para todas las configuraciones. En términos de optimización, se puede apreciar una leve ganancia por medio de la utilización de la configuración $\mathbf{conf}_2$.

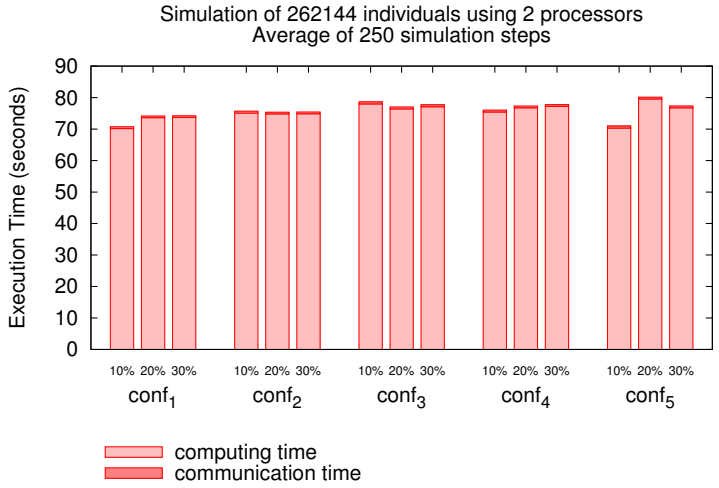


Figura 6.14: Simulación de 262144 individuos usando 2 cores.

En la figura 6.14 se muestran los resultados obtenidos para la simulación distribuida de 262144 individuos utilizando 2 cores. Se puede observar que los resultados son bastante similares en términos de rendimiento. En términos de optimización, se puede apreciar una ganancia significativa por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ y $\mathbf{conf}_5$, ambas con un umbral de desbalance del 10 %. También se puede apreciar que los tiempos de comunicación son insignificantes en comparación a los tiempos de ejecución obtenidos.

En la figura 6.15 se muestran los resultados obtenidos para la simulación distribuida de 262144 individuos utilizando 4 cores. A diferencia de casos anteriores, se puede apreciar la influencia de los umbrales de desbalance sobre los tiempos de ejecución. Utilizando todas las configuraciones, se observa que ajustando la carga de trabajo utilizando un umbral de desbalance del 10 % se pueden obtener mejores resultados. Los tiempos de comunicación son insignificantes en comparación a los tiempos de ejecución obtenidos.

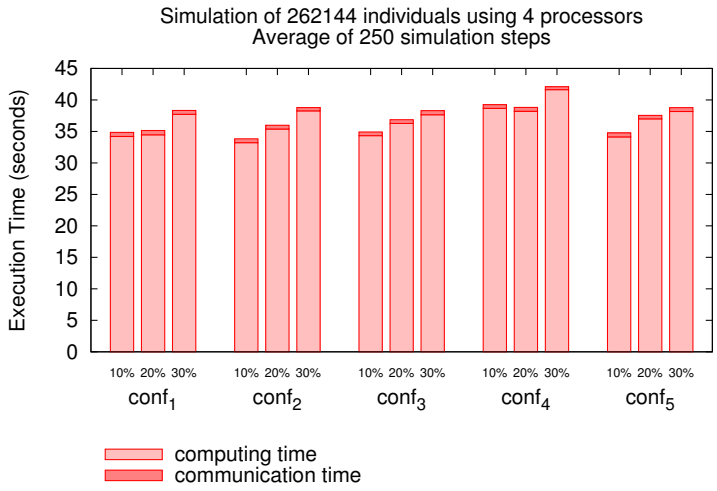


Figura 6.15: Simulación de 262144 individuos usando 4 cores.

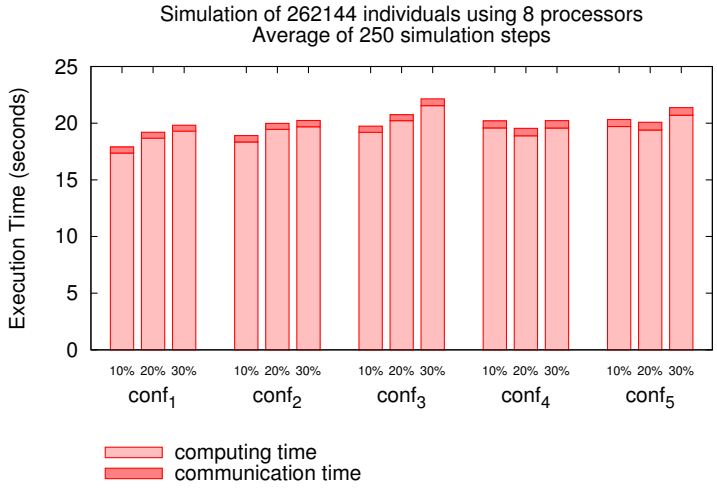


Figura 6.16: Simulación de 262144 individuos usando 8 cores.

En la figura 6.16 se muestran los resultados obtenidos para la simulación distribuida de 262144 individuos utilizando 8 cores. En este caso también se puede apreciar una tendencia similar al caso anterior, es decir, la magnitud del umbral de desbalance influye directamente en los tiempos de ejecución obtenidos. En términos de optimización, se puede apreciar que por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ y $\mathbf{conf}_2$, ambas con umbrales de desbalance de

un 10 %, existe una tendencia a obtener un mejor rendimiento. Los tiempos de comunicación son insignificantes en comparación a los tiempos de ejecución obtenidos.

En la figura 6.17 se muestran los resultados obtenidos para la simulación distribuida de 262144 individuos utilizando 16 cores. También se puede observar la influencia de la magnitud del umbral de desbalance en el rendimiento de la simulación distribuida. En términos de optimización, se puede apreciar que por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ y $\mathbf{conf}_2$, ambas con un umbral de desbalance del 10 %, existe una tendencia a lograr mejores resultados. Los tiempos de comunicación muestran un leve incremento que no representa una influencia significativa en el rendimiento de la simulación distribuida.

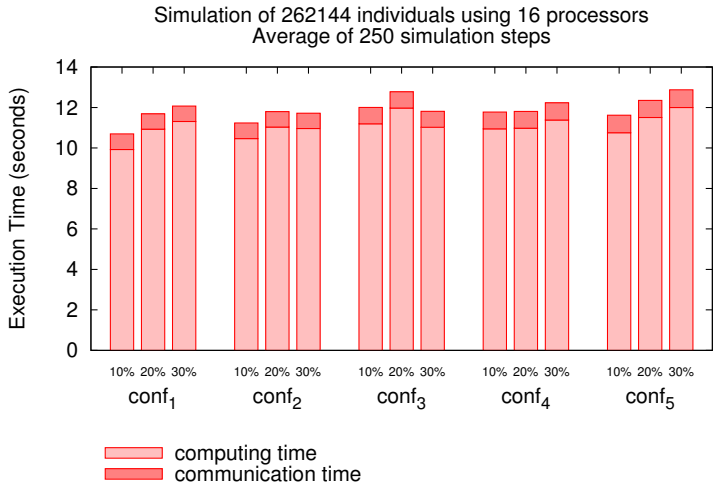


Figura 6.17: Simulación de 262144 individuos usando 16 cores.

En la figura 6.18 se muestran los resultados obtenidos para la simulación distribuida de 262144 individuos utilizando 32 cores. Los resultados obtenidos son muy similares en todos los casos, lo que indica que la utilización de cualquier configuración es viable. En términos de optimización, se observa una leve ganancia por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ con un umbral de desbalance del 20 % y $\mathbf{conf}_2$ con un umbral de desbalance del 30 %. También se puede apreciar que los tiempos de comunicación se han incrementado en comparación a los casos expuestos previamente.

En la figura 6.19 se muestran los resultados obtenidos para la simulación distribuida de 262144 individuos utilizando 64 cores. Se puede observar que todas las configuraciones utilizadas entregan resultados bastante aceptables. También se puede apreciar que aunque la magnitud de los radios cobertores ha

cambiado los tiempos de comunicación se mantienen dentro de rangos similares. En términos de optimización, se puede apreciar que por medio de la utilización de las configuraciones: $\mathbf{conf}_3$ con un umbral de desbalance del 20 % y $\mathbf{conf}_4$ con un umbral de desbalance del 30 %, existe una tendencia a obtener los mejores resultados. Los tiempos de comunicación se han incrementado sustancialmente en comparación al caso anterior (ver Fig. 6.18).

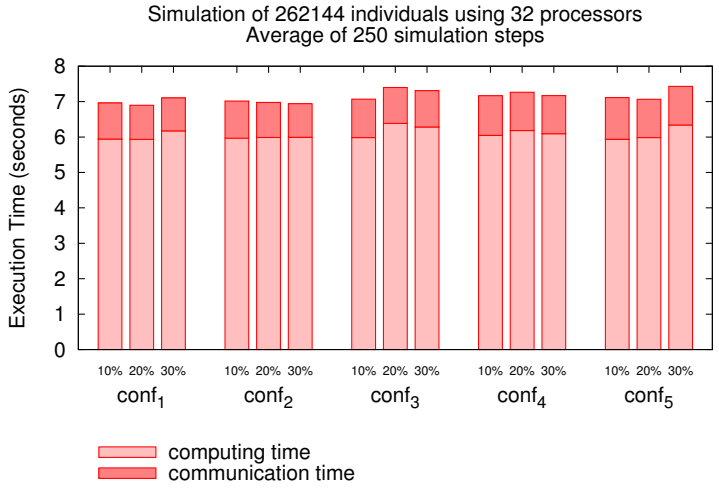


Figura 6.18: Simulación de 262144 individuos usando 32 cores.

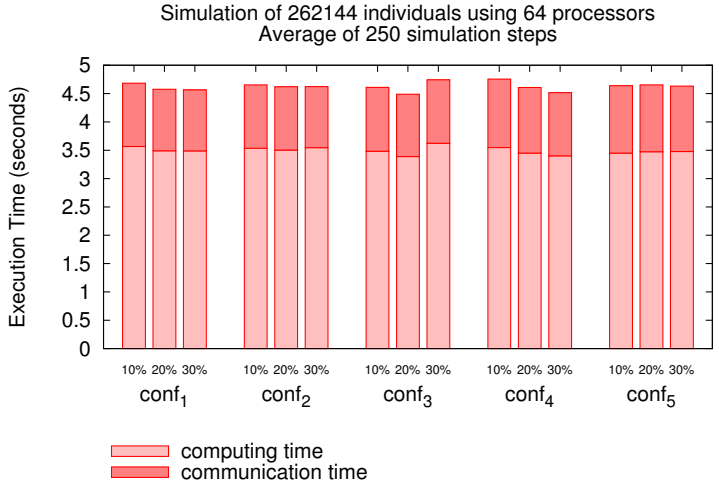


Figura 6.19: Simulación de 262144 individuos usando 64 cores.

En la figura 6.20 se muestran los resultados obtenidos para simulación distribuida de 262144 individuos utilizando 128 cores. Se puede apreciar que por medio de la utilización de todas las configuraciones los resultados en términos de tiempo de cómputo y comunicación son bastantes similares. En términos de optimización, se puede apreciar una leve ganancia por medio de la utilización de las configuraciones: $\mathbf{conf}_3$ y $\mathbf{conf}_4$, ambas con umbrales de desbalance del 20 %.

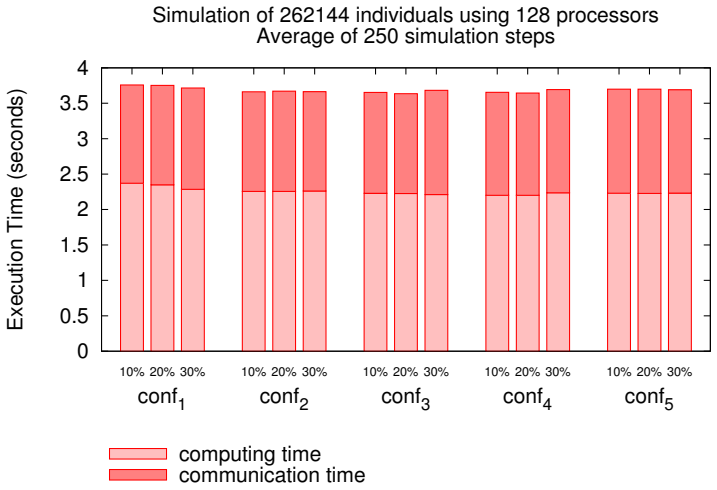


Figura 6.20: Simulación de 262144 individuos usando 128 cores.

En el cuadro 6.6 se muestra un resumen de los mejores resultados obtenido para la simulación distribuida de 262144 individuos. Los resultados han sido obtenidos a partir del promedio de 250 pasos de simulación.

cores	**conf**	$\mathbf{T}_{computo}$	$\mathbf{T}_{comunicacion}$	**umbral de desbalance**	*clusters*
1	$\mathbf{conf}_2$	127,155	-	-	1474
2	$\mathbf{conf}_1$	70,150254	0,593446	10 %	1648
4	$\mathbf{conf}_2$	33,218593	0,602807	10 %	1475
8	$\mathbf{conf}_1$	17,371473	0,523427	10 %	1648
16	$\mathbf{conf}_1$	9,924648	0,773552	10 %	1649
32	$\mathbf{conf}_1$	5,933397	0,964053	20 %	1648
64	$\mathbf{conf}_3$	3,38770	1,10171	20 %	1328
128	$\mathbf{conf}_3$	2,22557	1,40999	20 %	1329

Cuadro 6.6: Sumario de los mejores resultados obtenidos para la simulación de 262144 individuos.

En el cuadro 6.6 puede observar que la magnitud de los radios cobertores se ha

incrementado en comparación al caso anterior (simulación de 131072 individuos). Esto sucede debido a que no se puede incrementar el largo de la lista de forma lineal porque la eficiencia de la simulación se degradaría. Este comportamiento es ocasionado porque si el largo de la lista se incrementa linealmente el costo de construción y mantención de la estructura es demasiado elevado. Por otro lado, se puede observar que la magnitud de los umbrales de desbalance también deben incrementarse a medida que la cantidad de cores aumenta. Esto significa que a medida que la cantidad de cores se incrementa se debe ser más permisivo con la ejecución de la simulación distribuida.

6.2.2. Speedup

En la figura 6.21 se muestra la ganancia obtenida en términos de *speedup* para la simulación de 262144 individuos. Los valores utilizados para construir la curva fueron extraídos del cuadro 6.6. En general, se podría decir que el comportamiento de la aplicación es relativamente bueno, teniendo en cuenta que existe un decremento del *speedup* cuando se incrementa la cantidad de cores. Se puede observar que con la utilización de $2, 4, 8, 16$ y hasta 32 cores, el comportamiento de la curva de *speedup* se acerca al bastante al lineal. Por otro lado, cuando utilizamos 64 y 128 cores, el comportamiento de la curva de *speedup* se decrementa. Esto sucede debido a que los tiempos de comunicación se han incrementado a medida que la cantidad de cores se incrementa. También se debe tener en cuenta el costo computacional que significa mantener la estructura de datos, y más aún, sobre una arquitectura distribuida.

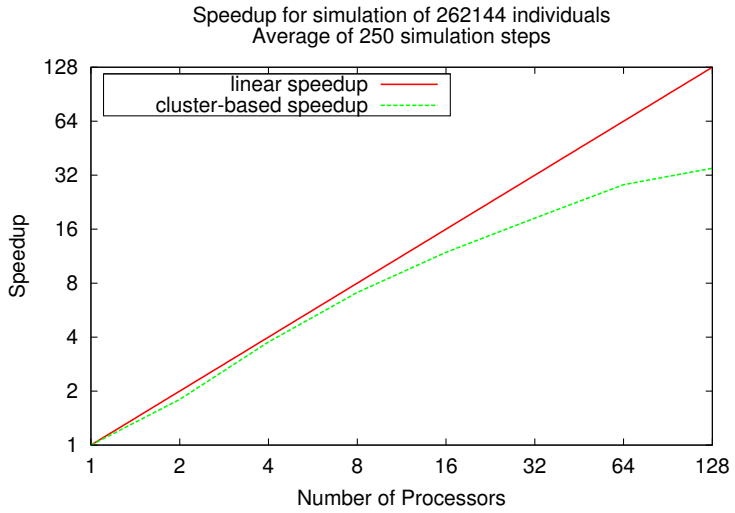


Figura 6.21: Speedup para la simulación de 262144 individuos.

6.2.3. Ajuste dinámico de la carga de trabajo

A continuación se exhibirá la evolución de las cargas de trabajo en función del tiempo para la simulación distribuida de 262144 individuos utilizando 128 cores en 250 pasos de simulación. Se analizarán los resultados para tres distintos umbrales de desbalance: 10 %, 20 % y 30 % de la carga de trabajo local de cada core.

En la figura 6.22 se muestra la evolución de la carga de trabajo en función del tiempo por medio de la utilización de un umbral de desbalance del 10 %. Este 10 % significa que la carga de trabajo se debe mantenerse dentro de los rangos [1844 : 2252] individuos por core. Se puede apreciar la existencia de picos que indican migraciones progresivas de individuos entre PLs, que implica la sobrecarga e infrautilización de recursos de cómputo. También se puede observar que las cargas de trabajo tienden a converger a la media de 2048 individuos por core.

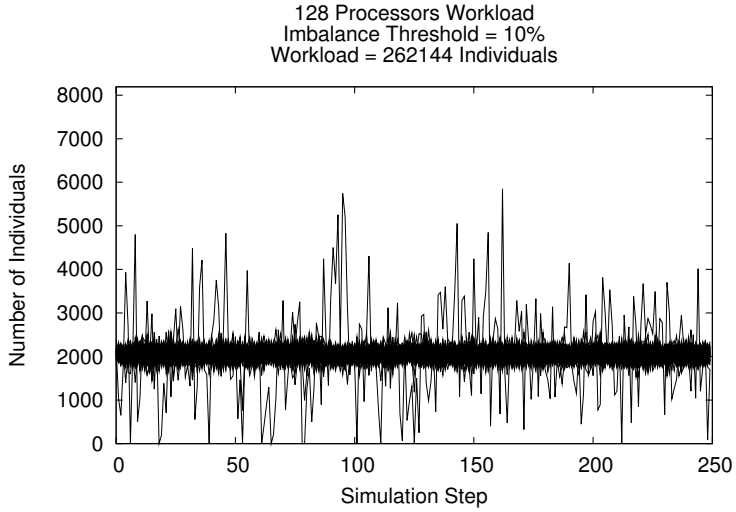


Figura 6.22: Carga de trabajo para la simulación 262144 individuos usando 128 cores con un 10 % de umbral de desbalance.

En la figura 6.23 se muestra la evolución de la carga de trabajo en función del tiempo por medio de la utilización de un umbral de desbalance del 20 %. Este 20 % significa que la carga de trabajo debe mantenerse dentro de los rangos [1637 : 2457] individuos por core. El comportamiento es similar al caso anterior, apreciándose una tendencia a mantener la carga de trabajo convergiendo a la media.

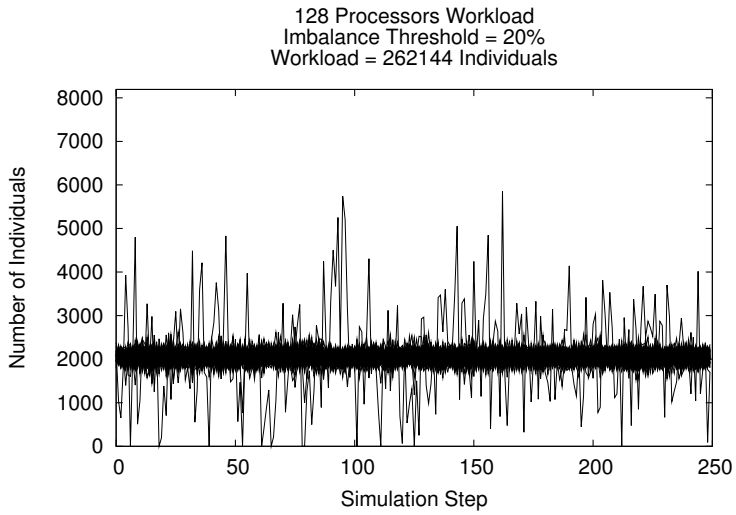


Figura 6.23: Carga de trabajo para la simulación 262144 individuos usando 128 cores con un 20 % de umbral de desbalance.

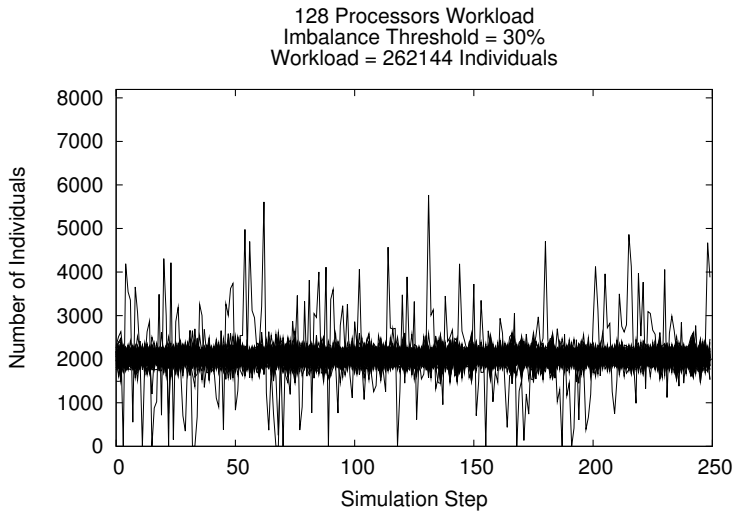


Figura 6.24: Carga de trabajo para la simulación 262144 individuos usando 128 cores con un 20 % de umbral de desbalance.

En la figura 6.24 se muestra la evolución de la carga de trabajo en función del tiempo por medio de la utilización de un umbral de desbalance del 30 %. Este 30 % significa que la carga de trabajo debe mantenerse dentro de los rangos [1434 : 2662] individuos por core. También se sigue observando la misma tendencia, es decir,

la carga de trabajo intenta converger a la media. Se puede observar la existencia de picos, pero como se ha comentado anteriormente, son causados principalmente por migraciones progresivas de individuos entre PLs.

6.3. Simulación de 524288 Individuos

A continuación se expondrán los resultados obtenidos para la simulación distribuida de 524288 individuos en distintos escenarios. Los escenarios están formados por distintos factores que influyen en el rendimiento de la simulación distribuida, tales como: *cantidad de cores*, *tamaño de los radios cobertores* y *umbrales de desbalance*. Las simulaciones fueron ejecutadas utilizando $1, 2, 4, 8, 16, 32, 64$ y 128 cores. Los rangos de los radios cobertores son mostrados en el cuadro 6.7. Se han utilizado tres distintos umbrales de desbalance: 10 %, 20 % y 30 % de la carga local asignada a cada core. Los resultados han sido obtenidos a partir de la ejecución de 250 pasos de simulación.

	R_{min}	R_{max}
$\mathbf{conf}_1$	$1{,}10R_{MAX}$	$1{,}15R_{MAX}$
$\mathbf{conf}_2$	$1{,}15R_{MAX}$	$1{,}20R_{MAX}$
$\mathbf{conf}_3$	$1{,}20R_{MAX}$	$1{,}25R_{MAX}$
$\mathbf{conf}_4$	$1{,}25R_{MAX}$	$1{,}30R_{MAX}$
$\mathbf{conf}_5$	$1{,}30R_{MAX}$	$1{,}35R_{MAX}$

Cuadro 6.7: Radios cobertores para la simulación de 524288 individuos.

6.3.1. Tiempos de ejecución

A continuación se exhibirán los tiempos de ejecución (descompuestos en tiempos de comunicación/cómputo) para la simulación distribuida de 524288 individuos usando distintas cantidades de cores. Los resultados mostrados han sido obtenidos a partir del promedio de 250 pasos de simulación. Para los casos en los que se utiliza más de un core, se utilizan tres umbrales de desbalance 10 %, 20 % y 30 % de la carga local asignada a cada core.

En la figura 6.25 se muestran los resultados obtenidos para la simulación de 524288 individuos por medio de la utilización de 1 core. Se puede apreciar resultados muy similares utilizando todas las configuraciones. También se puede apreciar levemente un decremento del rendimiento a medida que se aumenta la magnitud de los radios cobertores. En términos de optimización, se puede apreciar que por medio de la utilización de la configuración $\mathbf{conf}_1$ se pueden obtener mejores resultados.

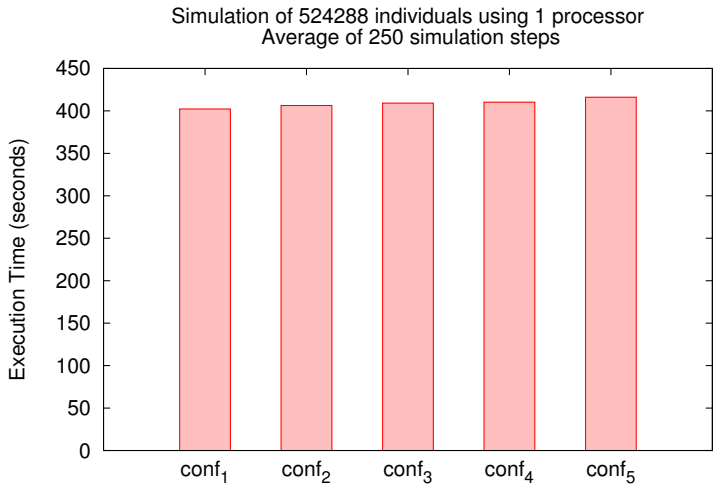


Figura 6.25: Simulación de 524288 individuos usando 1 core.

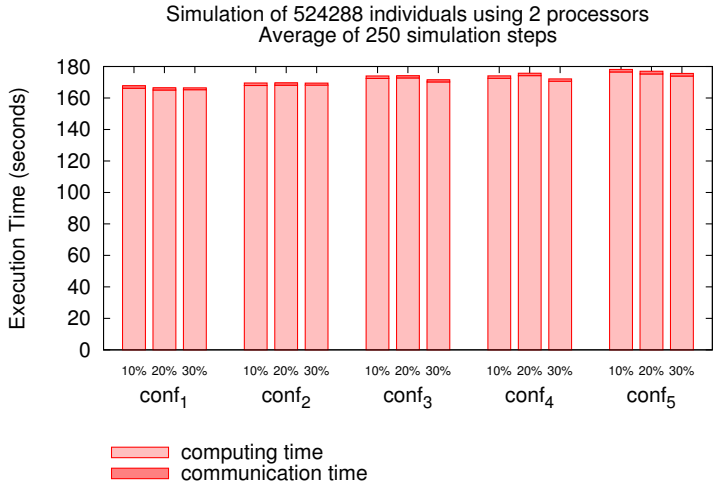


Figura 6.26: Simulación de 524288 individuos usando 2 cores.

En la figura 6.26 se exhiben los tiempos de comunicación/cómputo para la simulación distribuida de 524288 individuos por medio de la utilización de 2 cores. Se puede apreciar que a medida que se incrementa la magnitud de los radios cobertores el rendimiento de la simulación distribuida se decrementa levemente. En términos de optimización, se puede observar que por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ con umbrales de desbalance del 20 % y 30 % se pueden

obtener mejores resultados. Los tiempos de comunicación son insignificantes en comparación a los tiempo de cómputo.

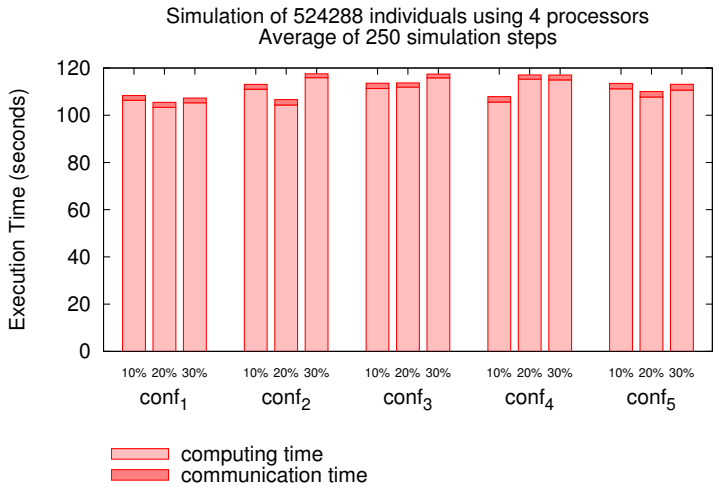


Figura 6.27: Simulación de 524288 individuos usando 4 cores.

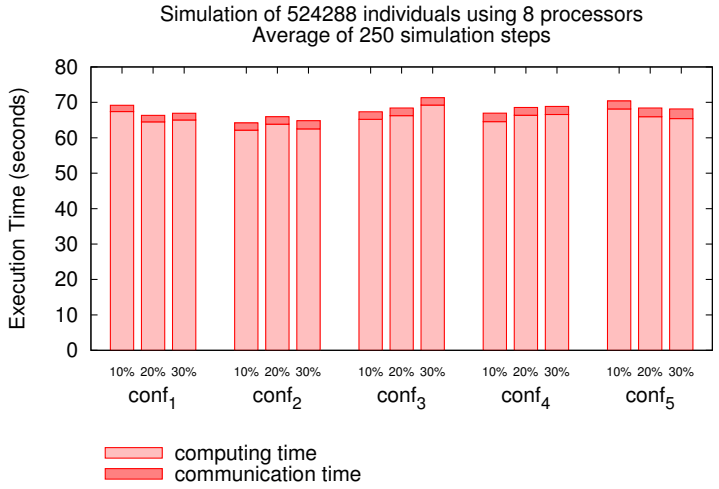


Figura 6.28: Simulación de 524288 individuos usando 8 cores.

En la figura 6.27 se muestran los resultados obtenidos para la simulación distribuida de 524288 individuos utilizando 4 cores. Se puede observar que por medio de la utilización de distintas configuraciones se pueden obtener

resultados bastante aceptables. En términos de optimización, Se puede apreciar una ganancia al utilizar las configuraciones $\mathbf{conf}_1$ y $\mathbf{conf}_2$, ambas con un umbral de desbalance del 20 %, y $\mathbf{conf}_4$ con un umbrales de desbalance 10 %. Los tiempos de comunicación son insignificantes en comparación a los tiempos de cómputo.

En la figura 6.28 se muestran los resultados obtenidos para la simulación distribuida de 524288 individuos por medio de la utilización de 8 cores. Se pueden apreciar resultados bastante variables dependiendo la configuración utilizada. En términos de optimización, se puede apreciar que por medio de la utilización de la configuración $\mathbf{conf}_2$ con umbrales de desbalance del 10 % y 30 % se exhiben muy buenos resultados. Los tiempos de comunicación se han incrementado levemente en comparación al caso anterior (ver Fig. 6.27), pero no representan un factor significativo en los resultados de la simulación distribuida.

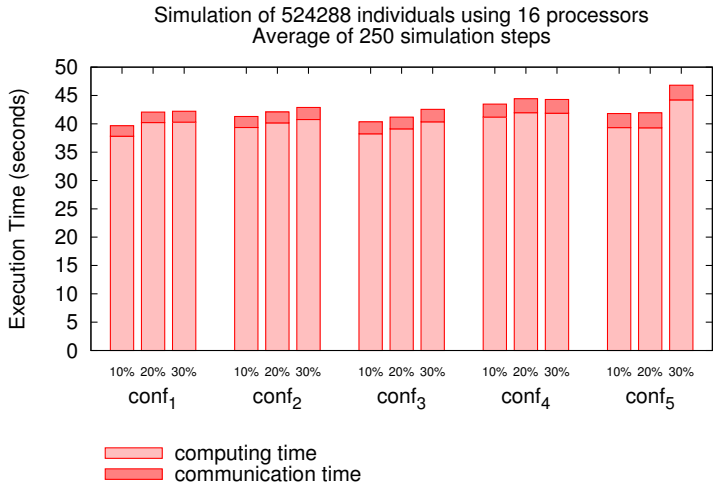


Figura 6.29: Simulación de 524288 individuos usando 16 cores.

En la figura 6.29 se muestran los resultados obtenidos para la simulación distribuida de 524288 individuos por medio de la utilización de 16 cores. Claramente se puede observar la influencia de la magnitud de los umbrales de desbalance en los resultados de la simulación. Esto significa que los tiempos de ejecución crecen acoplados a la magnitud de los umbrales de desbalance. En términos de optimización, se puede observar que por medio de la utilización de las configuraciones: $\mathbf{conf}_1$ y $\mathbf{conf}_3$, ambas con un umbral de desbalance del 10 %, se han obtenido los mejores resultados. Los tiempos de comunicación se han incrementado pero en relación a los tiempos de cómputo aún se mantienen despreciables.

En la figura 6.30 se muestran los resultados obtenidos para la simulación

distribuida de 524288 individuos por medio de la utilización de 32 cores. En términos optimización, se puede observar que por medio de la utilización de las configuraciones: $\mathbf{conf}_2$ con un umbral de desbalance del 20 % y $\mathbf{conf}_3$ con un umbral de desbalance del 30 % se han obtenido los mejores resultados. Los tiempos de comunicacion se han incrementado en comparación a los casos expuestos previamente.

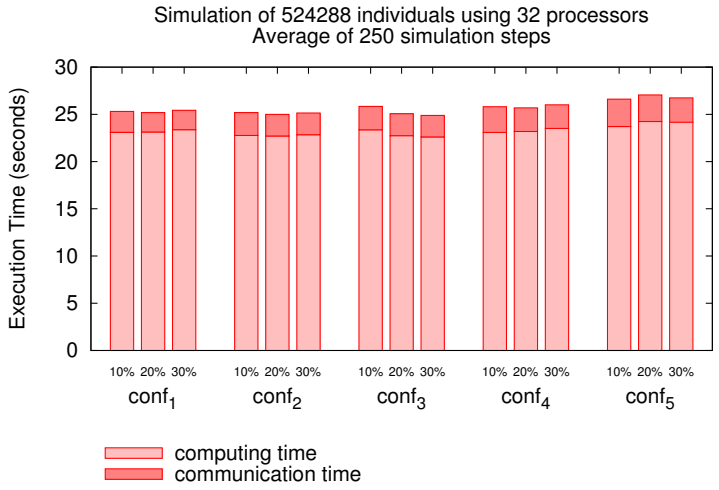


Figura 6.30: Simulación de 524288 individuos usando 32 cores.

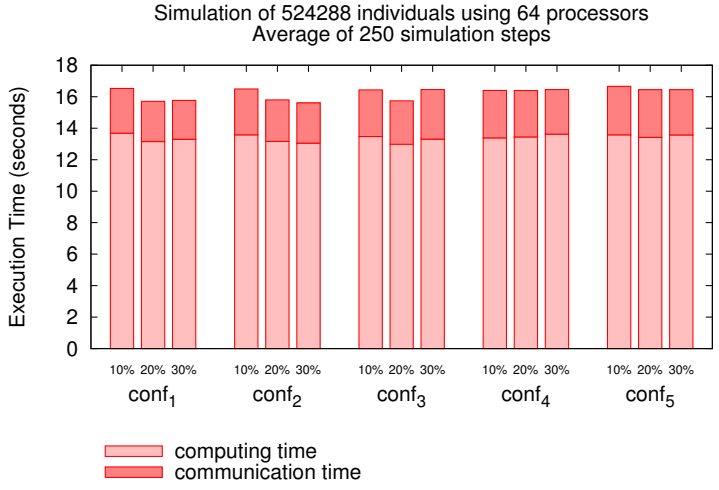


Figura 6.31: Simulación de 524288 individuos usando 64 cores.

En la figura 6.31 se muestran los resultados obtenidos para la simulación distribuida de 524288 individuos por medio de la utilización de 64 cores. Para todas las configuraciones, los resultados obtenidos son bastante aceptables. En términos de optimización, se puede apreciar que por medio de la utilización de las configuraciones: $\mathbf{conf}_2$ con un umbral de desbalance del 30 % y $\mathbf{conf}_3$ con un umbral de desbalance del 20 % se han obtenido los mejores resultados. Los tiempos de comunicación se han incrementado, pero en comparación a los tiempos de cómputo se mantienen dentro de rangos aceptables.

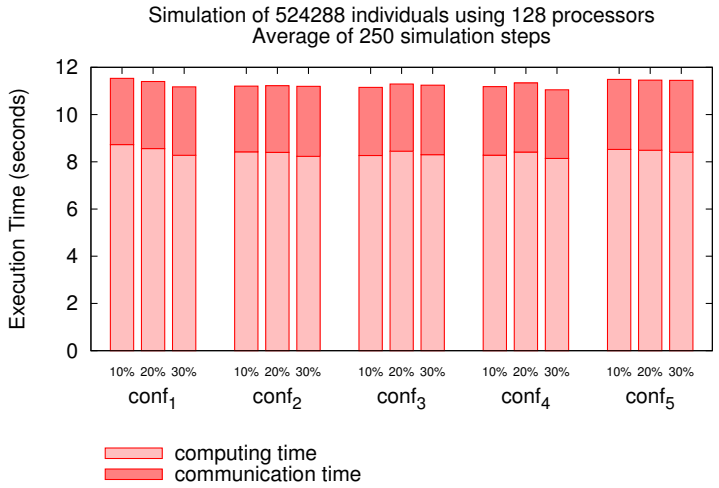


Figura 6.32: Simulación de 524288 individuos usando 128 cores.

En la figura 6.32 se muestran los resultados obtenidos para la simulación distribuida de 524288 individuos utilización 128 cores. Se pueden observar resultados bastante aceptables por medio de la utilización de todas las configuraciones. En términos de optimización, se puede apreciar una ganancia al utilizar las configuraciones: $\mathbf{conf}_3$ con un umbral de desbalance del 10 % y $\mathbf{conf}_4$ con un umbral de desbalance del 30 %. Los tiempos de comunicación se han incrementado sustancialmente en comparación a los casos expuestos anteriormente, pero aún se mantienen dentro de rangos bastante aceptables.

En el cuadro 6.8 se exhiben los mejores resultados obtenidos para simulación distribuida de 524288 individuos en distintos escenarios. Los resultados han sido generados a partir del promedio de 250 pasos de simulación. Claramente se puede apreciar que al aumentar la cantidad de cores el tiempo de ejecución se decrementa significativamente. También se puede apreciar que los tiempos de comunicación se han incrementado en comparación al caso anterior (simulación de 262144). Esto sucede debido a que incrementar la carga de trabajo implica un incremento en el

tamano del *meta-cluster* y de las fronteras que comparte con otros *meta-clusters* pertenecientes a otros LPs. También se puede apreciar como el tamano del radio cobertor influye en la cantidad de *clusters* obtenidos.

cores	**conf**	$\mathbf{T}_{computo}$	$\mathbf{T}_{comunicacion}$	**umbral de desbalance**	*clusters*
1	$\mathbf{conf}_1$	402,266	-	-	2898
2	$\mathbf{conf}_1$	165,03207	1,41593	20 %	2894
4	$\mathbf{conf}_1$	103,34568	2,09832	20 %	2901
8	$\mathbf{conf}_2$	62,16849	2,03891	10 %	2594
16	$\mathbf{conf}_1$	37,80292	1,85458	10 %	2900
32	$\mathbf{conf}_3$	22,60417	2,28073	30 %	2337
64	$\mathbf{conf}_2$	13,05105	2,55715	30 %	2596
128	$\mathbf{conf}_4$	8,13937	2,90843	30 %	2596

Cuadro 6.8: Sumario de los mejores resultados obtenidos para la simulación de 524288 individuos.

Además, se puede observar que a medida que se incrementa la cantidad de cores, las magnitudes de los radios cobertores deben incrementarse para no degradar el funcionamiento de la estructura de datos. Por otro lado, se puede apreciar que los umbrales de desbalance para los casos en los cuales se utilizan $32, 64$ y 128 cores, deben ser más de magnitud más amplia que en el resto de los casos. Esto sucede debido a que a medida que se incrementa la cantidad cores se debe ser más permisivo con el progreso de la simulación distribuida, debido a que la probabilidad de ocurrencia de migraciones progresivas de individuos entre PLs se incrementa y forzar el ajuste de la carga de trabajo perjudica al rendimiento de la simulación distribuida.

6.3.2. Speedup

En la figura 6.33 se muestra la ganancia obtenida en términos de *speedup* para la simulación distribuida de 524288 individuos. Los resultados fueron generados a partir del promedio de 250 pasos de simulación. En los casos en los cuales se utilizan $2, 4, 8$ y 16, se pueden apreciar resultados bastantes cercanos al *speedup* lineal. Por otra parte, en los casos en los cuales se utilizan $32, 64$ y 128 el rendimiento decrece a medida que se incrementa la cantidad de cores. Esto puede ser principalmente causado debido a que aplicar una política de balance de carga dinámico no implica el balance de cómputo.

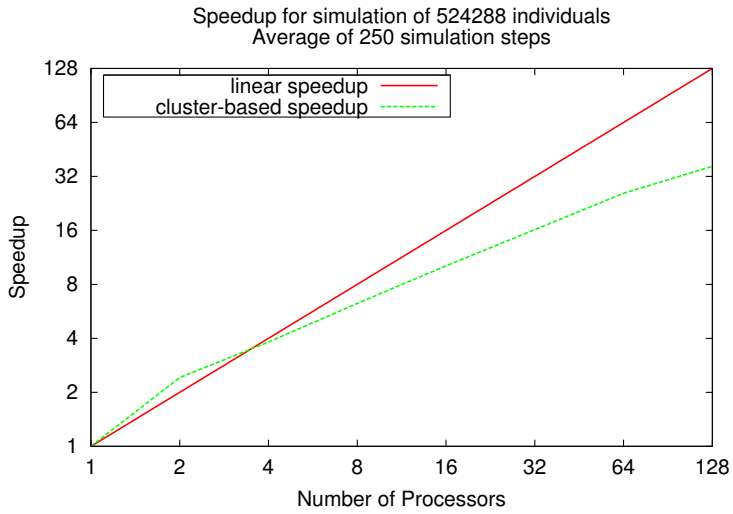


Figura 6.33: Speedup para la simulación de 524288 individuos.

En esta clase de simulación distribuida, un paso simulación consiste en utilizar los vecindarios de cada *cluster* para evaluar a los individuos pertenecientes a *cluster* locales. En algunos casos, un PL comparte fronteras con más PLs que otros, lo que implica que al momento de ejecutar un paso de simulación se realizará más cómputo local. Por otro lado, también hay que tener en cuenta que las comunicaciones involucradas al momento de la simulación también pueden influir en el resultado de la simulación, es decir, si PL_1 comparte frontera con otros n PLs, y PL_2 comparte frontera con otros m PLs y $n > m$, los tiempos de comunicación y cómputo de PL_1 serán superiores que los de PL_2.

6.3.3. Ajuste dinámico de la carga de trabajo

A continuación se mostrará la evolución de las cargas de trabajo para la simulación de 524288 individuos sobre 250 pasos de simulación utilizando 128 cores. Se analizarán los resultados para tres diferentes umbrales de desbalance: $10\,\%, 20\,\%$ y $30\,\%$ de la carga de trabajo local asignada a cada core.

En la figura 6.34 se muestra la evolución de la carga de trabajo en función del tiempo con un umbral de desbalance del $10\,\%$ de la carga de trabajo local. Esto significa que la carga de trabajo de cada core debe mantenerse entre los rangos $[3687 : 4505]$ individuos. En la gráfica se pueden apreciar bastantes picos que indican el desbalance de la carga en algunos periodos de tiempo. Esto sucede debido a que la migraciones de individuos entre PLs implica la sobrecarga (sobre la media) y la infrautilización (bajo la media) de los recursos de cómputo. Se puede observar que a medida que la simulación avanza la carga de trabajo intenta

converger a la media.

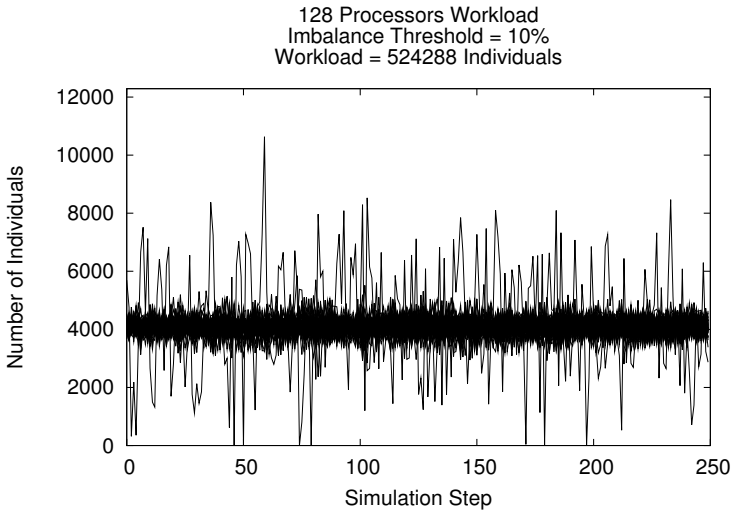


Figura 6.34: Carga de trabajo para la simulación 524288 individuos usando 128 cores con un 10 % de umbral de desbalance.

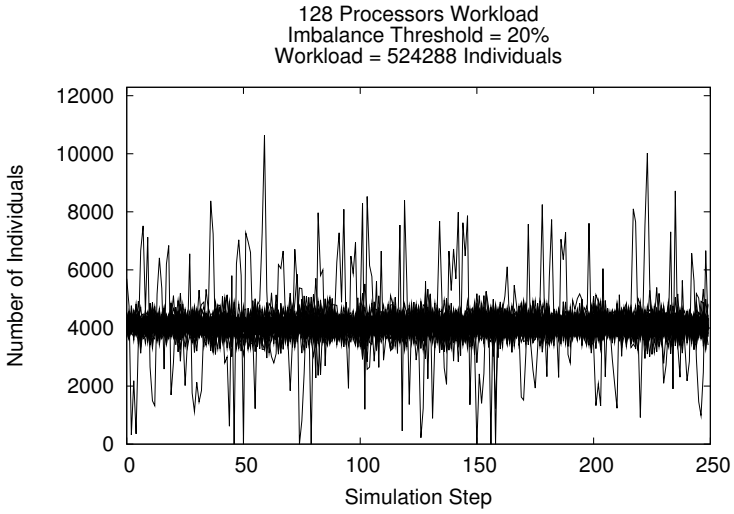


Figura 6.35: Carga de trabajo para la simulación 524288 individuos usando 128 cores con un 20 % de umbral de desbalance.

En la figura 6.35 se puede apreciar la evolución de la carga de trabajo en función del tiempo utilizando un umbral de desbalance del 20 %. Esto significa

que la carga de trabajo de cada core debe mantenerse entre los rangos [3277 : 4915] individuos. Se puede apreciar un comportamiento similar al caso anterior: existencia de picos y convergencia de la carga de trabajo a la media.

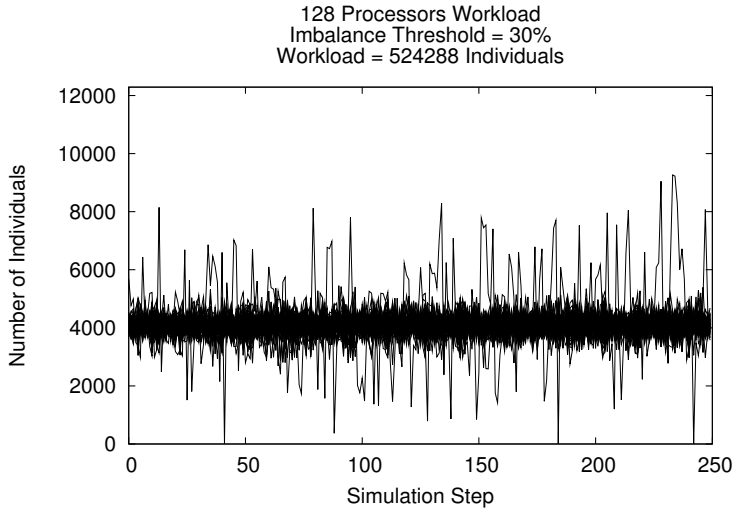


Figura 6.36: Carga de trabajo para la simulación 524288 individuos usando 128 cores con un 20 % de umbral de desbalance.

En la figura 6.36 se muestra la evolución de la carga de trabajo en función del tiempo utilizando un 30 % de umbral de desbalance. Esto significa que la carga de trabajo por core debe mantenerse entre los rangos [2808 : 5348] individuos. En este caso se puede apreciar que los picos alcanzados por la carga de trabajo no son tan amplios como los mostrados en los dos casos anteriormente expuestos. Esto puede ser causado principalmente porque permitir que la ejecución de la simulación distribuida fluya durante el tiempo puede ser más provechoso que tratar de forzar el balance de carga. En este tipo de simulaciones la carga de trabajo (individuos) se mantienen en continuo movimiento entre los LPs, lo que puede causar que la carga de trabajo se ajuste dinámica y automáticamente a partir del movimiento inherente en el modelo.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones generales

Las simulaciones orientadas al individuo pueden obtener un gran provecho al ejecutarse en un ambiente paralelo/distribuido. Por lo general, este tipo de simulaciones son bastante complejas y requieren un gran poder de cómputo y un enorme espacio de memoria. Esto se traduce en un gran costo en términos de tiempos de ejecución en un entorno secuencial.

Distribuir la simulación sobre múltiples unidades de cómputo puede ser una tarea bastante difícil, ya que existe una infinidad de configuraciones válidas para este tipo de problemas. Cuando se utiliza un método de particionamiento de grano grueso, la distribución de las particiones es bastante fácil ya que se asigna una única partición a cada unidad de cómputo. En cada paso de simulación, calcular el nuevo estado de un único individuo consiste en evaluar a todos los individuos pertenecientes a la partición local más los individuos pertenecientes a las fronteras que comparte con otras unidades de cómputo. Este proceso es extremadamente ineficiente debido a que algunos individuos ni siquiera se encuentra a su dentro de su área de visibilidad.

Si se utiliza un método de particionamiento de grano fino, la tarea se vuelve un poco más compleja. Se debe tener en cuenta que cada partición comparte frontera con sus particiones adyacentes. Administrar inadecuadamente el conjunto de particiones puede acarrear un incremento de los tiempos de comunicación (ya que las fronteras deben ser enviadas como un mensaje). Este tipo de particionamiento soluciona el problema de la evaluación exhaustiva en cada paso de simulación, ya que cada *micro-partición* comparte frontera solamente con las *micro-particiones* adyacentes.

Por otro lado, si el MoI presenta patrones de movimiento, se debe ser

capaz de mantener el equilibrio de la carga de trabajo entre las unidades de cómputo a medida que la simulación evoluciona en función del tiempo. Los patrones de movimiento causan migraciones de individuos entre procesos lógicos logrando la sobrecarga o la infrautilización de uno o más recursos de cómputo. Estas migraciones pueden provocar un decremento en el rendimiento global de la aplicación de simulación distribuida. El primer paso para implementar un mecanismo de balance de carga dinámico consiste en detectar el desequilibrio. Esto puede ser logrado por medio de mecanismos basados en umbrales, probabilísticos o de tiempos de ejecución. Una vez detectado el desbalance, se pueden elegir entre dos opciones: 1) migración de individuos entre PLs o 2) re-particionamiento y re-asignación.

Las principales conclusiones extraídas del presente trabajo pueden ser resumidas en:

- Se ha desarrollado en método de particionamiento de grano fino basado en particiones compactas. Estas particiones están definidas a partir del máximo rango de visión de los individuos. Esto permite crear un conjunto de particiones en las cuales los individuos de una partición sólo pueden ver a los individuos pertenecientes a particiones adyacentes. Además, existe la noción de proximidad entre particiones, lo que permite descartar rápidamente al momento de buscar el vecindario de una partición. Todo esto permite eliminar el cómputo innecesario involucrado en la búsqueda de los vecinos más influyentes, decrementado el orden del algoritmo secuencial de $O(n^2)$ a $O(km)$, en donde k es la cantidad de particiones y m la cantidad de individuos por partición.

- El método de particionamiento desarrollado en el presente trabajo se caracteriza por su adaptabilidad, es decir, a medida que la simulación avanza las particiones se van ajustando a la forma del sistema simulado (banco de peces), evitando particiones vacías.

- Se ha utilizado una estructura de datos de alto nivel, generalmente utilizada para búsqueda por similitud en espacios métricos, para almacenar a los individuos de la simulación. Esto permite administrar eficientemente las particiones y a los individuos a medida que la simulación avanza. La estructura de datos es una lista enlazada de particiones que además almacena alguna información adicional que permite descartar una partición rápidamente al momento de buscar el vecindario de una partición.

- Se ha desarrollado un método de agrupación de particiones en *meta-particiones*. Estas *meta-particiones* son formadas a partir de la agrupación de particiones en conjunto de particiones contiguas. La cardinalidad del conjunto de *meta-particiones* debe ser igual a la cantidad de unidades

de cómputo de la arquitectura paralela/distribuida en donde se ejecuta la simulación. La formación de grupos de particiones por proximidad se realiza debido a que en cada paso de simulación se debe calcular el vecindario de cada partición, y una distribución inadecuada de las particiones puede incrementar excesivamente los tiempos de comunicación.

- Se ha adaptado la estructura de datos para su utilización en un entorno distribuido. Esto permite identificar las particiones almacenadas local y remotamente, intersectarlas y calcular el vecindario de cada partición sin inconvenientes.

- La mejora del orden del algoritmo secuencial ha permitido lograr una curva de escalabilidad cercana al *speedup* lineal.

- Se ha logrado representar el movimiento de un banco de peces en un ambiente tridimensional de forma realista, y se ha disminuido considerablemente los tiempo de ejecución de este tipo de modelos gracias a su implementación en una arquitectura paralela/distribuida.

- Se ha logrado incluir un criterio de proximidad entre particiones. Esto es debido a que cada partición posee una tabla de distancia en la cual se almacenan las distancia al resto de las particiones. En un entorno paralelo/distribuido, cada unidad de cómputo almacena la cabecera de la lista por completo, más un conjunto de particiones asociadas por proximidad.

- Se ha incluido un mecanismo de balance dinámico de carga basado en la proximidad entre particiones. El proceso de detección del desequilibrio se realiza por medio de un método basado en umbrales, el cual indica cuando existen un desbalance local. El mecanismo de balance de carga dinámico consiste en re-agrupar particiones por proximidad. En este proceso no se incluye paso de mensajes, ya que posteriormente se ejecutará el proceso de *migración*, que es el encargado de enviar los individuos almacenados localmente que pertenecen a otras unidades de cómputo.

- Se ha demostrado experimentalmente la factibilidad del método de particionamiento implementado en el presente trabajo, y su aplicación en simulación distribuida.

- Se ha demostrado experimentalmente que el mecanismo de balance de carga dinámicamente ajusta la carga de trabajo a medida que la simulación avanza.

7.2. Trabajos futuros

A continuación se presentarán los trabajos futuros propuestos para el presente trabajo:

- Comprobar la viabilidad de los métodos de particionamiento, asignación y balance de carga dinámico con otros modelos orientados al individuos espacialmente explícitos que presentan patrones de movimiento, tales como: *boids*, *self-propelled particles*, *social force model*, etc. Es importante demostrar que los mecanismos implementados en el presente trabajo son un aporte para la simulación distribuida de este tipo de modelos en general.

- Realizar un análisis de los patrones de comunicación, con el objetivo de tratar de alcanzar un balance en las comunicaciones.

- Experimentalmente, se ha descubierto que las particiones interiores comparten con más fronteras que las particiones exteriores. Esto produce que aunque exista un balance de carga de trabajo, no existe un balance en el cómputo. Debido a esto, es necesario establecer una política eficiente de balance de cómputo.

7.3. Artículos publicados en congresos

A continuación se expondrán y analizarán los artículos publicados en congresos que forman parte de la validación de los resultados obtenidos en el presente trabajo:

Este artículo (ver Cuadro 7.1) forma parte de la etapa preliminar del presente trabajo (tesina). En este trabajo se realiza un estudio acerca de como la complejidad del modelo influye sobre la escalabilidad en la simulación distribuida. El nivel de complejidad del modelo es incrementado por medio de la inclusión de estímulos externos, tales como: *obstáculos*, *depredadores* y *fronteras del espacio de simulación*.

Estado	Aceptada
Autores	Roberto Solar, Remo Suppi, and Emilio Luque
Título	High performance individual-oriented simulation using complex models
Año	2010
Conferencia	International conference on computer science, ICCS 2010, Amsterdam, Holanda

Cuadro 7.1: ICCS 2010

El trabajo desarrollado para este artículo (ver Cuadro 7.2) consiste en establecer un mecanismo robusto de particionamiento basado en *clustering* y un método de distribución de particiones basado en proximidad para simulaciones distribuidas de bancos de peces. Los resultados obtenidos cumplieron con las expectativas, por lo que se continuo trabajando sobre esta estrategia.

Estado	Aceptada
Autores	Roberto Solar, Remo Suppi, and Emilio Luque
Título	High performance distributed cluster-based individual-oriented fish school simulation
Año	2011
Conferencia	International conference on computer science, ICCS 2011, Nanyang, Singapur

Cuadro 7.2: ICCS 2011

En el trabajo desarrollado para este artículo (ver Cuadro 7.3) se implementó un mecanismo de balance de carga dinámico basado en proximidad para un método de particionamiento basado en *clustering* (ver Cuadro 7.2) en simulaciones distribuidas de bancos de peces. Experimentalmente, se ha demostrado que gracias a la implementación de un algoritmo de balance de carga dinámico el tamano de la carga de trabajo se mantiene similar en cada unidad de cómputo a medida que la simulación avanza.

Estado	Aceptada
Autores	Roberto Solar, Remo Suppi, and Emilio Luque
Título	Proximity load balancing for distributed cluster-based individual-oriented fish school simulations
Año	2011
Conferencia	International conference on computer science, ICCS 2012, Omaha, Nebraska, Usa

Cuadro 7.3: ICCS 2012

7.4. Otras aportaciones

En el trabajo desarrollado en este artículo (ver Cuadro 7.4) se realiza una comparación en términos de escalabilidad entre una simulación distribuida basada en un MoI de *fish school* y una simulación distribuida basada en un MoI *presa-depredador*.

Estado	Aceptada
Autores	Roberto Solar, Remo Suppi, and Emilio Luque
Título	Simulación distribuida de modelos presa-depredador en sistemas orientados al individuo
Año	2010
Conferencia	Jornadas de Paralelismo, JP 2010, Valencia, Espana

Cuadro 7.4: JP 2009

Este trabajo (ver Cuadro 7.5) se encuentra pendiente de respuesta. En este trabajo se realiza un estudio acerca de la adaptabilidad de método de particionamiento basado en *clustering*. Las características adaptativas del método de particionamiento se logran por medio de la implementación de mecanismos de *eliminación de solapamiento* entre *clusters* y la creación cooperativa de *clusters*.

Estado	Esperando respuesta
Autores	Roberto Solar, Remo Suppi, and Emilio Luque
Título	Adaptive cluster-based partitioning for distributed individual-oriented fish school simulations
Año	2012
Conferencia	International Conference on Parallel Problem Solving From Nature, PPSN 2012, Taormina, Italy

Cuadro 7.5: PPSN 2012

Bibliografía

[1] I. Aoki. A simulation experiment on individual differences in schooling behaviour of fish. *Bulletin of the Japanese Society of Scientific Fisheries*, 52:1115 1119, 1986.

[2] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345 405, September 1991.

[3] J.E. Beck and D.P. Siewiorek. Simulated annealing applied to multicomputer task allocation and processor specification. In *Parallel and Distributed Processing, 1996. Eighth IEEE Symposium on*, pages 232 239, oct 1996.

[4] M.J. Berger and S.H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *Computers, IEEE Transactions on*, C-36(5):570 580, may 1987.

[5] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(3):7280 7287, 2002.

[6] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Softw. Eng.*, 5:440 452, 1979.

[7] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Commun. ACM*, 24(4):198 206, 1981.

[8] E. Chavez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, pages 75 , Washington, DC, USA, 2000. IEEE Computer Society.

[9] S. Chowdhury and B. Sengupta. Threshold-based load balancing in an atm switch consisting of parallel output buffered switch elements. In *Global Telecommunications Conference, 1994. GLOBECOM '94. Communications: The Global Bridge., IEEE*, 1994.

[10] B. Cosenza, G. Cordasco, R. De Chiara, and V. Scarano. Distributed load balancing for parallel agent-based simulations. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 62 69, feb 2011.

[11] Phyllis Crandall and Michael J. Quinn. Block data decomposition for data-parallel programming on a heterogeneous workstation network. In *HPDC*, pages 42 49, 1993.

[12] Phyllis E. Crandall and Michael J. Quinn. Data partitioning for networked parallel processing, 1993.

[13] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7(2):279 301, 1989.

[14] Andras Czirok and Tamas Vicsek. Collective behavior of interacting self-propelled particles. *PHYSICA A*, 281:17, 2000.

[15] Chris Ding and Yun He. A ghost cell expansion method for reducing communications in solving pde problems. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing 01, pages 50 50, New York, NY, USA, 2001. ACM.

[16] Nassrin Eftekhari, Farid Haji Zeinalabedin, and Abolfazl Torghi Haghighat. A novel threshold-based dynamic load balancing algorithm using mobile agent in distributed system. In *Computational Intelligence and Information Technology*, volume 250 of *Communications in Computer and Information Science*, pages 103 109. Springer Berlin Heidelberg, 2011.

[17] U. Erra, B. Frola, V. Scarano, and I. Couzin. An efficient gpu implementation for large scale individual-based simulation of collective behavior. In *High Performance Computational Systems Biology, 2009. HIBI '09. International Workshop on*, pages 51 58, oct 2009.

[18] Richard M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30 53, 1990.

[19] Richard M. Fujimoto. *Parallel and distributed simulation systems (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 1 edition, jan 2000.

[20] M. Gardner. Mathematical games: The fantastic combinations of john conway s new solitaire game life . *Scientific American*, 223(4):120 123, 1970.

[21] W.A. Greene. Dynamic load-balancing via a genetic algorithm. In *Tools with Artificial Intelligence, Proceedings of the 13th International Conference on*, pages 121 128, nov 2001.

[22] R. Gregory, R. Paton, J. Saunders, and Q.H. Wu. Parallelising a model of bacterial interaction and evolution. *Biosystems*, 76:121 131, 2004.

[23] Volker Grimm and Steven F. Railsback. *Individual-based Modeling and Ecology*. Princeton University Press, Princeton, 2005.

[24] Hermann Haken. *Synergetics. An introduction. Nonequilibrium phase transitions and self organization in physics, chemistry and biology*. Springer series in synergetics. Springer, Berlin, New York, 3rd ed. edition, 1983.

[25] D. Helbing, Moln, I. J. Farkas, and K. Bolay. Self-organizing pedestrian movement. *Environment and Planning B: Planning and Design*, 28(3):361 383, 2001.

[26] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical Review E*, 51:4282, 1995.

[27] E E Holmes, M A Lewis, J E Banks, and R R Veit. Partial differential equations in ecology : Spatial interactions and population dynamics. *Ecology*, 75(1):17 29, 1994.

[28] Andreas Huth and Christian Wissel. The simulation of the movement of fish schools. *Journal of Theoretical Biology*, 156(3):365 385, 1992.

[29] Andreas Huth and Christian Wissel. The simulation of fish schools in comparison with experimental data. *Ecological Modelling*, 75-76(0):135 146, 1994.

[30] Y. Inada and K. Kawachi. Order and flexibility in the motion of fish schools. *J. Theor. Biol.*, 214:371 387, 2002.

[31] F. Jopp, H. Reuter, and B. Breckling. *Modelling Complex Ecological Dynamics*. Springer, 2011.

[32] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942 1948, 1995.

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671 680, 1983.

[34] Tanya Kostova, Tina Carlsen, and Jim Kercher. Individual-based spatially-explicit model of an herbivore and its resource: the effect of habitat reduction and fragmentation. *Comptes Rendus Biologies*, 327(3):261 276, 2004.

[35] Averill M. Law and W. David Kelton. *Simulation modeling and analysis*. McGraw-Hill, 3rd edition, 2000.

[36] Hong Li, Allison Kolpas, Linda Petzold, and Jeff Moehlis. Parallel simulation for a fish schooling model on a general-purpose graphics processing unit. *Concurr. Comput. : Pract. Exper.*, 21(6):725 737, apr 2009.

[37] Frank C. H. Lin and Robert M. Keller. The gradient model load balancing method. *IEEE Trans. Softw. Eng.*, 13(1):32 38, 1987.

[38] Jiming Liu, Michael B. Dillencourt, Lubomir F. Bic, Daniel Gillen, and Arthur D. Lander. Distributed individual-based simulation. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par 09, pages 590 601, Berlin, Heidelberg, 2009. Springer-Verlag.

[39] A. J. Lotka. Elements of physical biology. *Nature*, 116:461 461, 1925.

[40] Charles M. Macal and Michael J. North. Tutorial on agent-based modeling and simulation. In *Proceedings of the 37th conference on Winter simulation*, WSC 05, pages 2 15. Winter Simulation Conference, 2005.

[41] Charles M. Macal and Michael J. North. Tutorial on agent-based modeling and simulation part 2: how to model with agents. In *Proceedings of the 38th conference on Winter simulation*, WSC 06, pages 73 83. Winter Simulation Conference, 2006.

[42] Charles M. Macal and Michael J. North. Agent-based modeling and simulation: Abms examples. In *Proceedings of the 40th Conference on Winter Simulation*, WSC 08, pages 101 112. Winter Simulation Conference, 2008.

[43] C.M. Macal and M.J. North. Agent-based modeling and simulation. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 86 98, 2009.

[44] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281 297, 1967.

[45] Peter F. Major. Predator-prey interactions in two schooling fishes, caranx ignobilis and stolephorus purpureus. *Animal Behaviour*, 26, Part 3:760 777, 1978.

[46] Youssef M. Marzouk and Ahmed F. Ghoniem. K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical n-body simulations. *J. Comput. Phys.*, 207(2):493 528, aug 2005.

[47] Johan E. Mebius. Derivation of the euler-rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations. 2007.

[48] Fehmina Merchant. *Load balancing in spatial individual-based systems using autonomous objects*. PhD thesis, University of California, 1998. AAI9903278.

[49] Jayadev Misra. Distributed discrete-event simulation. *ACM Comput. Surv.*, 18(1):39 65, mar 1986.

[50] D. Newth and J. Finnigan. Emergence and self-organization in chemistry and biology. *Australian Journal of Chemistry*, 59:841 848, 2006.

[51] David M. Nicol. Rectilinear partitioning of irregular data parallel computations. *J. Parallel Distrib. Comput.*, 23(2):119 134, November 1994.

[52] David M. Nicol and Joel H. Saltz. An analysis of scatter decomposition. *IEEE Trans. Comput.*, 39(11):1337 1345, November 1990.

[53] Roberto Uribe Paredes, Claudio Marquez, and Roberto Solar. Construction strategies on metric structures for similarity search. *CLEI Electron. J.*, 12(3), 2009.

[54] Julia K. Parrish, Steven V. Viscido, and Daniel Grunbaum. Self-organized fish schools: An examination of emergent properties. *Biol. Bull*, 202:296 305, 2002.

[55] B. L. Partridge. The structure and function of fish schools. *Scientic American*, 246:90 99, 1982.

[56] G.V.N. Powell. Experimental analysis of the social value of flocking by starlings (sturnus vulgaris) in relation to predation and foraging. *Animal Behaviour*, 22(2):501 505, 1974.

[57] R. H. Pulliam. On the advantages of flocking. *J. Theor. Biol.*, 38:419 422, 1973.

[58] Michael J. Quinn, Ronald A. Metoyer, and Katharine Hunter-zaworski. Parallel implementation of the social forces model. In *in Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, pages 63 74, 2003.

[59] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21:25 34, 1987.

[60] Craig W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference*, pages 763 782, 1999.

[61] E. Saule, E.O. Bas, and U.V. Ç andatalyu andrek. Partitioning spatially located computations using rectangles. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 709 720, may 2011.

[62] J. G. Skellam. Random dispersal in theoretical populations. *Biometrika*, 38:196 218, 1951.

[63] Roberto Solar, Remo Suppi, and Emilio Luque. High performance individual-oriented simulation using complex models. *Procedia Computer Science*, 1(1):447 456, 2010.

[64] Roberto Solar, Remo Suppi, and Emilio Luque. High performance distributed cluster-based individual-oriented fish school simulation. *Procedia Computer Science*, 4:76 85, 2011.

[65] Roberto Solar, Remo Suppi, and Emilio Luque. Proximity load balancing for distributed cluster-based individual-oriented fish school simulations. *Procedia Computer Science*, 2012.

[66] Annette F. Taylor and Mark R. Tinsley. Chemical self-organization: A path to patterns. *Nature Chemistry*, 1(5):340 341, August 2009.

[67] G. F. Turner and T. J. Pitcher. Attack abatement: a model for group protection by combined avoidance and dilution. *Am. Nat.*, 128:228 240, 1986.

[68] Tamas Vicsek, Andras Czirok, Eshel Ben-Jacob, Inon Cohen, and Ofer Sochet. Novel type of phase transition in a system of self-driven particles. *PHYS REV LETT.*, 75:1226, 1995.

[69] G. Vigueras, M. Lozano, J. M. Orduna, and F. Grimaldo. Improving the performance of partitioning methods for crowd simulations. In *Proceedings of the 2008 8th International Conference on Hybrid Intelligent Systems*, HIS 08, pages 102 107, Washington, DC, USA, 2008. IEEE Computer Society.

[70] G. Vigueras, M. Lozano, J. M. Orduna, and F. Grimaldo. A comparative study of partitioning methods for crowd simulations. *Appl. Soft Comput.*, 10(1):225 235, January 2010.

[71] Guillermo Vigueras, Miguel Lozano, and Juan M. Orduna. Workload balancing in distributed crowd simulations: the partitioning method. *J. Supercomput.*, 58(2):261 269, nov 2011.

[72] V. Volterra. Variazioni e fluttuazioni del numero d individui in specie animali conviventi. *Mem. R. Accad. Naz. dei Lincei*, 2:31 113, 1926.

[73] Ludwig von Bertalanffy. *General System Theory: Foundations, Development, Applications (Revised Edition)*. George Braziller, Inc., revised edition, 1969.

[74] Yongwei Wang, M. Lees, Wentong Cai, Suiping Zhou, and M.Y.H. Low. Cluster based partitioning for agent-based crowd simulations. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 1047 1058, dec 2009.

[75] M. H. Willebeek-LeMair and A. P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans. Parallel Distrib. Syst.*, 4(9):979 993, 1993.

[76] Dongliang Zhang, Changjun Jiang, and Shu Li. A fast adaptive load balancing method for parallel particle-based simulations. *Simulation Modelling Practice and Theory*, 17(6):1032 1042, 2009.

[77] Bo Zhou and Suiping Zhou. Parallel simulation of group behaviors. In *Proceedings of the 36th conference on Winter simulation*, WSC 04, pages 364 370. Winter Simulation Conference, 2004.