

A Framework for Small Distributed Projects

And a Protein Clusterer Application

Guillermo Biset

DEPARTAMENTO DE COMPUTACIÓN
UNIVERSIDAD NACIONAL DE RÍO CUARTO



Creer... Crear... Crecer...



Temario:

- 1 Motivación y Problema
 - FuD
 - El clusterer
- 2 Marco Teórico
 - Programación en Paralelo
 - Clustering
- 3 FuD
 - Diseño
 - Implementación
- 4 El Clusterer
 - Problema
 - RMSD: disimilitud
 - Algoritmo
- 5 Conclusión y Trabajo a Futuro

Motivación

El uso de recursos computacionales es transversal a la mayoría de las ciencias aplicadas.

Muchos de estos usos presentan requerimientos de procesamiento de un conjunto considerable de datos.

Un buen enfoque a la hora de procesar eficientemente un gran conjunto de datos es mediante técnicas computacionales de paralelización.

El **problema**, sin embargo, es que los científicos especialistas en los problemas a resolver no necesariamente son especialistas en programación paralela.

Motivación

El uso de recursos computacionales es transversal a la mayoría de las ciencias aplicadas.

Muchos de estos usos presentan requerimientos de procesamiento de un conjunto considerable de datos.

Un buen enfoque a la hora de procesar eficientemente un gran conjunto de datos es mediante técnicas computacionales de paralelización.

El **problema**, sin embargo, es que los científicos especialistas en los problemas a resolver no necesariamente son especialistas en programación paralela.

Motivación

El uso de recursos computacionales es transversal a la mayoría de las ciencias aplicadas.

Muchos de estos usos presentan requerimientos de procesamiento de un conjunto considerable de datos.

Un buen enfoque a la hora de procesar eficientemente un gran conjunto de datos es mediante técnicas computacionales de paralelización.

El **problema**, sin embargo, es que los científicos especialistas en los problemas a resolver no necesariamente son especialistas en programación paralela.

Motivación

El uso de recursos computacionales es transversal a la mayoría de las ciencias aplicadas.

Muchos de estos usos presentan requerimientos de procesamiento de un conjunto considerable de datos.

Un buen enfoque a la hora de procesar eficientemente un gran conjunto de datos es mediante técnicas computacionales de paralelización.

El **problema**, sin embargo, es que los científicos especialistas en los problemas a resolver no necesariamente son especialistas en programación paralela.

Problema

Se desea, entonces, desarrollar un marco de aplicación (o framework) que permita, a científicos no especialistas en técnicas de paralelización, implementar soluciones informáticas para sus problemas con las siguientes características:

- No debe ser necesario poseer conocimientos avanzados de programación ni concurrencia.
- Se deben aprovechar los beneficios de la programación paralela mediante el uso del framework.
- El uso del framework no debe imponer serias desventajas de performance.
- El framework debe aprovechar diferentes disposiciones de los recursos computacionales.

Problema

Se desea, entonces, desarrollar un marco de aplicación (o framework) que permita, a científicos no especialistas en técnicas de paralelización, implementar soluciones informáticas para sus problemas con las siguientes características:

- No debe ser necesario poseer conocimientos avanzados de programación ni concurrencia.
- Se deben aprovechar los beneficios de la programación paralela mediante el uso del framework.
- El uso del framework no debe imponer serias desventajas de performance.
- El framework debe aprovechar diferentes disposiciones de los recursos computacionales.

Problema

Se desea, entonces, desarrollar un marco de aplicación (o framework) que permita, a científicos no especialistas en técnicas de paralelización, implementar soluciones informáticas para sus problemas con las siguientes características:

- No debe ser necesario poseer conocimientos avanzados de programación ni concurrencia.
- Se deben aprovechar los beneficios de la programación paralela mediante el uso del framework.
- El uso del framework no debe imponer serias desventajas de performance.
- El framework debe aprovechar diferentes disposiciones de los recursos computacionales.

Problema

Se desea, entonces, desarrollar un marco de aplicación (o framework) que permita, a científicos no especialistas en técnicas de paralelización, implementar soluciones informáticas para sus problemas con las siguientes características:

- No debe ser necesario poseer conocimientos avanzados de programación ni concurrencia.
- Se deben aprovechar los beneficios de la programación paralela mediante el uso del framework.
- El uso del framework no debe imponer serias desventajas de performance.
- El framework debe aprovechar diferentes disposiciones de los recursos computacionales.

Problema

Se desea, entonces, desarrollar un marco de aplicación (o framework) que permita, a científicos no especialistas en técnicas de paralelización, implementar soluciones informáticas para sus problemas con las siguientes características:

- No debe ser necesario poseer conocimientos avanzados de programación ni concurrencia.
- Se deben aprovechar los beneficios de la programación paralela mediante el uso del framework.
- El uso del framework no debe imponer serias desventajas de performance.
- El framework debe aprovechar diferentes disposiciones de los recursos computacionales.

Enfoques Similares

Existen varios enfoques que permiten, parcialmente, tratar estos problemas:

Enfoques notables

BOINC : Es un middleware no comercial diseñado en Berkeley para computación voluntaria y grid[Anderson, 2004]

MPI : Es una interfaz que permite que muchas computadoras se comuniquen entre sí principalmente utilizada en clusters de alta performance[Snir et al., 1996]

MapReduce : Es un modelo de programación desarrollado en Google para permitir la implementación de algoritmos distribuidos[Dean and Ghemawat, 2004].

Enfoques Similares

Existen varios enfoques que permiten, parcialmente, tratar estos problemas:

Enfoques notables

BOINC : Es un middleware no comercial diseñado en Berkeley para computación voluntaria y grid[Anderson, 2004]

MPI : Es una interfaz que permite que muchas computadoras se comuniquen entre sí principalmente utilizada en clusters de alta performance[Snir et al., 1996]

MapReduce : Es un modelo de programación desarrollado en Google para permitir la implementación de algoritmos distribuidos[Dean and Ghemawat, 2004].

Enfoques Similares

Existen varios enfoques que permiten, parcialmente, tratar estos problemas:

Enfoques notables

BOINC : Es un middleware no comercial diseñado en Berkeley para computación voluntaria y grid[Anderson, 2004]

MPI : Es una interfaz que permite que muchas computadoras se comuniquen entre sí principalmente utilizada en clusters de alta performance[Snir et al., 1996]

MapReduce : Es un modelo de programación desarrollado en Google para permitir la implementación de algoritmos distribuidos[Dean and Ghemawat, 2004].

El clusterer de proteínas

En **FuDePAN**, un proyecto (Galadriel) se encarga de generar estructuras de proteínas automáticamente, sujetas a viabilidad tridimensional.

Pero la mayoría de estas estructuras generadas presentan similitudes geométricas al punto de evidenciar comportamiento idéntico ante aplicaciones medicinales.

Se debe, entonces, tomar datos generados por Galadriel y realizar agrupaciones de estas estructuras basados en sus similitudes geométricas, para permitir a los investigadores concentrarse solamente en alguna de estas proteínas.

El clusterer de proteínas

En **FuDePAN**, un proyecto (Galadriel) se encarga de generar estructuras de proteínas automáticamente, sujetas a viabilidad tridimensional.

Pero la mayoría de estas estructuras generadas presentan similitudes geométricas al punto de evidenciar comportamiento idéntico ante aplicaciones medicinales.

Se debe, entonces, tomar datos generados por Galadriel y realizar agrupaciones de estas estructuras basados en sus similitudes geométricas, para permitir a los investigadores concentrarse solamente en alguna de estas proteínas.

El clusterer de proteínas

En **FuDePAN**, un proyecto (Galadriel) se encarga de generar estructuras de proteínas automáticamente, sujetas a viabilidad tridimensional.

Pero la mayoría de estas estructuras generadas presentan similitudes geométricas al punto de evidenciar comportamiento idéntico ante aplicaciones medicinales.

Se debe, entonces, tomar datos generados por Galadriel y realizar agrupaciones de estas estructuras basados en sus similitudes geométricas, para permitir a los investigadores concentrarse solamente en alguna de estas proteínas.

Programación en Paralelo

La programación en paralelo se refiere a algoritmos y programas que se ejecutan en computadoras paralelas, aunque hay varias presentaciones diferentes que constituyen a una computadora paralela, una definición aceptada es la siguiente[Foster, 1995]:

Una computadora paralela es un conjunto de procesadores que pueden trabajar de manera cooperativa para resolver un problema computacional.

Esta definición es lo suficientemente amplia para incluir supercomputadoras con cientos o miles de procesadores, redes amplias de computadoras o clusters centralizados de ellas.

Programación en Paralelo

La programación en paralelo se refiere a algoritmos y programas que se ejecutan en computadoras paralelas, aunque hay varias presentaciones diferentes que constituyen a una computadora paralela, una definición aceptada es la siguiente[Foster, 1995]:

Una computadora paralela es un conjunto de procesadores que pueden trabajar de manera cooperativa para resolver un problema computacional.

Esta definición es lo suficientemente amplia para incluir supercomputadoras con cientos o miles de procesadores, redes amplias de computadoras o clusters centralizados de ellas.

Programación en Paralelo

La programación en paralelo se refiere a algoritmos y programas que se ejecutan en computadoras paralelas, aunque hay varias presentaciones diferentes que constituyen a una computadora paralela, una definición aceptada es la siguiente[Foster, 1995]:

Una computadora paralela es un conjunto de procesadores que pueden trabajar de manera cooperativa para resolver un problema computacional.

Esta definición es lo suficientemente amplia para incluir supercomputadoras con cientos o miles de procesadores, redes amplias de computadoras o clusters centralizados de ellas.

La taxonomía de Flynn

Flynn[Flynn, 1972] propuso una taxonomía de arquitecturas de computadoras basado en la cantidad de instrucciones que se pueden ejecutar concurrentemente y la cantidad de flujos de datos disponibles distinguiendo los siguientes casos:

Single Instruction, Single Data stream (o SISD): Una computadora seceuncial que no explota el paralelismo en cuantos a instrucciones o datos.

Single Instruction, Multiple Data streams (o SIMD): Una computadora que utiliza manejo paralelo de datos sobre un único nodo de procesamiento.

Multiple Instruction, Single Data stream (o MISD): Referido a múltiples nodos de procesamiento que utilizan un único repositorio de datos.

Multiple Instruction, Multiple Data streams (o MIMD): Cuando múltiples procesadores hacen uso de datos ubicados en de manera distribuída.

La taxonomía de Flynn

Flynn[Flynn, 1972] propuso una taxonomía de arquitecturas de computadoras basado en la cantidad de instrucciones que se pueden ejecutar concurrentemente y la cantidad de flujos de datos disponibles distinguiendo los siguientes casos:

Single Instruction, Single Data stream (o SISD): Una computadora seceuncial que no explota el paralelismo en cuantos a instrucciones o datos.

Single Instruction, Multiple Data streams (o SIMD): Una computadora que utiliza manejo paralelo de datos sobre un único nodo de procesamiento.

Multiple Instruction, Single Data stream (o MISD): Referido a múltiples nodos de procesamiento que utilizan un único repositorio de datos.

Multiple Instruction, Multiple Data streams (o MIMD): Cuando múltiples procesadores hacen uso de datos ubicados en de manera distribuída.

La taxonomía de Flynn

Flynn[Flynn, 1972] propuso una taxonomía de arquitecturas de computadoras basado en la cantidad de instrucciones que se pueden ejecutar concurrentemente y la cantidad de flujos de datos disponibles distinguiendo los siguientes casos:

Single Instruction, Single Data stream (o SISD): Una computadora seceuncial que no explota el paralelismo en cuantos a instrucciones o datos.

Single Instruction, Multiple Data streams (o SIMD): Una computadora que utiliza manejo paralelo de datos sobre un único nodo de procesamiento.

Multiple Instruction, Single Data stream (o MISD): Referido a múltiples nodos de procesamiento que utilizan un único repositorio de datos.

Multiple Instruction, Multiple Data streams (o MIMD): Cuando múltiples procesadores hacen uso de datos ubicados en de manera distribuída.

La taxonomía de Flynn

Flynn[Flynn, 1972] propuso una taxonomía de arquitecturas de computadoras basado en la cantidad de instrucciones que se pueden ejecutar concurrentemente y la cantidad de flujos de datos disponibles distinguiendo los siguientes casos:

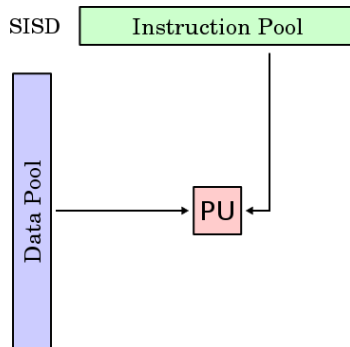
Single Instruction, Single Data stream (o SISD): Una computadora seceuncial que no explota el paralelismo en cuantos a instrucciones o datos.

Single Instruction, Multiple Data streams (o SIMD): Una computadora que utiliza manejo paralelo de datos sobre un único nodo de procesamiento.

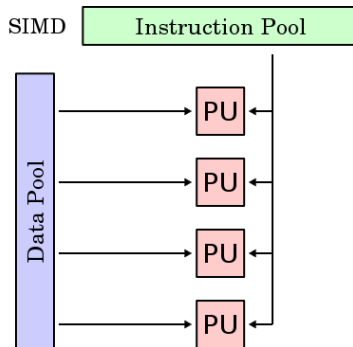
Multiple Instruction, Single Data stream (o MISD): Referido a múltiples nodos de procesamiento que utilizan un único repositorio de datos.

Multiple Instruction, Multiple Data streams (o MIMD): Cuando múltiples procesadores hacen uso de datos ubicados en de manera distribuída.

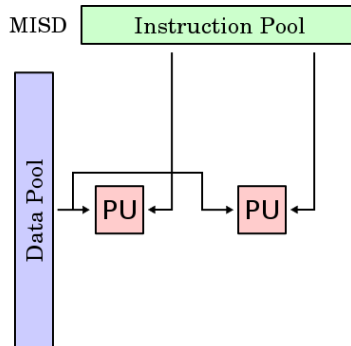
SISD



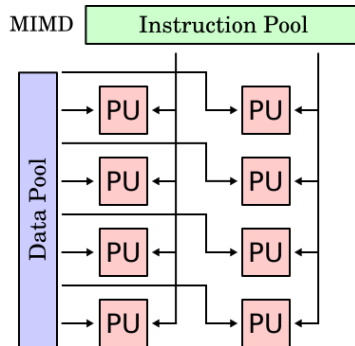
SIMD



MISD



MIMD



Clustering

El clustering de datos se realiza con el propósito de agrupar objetos bajo ciertas propiedades que estos evidencian. Una definición aceptada, tomada de [Gan et al., 2007] es la siguiente:

Data Clustering

El clustering de datos (o clustering) es un método para crear grupos de objetos (o clusters) de tal modo que los objetos en un cluster son muy similares entre sí y objetos en grupos diferentes son bastante diferentes.

De manera más formal:

El clustering de datos se realiza sobre un determinado conjunto de datos utilizando una noción de similitud dada mediante la agrupación de los datos de manera que los elementos en un mismo grupo son similares para esta noción y elementos de diferentes grupos son diferentes.

Clustering

El clustering de datos se realiza con el propósito de agrupar objetos bajo ciertas propiedades que estos evidencian. Una definición aceptada, tomada de [Gan et al., 2007] es la siguiente:

Data Clustering

El clustering de datos (o clustering) es un método para crear grupos de objetos (o clusters) de tal modo que los objetos en un cluster son muy similares entre sí y objetos en grupos diferentes son bastante diferentes.

De manera más formal:

El clustering de datos se realiza sobre un determinado conjunto de datos utilizando una noción de similitud dada mediante la agrupación de los datos de manera que los elementos en un mismo grupo son similares para esta noción y elementos de diferentes grupos son diferentes.

Clustering

El clustering de datos se realiza con el propósito de agrupar objetos bajo ciertas propiedades que estos evidencian. Una definición aceptada, tomada de [Gan et al., 2007] es la siguiente:

Data Clustering

El clustering de datos (o clustering) es un método para crear grupos de objetos (o clusters) de tal modo que los objetos en un cluster son muy similares entre sí y objetos en grupos diferentes son bastante diferentes.

De manera más formal:

El clustering de datos se realiza sobre un determinado conjunto de datos utilizando una noción de similitud dada mediante la agrupación de los datos de manera que los elementos en un mismo grupo son similares para esta noción y elementos de diferentes grupos son diferentes.

Algoritmo *k*-means

```
Require: Data set D, Number of clusters k, Dimensions d:  
  {Ci is the ith cluster}  
  {1. Initialization Phase}  
1 : (C1,C2,...,Ck) := Initial partition of D.  
  {2. Iteration Phase}  
2 : repeat  
3 :   dij := distance between case i and cluster j;  
4 :   ni := arg min 1<=j<=k dij;  
5 :   Assign case i to cluster ni;  
6 :   Recompute the cluster means of any changed clusters above;  
7 : until no further changes of cluster membership occur  
8 : Output Results
```

The conventional *k*-means algorithm.

Proteínas

Las proteínas son polímeros de aminoácidos unidos entre sí de forma lineal mediante uniones péptidicas.

Una proteína posee tres tipos de estructuras básicas:

Primaria : La secuencia de aminoácidos.

Secundaria : Se refiere al plegamiento de estructuras locales que se repiten regularmente, estabilizadas por uniones del tipo hidrógeno.

Terciaria : Refiere a la forma general de una molécula individual y las relaciones espaciales que las estructuras secundarias tienen entre sí.

Proteínas

Las proteínas son polímeros de aminoácidos unidos entre sí de forma lineal mediante uniones péptidicas.

Una proteína posee tres tipos de estructuras básicas:

Primaria : La secuencia de aminoácidos.

Secundaria : Se refiere al plegamiento de estructuras locales que se repiten regularmente, estabilizadas por uniones del tipo hidrógeno.

Terciaria : Refiere a la forma general de una molécula individual y las relaciones espaciales que las estructuras secundarias tienen entre sí.

Proteínas

Las proteínas son polímeros de aminoácidos unidos entre sí de forma lineal mediante uniones péptidicas.

Una proteína posee tres tipos de estructuras básicas:

Primaria : La secuencia de aminoácidos.

Secundaria : Se refiere al plegamiento de estructuras locales que se repiten regularmente, estabilizadas por uniones del tipo hidrógeno.

Terciaria : Refiere a la forma general de una molécula individual y las relaciones espaciales que las estructuras secundarias tienen entre sí.

Proteínas

Las proteínas son polímeros de aminoácidos unidos entre sí de forma lineal mediante uniones péptidicas.

Una proteína posee tres tipos de estructuras básicas:

Primaria : La secuencia de aminoácidos.

Secundaria : Se refiere al plegamiento de estructuras locales que se repiten regularmente, estabilizadas por uniones del tipo hidrógeno.

Terciaria : Refiere a la forma general de una molécula individual y las relaciones espaciales que las estructuras secundarias tienen entre sí.

C++

C++ es un lenguaje de programación de tipado estático, multi paradigma, de propósito general y compilado. Se lo reconoce como un lenguaje de nivel-medio, ya que es provee una combinación de capacidades de lenguajes de bajo y alto nivel.

```
//exception if _ids_to_job_map[id] isn't defined
mili::find(_ids_to_job_map,id)->process_results(id, message);

//remove from pending list
std::list<JobUnit*>::iterator it;
it = find_if(_pendingList.begin(),_pendingList.end(),
            boost::bind(&JobUnit::get_id, _1) == id);

if (it != _pendingList.end())
{
    delete *it;
    _pendingList.erase(it);
}
else
    syslog(LOG_NOTICE,"Finished JobUnit %u was not pending.",id);
```

Razón del framework

El framework (**FuD**) se desarrolló para contemplar el problema expresado previamente: permitir a científicos no especialistas en ciencias de la computación implementar de manera sencilla soluciones informáticas que hagan uso de técnicas distribuidas de programación. Esto se puede resumir en el siguiente objetivo:

Diseño y desarrolle un framework para automatizar la distribución de aplicaciones computacionales a través de disposiciones heterogéneas y dinámicas de nodos de procesamiento.

Además, el diseño debería:

- Ser lo suficientemente flexible para permitir desarrollar cualquier tipo de problema computacional.
- Esta implementación debe ser independiente de los recursos que se dispondrán eventualmente par llevar a cabo el cómputo.

Razón del framework

El framework (**FuD**) se desarrolló para contemplar el problema expresado previamente: permitir a científicos no especialistas en ciencias de la computación implementar de manera sencilla soluciones informáticas que hagan uso de técnicas distribuidas de programación. Esto se puede resumir en el siguiente objetivo:

Diseñe y desarrolle un framework para automatizar la distribución de aplicaciones computacionales a través de disposiciones heterogéneas y dinámicas de nodos de procesamiento.

Además, el diseño debería:

- Ser lo suficientemente flexible para permitir desarrollar cualquier tipo de problema computacional.
- Esta implementación debe ser independiente de los recursos que se dispondrán eventualmente para llevar a cabo el cómputo.

Razón del framework

El framework (**FuD**) se desarrolló para contemplar el problema expresado previamente: permitir a científicos no especialistas en ciencias de la computación implementar de manera sencilla soluciones informáticas que hagan uso de técnicas distribuidas de programación. Esto se puede resumir en el siguiente objetivo:

Diseñe y desarrolle un framework para automatizar la distribución de aplicaciones computacionales a través de disposiciones heterogéneas y dinámicas de nodos de procesamiento.

Además, el diseño debería:

- Ser lo suficientemente flexible para permitir desarrollar cualquier tipo de problema computacional.
- Esta implementación debe ser independiente de los recursos que se dispondrán eventualmente par llevar a cabo el cómputo.

Razón del framework

El framework (**FuD**) se desarrolló para contemplar el problema expresado previamente: permitir a científicos no especialistas en ciencias de la computación implementar de manera sencilla soluciones informáticas que hagan uso de técnicas distribuidas de programación. Esto se puede resumir en el siguiente objetivo:

Diseñe y desarrolle un framework para automatizar la distribución de aplicaciones computacionales a través de disposiciones heterogéneas y dinámicas de nodos de procesamiento.

Además, el diseño debería:

- Ser lo suficientemente flexible para permitir desarrollar cualquier tipo de problema computacional.
- Esta implementación debe ser independiente de los recursos que se dispondrán eventualmente para llevar a cabo el cómputo.

Resumiendo

En resumen, **FuD** debe permitir ser utilizado para resolver cualquier problema que se puede solucionar secuencialmente. Además, debe permitir esto sin requerir que el implementador:

- Sepa conceptos de programación paralela.
- Quede restringido a una determinada disposición de los nodos procesadores.
- Quede limitado por un particular sistema de comunicación.

Finalmente, **FuD** asegura:

- Que la aplicación implementada tiene el potencial de ejecutarse paralelamente.
- Que la aplicación explotará los recursos que se pongan a su disposición.
- El uso del framework no generará pérdidas de performance considerables.
- Los datos intercambiados entre aplicaciones cliente y servidor no llevarán una carga adicional significativa sobre los datos necesarios para la aplicación.

Resumiendo

En resumen, **FuD** debe permitir ser utilizado para resolver cualquier problema que se puede solucionar secuencialmente. Además, debe permitir esto sin requerir que el implementador:

- Sepa conceptos de programación paralela.
- Quede restringido a una determinada disposición de los nodos procesadores.
- Quede limitado por un particular sistema de comunicación.

Finalmente, **FuD** asegura:

- Que la aplicación implementada tiene el potencial de ejecutarse paralelamente.
- Que la aplicación explotará los recursos que se pongan a su disposición.
- El uso del framework no generará pérdidas de performance considerables.
- Los datos intercambiados entre aplicaciones cliente y servidor no llevarán una carga adicional significativa sobre los datos necesarios para la aplicación.

Resumiendo

En resumen, **FuD** debe permitir ser utilizado para resolver cualquier problema que se puede solucionar secuencialmente. Además, debe permitir esto sin requerir que el implementador:

- Sepa conceptos de programación paralela.
- Quede restringido a una determinada disposición de los nodos procesadores.
- Quede limitado por un particular sistema de comunicación.

Finalmente, **FuD** asegura:

- Que la aplicación implementada tiene el potencial de ejecutarse paralelamente.
- Que la aplicación explotará los recursos que se pongan a su disposición.
- El uso del framework no generará pérdidas de performance considerables.
- Los datos intercambiados entre aplicaciones cliente y servidor no llevarán una carga adicional significativa sobre los datos necesarios para la aplicación.

Resumiendo

En resumen, **FuD** debe permitir ser utilizado para resolver cualquier problema que se puede solucionar secuencialmente. Además, debe permitir esto sin requerir que el implementador:

- Sepa conceptos de programación paralela.
- Quede restringido a una determinada disposición de los nodos procesadores.
- Quede limitado por un particular sistema de comunicación.

Finalmente, **FuD** asegura:

- Que la aplicación implementada tiene el potencial de ejecutarse paralelamente.
- Que la aplicación explotará los recursos que se pongan a su disposición.
- El uso del framework no generará pérdidas de performance considerables.
- Los datos intercambiados entre aplicaciones cliente y servidor no llevarán una carga adicional significativa sobre los datos necesarios para la aplicación.

Resumiendo

En resumen, **FuD** debe permitir ser utilizado para resolver cualquier problema que se puede solucionar secuencialmente. Además, debe permitir esto sin requerir que el implementador:

- Sepa conceptos de programación paralela.
- Quede restringido a una determinada disposición de los nodos procesadores.
- Quede limitado por un particular sistema de comunicación.

Finalmente, **FuD** asegura:

- Que la aplicación implementada tiene el potencial de ejecutarse paralelamente.
- Que la aplicación explotará los recursos que se pongan a su disposición.
- El uso del framework no generará pérdidas de performance considerables.
- Los datos intercambiados entre aplicaciones cliente y servidor no llevarán una carga adicional significativa sobre los datos necesarios para la aplicación.

Resumiendo

En resumen, **FuD** debe permitir ser utilizado para resolver cualquier problema que se puede solucionar secuencialmente. Además, debe permitir esto sin requerir que el implementador:

- Sepa conceptos de programación paralela.
- Quede restringido a una determinada disposición de los nodos procesadores.
- Quede limitado por un particular sistema de comunicación.

Finalmente, **FuD** asegura:

- Que la aplicación implementada tiene el potencial de ejecutarse paralelamente.
- Que la aplicación explotará los recursos que se pongan a su disposición.
- El uso del framework no generará pérdidas de performance considerables.
- Los datos intercambiados entre aplicaciones cliente y servidor no llevarán una carga adicional significativa sobre los datos necesarios para la aplicación.

Resumiendo

En resumen, **FuD** debe permitir ser utilizado para resolver cualquier problema que se puede solucionar secuencialmente. Además, debe permitir esto sin requerir que el implementador:

- Sepa conceptos de programación paralela.
- Quede restringido a una determinada disposición de los nodos procesadores.
- Quede limitado por un particular sistema de comunicación.

Finalmente, **FuD** asegura:

- Que la aplicación implementada tiene el potencial de ejecutarse paralelamente.
- Que la aplicación explotará los recursos que se pongan a su disposición.
- El uso del framework no generará pérdidas de performance considerables.
- Los datos intercambiados entre aplicaciones cliente y servidor no llevarán una carga adicional significativa sobre los datos necesarios para la aplicación.

Enfoque

Para lograr esto, se debe implementar un esquema *Divide&Conquer*, donde el procesamiento será llevado a cabo por los nodos procesadores (o clientes):

Divide&Conquer + Process

Divide : Cuando se realiza la división de un trabajo en una unidad de trabajo. Esto debe ser un proceso *transaccional* y está implementado por el método `produce_next_job_unit(size)`.

Conquer : Se lleva a cabo al incorporar los resultados de una unidad de trabajo, mediante `handle_results(id, input)`.

Process : Las unidades de trabajo son comunicadas a un cliente, este las procesa y comunica los resultados para ser incorporados.

Enfoque

Para lograr esto, se debe implementar un esquema *Divide&Conquer*, donde el procesamiento será llevado a cabo por los nodos procesadores (o clientes):

Divide&Conquer + Process

Divide : Cuando se realiza la división de un trabajo en una unidad de trabajo. Esto debe ser un proceso *transaccional* y está implementado por el método `produce_next_job_unit(size)`.

Conquer : Se lleva a cabo al incorporar los resultados de una unidad de trabajo, mediante `handle_results(id, input)`.

Process : Las unidades de trabajo son comunicadas a un cliente, este las procesa y comunica los resultados para ser incorporados.

Enfoque

Para lograr esto, se debe implementar un esquema *Divide&Conquer*, donde el procesamiento será llevado a cabo por los nodos procesadores (o clientes):

Divide&Conquer + Process

Divide : Cuando se realiza la división de un trabajo en una unidad de trabajo. Esto debe ser un proceso *transaccional* y está implementado por el método `produce_next_job_unit(size)`.

Conquer : Se lleva a cabo al incorporar los resultados de una unidad de trabajo, mediante `handle_results(id, input)`.

Process : Las unidades de trabajo son comunicadas a un cliente, este las procesa y comunica los resultados para ser incorporados.

Enfoque

Para lograr esto, se debe implementar un esquema *Divide&Conquer*, donde el procesamiento será llevado a cabo por los nodos procesadores (o clientes):

Divide&Conquer + Process

Divide : Cuando se realiza la división de un trabajo en una unidad de trabajo. Esto debe ser un proceso *transaccional* y está implementado por el método `produce_next_job_unit(size)`.

Conquer : Se lleva a cabo al incorporar los resultados de una unidad de trabajo, mediante `handle_results(id, input)`.

Process : Las unidades de trabajo son comunicadas a un cliente, este las procesa y comunica los resultados para ser incorporados.

Principios Utilizados

SRP : (o Single Responsibility Principle) No debería haber más de una razón de cambio para una clase.

DIP : (o Dependency Inversion Principle) Argumenta:

- Módulos de alto nivel no deberían depender de módulos inferiores, ambos deberían depender de abstracciones.
- Las abstracciones no deberían depender de detalles, sino que los detalles deberían depender de las abstracciones.

ISP : (o Interface Segregation Principle) Los clientes no deberían depender de interfases que no utilizarán.

LSV : (o Liskov Substitution Principle) Funciones que usan punteros o referencias a clases base deben poder utilizar objetos de subclases de la original sin saberlo.

Principios Utilizados

SRP : (o Single Responsibility Principle) No debería haber más de una razón de cambio para una clase.

DIP : (o Dependency Inversion Principle) Argumenta:

- Módulos de alto nivel no deberían depender de módulos inferiores, ambos deberían depender de abstracciones.
- Las abstracciones no deberían depender de detalles, sino que los detalles deberían depender de las abstracciones.

ISP : (o Interface Segregation Principle) Los clientes no deberían depender de interfases que no utilizarán.

LSV : (o Liskov Substitution Principle) Funciones que usan punteros o referencias a clases base deben poder utilizar objetos de subclases de la original sin saberlo.

Principios Utilizados

SRP : (o Single Responsibility Principle) No debería haber más de una razón de cambio para una clase.

DIP : (o Dependency Inversion Principle) Argumenta:

- Módulos de alto nivel no deberían depender de módulos inferiores, ambos deberían depender de abstracciones.
- Las abstracciones no deberían depender de detalles, sino que los detalles deberían depender de las abstracciones.

ISP : (o Interface Segregation Principle) Los clientes no deberían depender de interfases que no utilizarán.

LSV : (o Liskov Substitution Principle) Funciones que usan punteros o referencias a clases base deben poder utilizar objetos de subclases de la original sin saberlo.

Principios Utilizados

SRP : (o Single Responsibility Principle) No debería haber más de una razón de cambio para una clase.

DIP : (o Dependency Inversion Principle) Argumenta:

- Módulos de alto nivel no deberían depender de módulos inferiores, ambos deberían depender de abstracciones.
- Las abstracciones no deberían depender de detalles, sino que los detalles deberían depender de las abstracciones.

ISP : (o Interface Segregation Principle) Los clientes no deberían depender de interfases que no utilizarán.

LSV : (o Liskov Substitution Principle) Funciones que usan punteros o referencias a clases base deben poder utilizar objetos de subclases de la original sin saberlo.

Conceptos centrales a **FuD**

FuD se organiza con un esquema Master-Worker (un servidor y varios clientes conectados al mismo). Los siguientes conceptos sirven para entender el funcionamiento de una aplicación que usa el framework:

Cliente : Un cliente es una aplicación conectada al servidor y es el encargado de realizar las tareas de procesamiento.

Trabajo : Un trabajo es cualquier tarea a realizar por la aplicación (también llamado trabajo distribuible). Los mismos se pueden subdividir en unidades de trabajo.

Unidad de Trabajo : Representa un cómputo concreto y pertenece a un trabajo particular. No se puede dividir.

El manejador de clientes : Este módulo es el encargado de manejar los clientes conectados al servidor.

Manejador de trabajos : Este es el módulo de **FuD** encargado de llevar cuenta de los trabajos (de ambos tipos) que maneja el framework en un momento dado.

Conceptos centrales a **FuD**

FuD se organiza con un esquema Master-Worker (un servidor y varios clientes conectados al mismo). Los siguientes conceptos sirven para entender el funcionamiento de una aplicación que usa el framework:

Cliente : Un cliente es una aplicación conectada al servidor y es el encargado de realizar las tareas de procesamiento.

Trabajo : Un trabajo es cualquier tarea a realizar por la aplicación (también llamado trabajo distribuible). Los mismos se pueden subdividir en unidades de trabajo.

Unidad de Trabajo : Representa un cómputo concreto y pertenece a un trabajo particular. No se puede dividir.

El manejador de clientes : Este módulo es el encargado de manejar los clientes conectados al servidor.

Manejador de trabajos : Este es el módulo de **FuD** encargado de llevar cuenta de los trabajos (de ambos tipos) que maneja el framework en un momento dado.

Conceptos centrales a **FuD**

FuD se organiza con un esquema Master-Worker (un servidor y varios clientes conectados al mismo). Los siguientes conceptos sirven para entender el funcionamiento de una aplicación que usa el framework:

Cliente : Un cliente es una aplicación conectada al servidor y es el encargado de realizar las tareas de procesamiento.

Trabajo : Un trabajo es cualquier tarea a realizar por la aplicación (también llamado trabajo distribuible). Los mismos se pueden subdividir en unidades de trabajo.

Unidad de Trabajo : Representa un cómputo concreto y pertenece a un trabajo particular. No se puede dividir.

El manejador de clientes : Este módulo es el encargado de manejar los clientes conectados al servidor.

Manejador de trabajos : Este es el módulo de **FuD** encargado de llevar cuenta de los trabajos (de ambos tipos) que maneja el framework en un momento dado.

Conceptos centrales a **FuD**

FuD se organiza con un esquema Master-Worker (un servidor y varios clientes conectados al mismo). Los siguientes conceptos sirven para entender el funcionamiento de una aplicación que usa el framework:

Cliente : Un cliente es una aplicación conectada al servidor y es el encargado de realizar las tareas de procesamiento.

Trabajo : Un trabajo es cualquier tarea a realizar por la aplicación (también llamado trabajo distribuible). Los mismos se pueden subdividir en unidades de trabajo.

Unidad de Trabajo : Representa un cómputo concreto y pertenece a un trabajo particular. No se puede dividir.

El manejador de clientes : Este módulo es el encargado de manejar los clientes conectados al servidor.

Manejador de trabajos : Este es el módulo de **FuD** encargado de llevar cuenta de los trabajos (de ambos tipos) que maneja el framework en un momento dado.

Conceptos centrales a **FuD**

FuD se organiza con un esquema Master-Worker (un servidor y varios clientes conectados al mismo). Los siguientes conceptos sirven para entender el funcionamiento de una aplicación que usa el framework:

Cliente : Un cliente es una aplicación conectada al servidor y es el encargado de realizar las tareas de procesamiento.

Trabajo : Un trabajo es cualquier tarea a realizar por la aplicación (también llamado trabajo distribuible). Los mismos se pueden subdividir en unidades de trabajo.

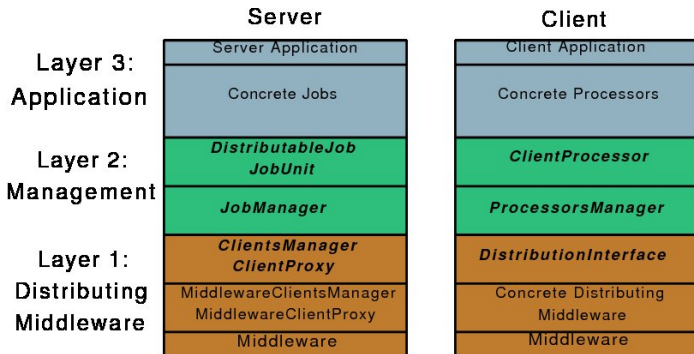
Unidad de Trabajo : Representa un cómputo concreto y pertenece a un trabajo particular. No se puede dividir.

El manejador de clientes : Este módulo es el encargado de manejar los clientes conectados al servidor.

Manejador de trabajos : Este es el módulo de **FuD** encargado de llevar cuenta de los trabajos (de ambos tipos) que maneja el framework en un momento dado.

Capas del framework

FuD se divide en 3 capas bien separadas. La siguiente figura ilustra esta característica:



Capas del framework

Aplicación : Esta capa provee componentes que contienen todos los aspectos del dominio del problema. Como tal, incluye las definiciones y manejo de los datos usados y el algoritmo relevante a la solución implementada.

Manejo de trabajo : Esta capa se encarga de manejar ambos tipos de trabajo, haciendo de nexo entre la aplicación, donde se encuentran los trabajos concretos y se producen las unidades de trabajo, y los clientes, que se manejan en la capa inferior.

Comunicación : En esta capa existe el único vínculo real entre clientes y servidor, la responsabilidad de esta capa es manejar los clientes conectados al servidor y llevar acabo los procedimientos de comunicación entre ambas aplicaciones.

Capas del framework

Aplicación : Esta capa provee componentes que contienen todos los aspectos del dominio del problema. Como tal, incluye las definiciones y manejo de los datos usados y el algoritmo relevante a la solución implementada.

Manejo de trabajo : Esta capa se encarga de manejar ambos tipos de trabajo, haciendo de nexo entre la aplicación, donde se encuentran los trabajos concretos y se producen las unidades de trabajo, y los clientes, que se manejan en la capa inferior.

Comunicación : En esta capa existe el único vínculo real entre clientes y servidor, la responsabilidad de esta capa es manejar los clientes conectados al servidor y llevar acabo los procedimientos de comunicación entre ambas aplicaciones.

Capas del framework

Aplicación : Esta capa provee componentes que contienen todos los aspectos del dominio del problema. Como tal, incluye las definiciones y manejo de los datos usados y el algoritmo relevante a la solución implementada.

Manejo de trabajo : Esta capa se encarga de manejar ambos tipos de trabajo, haciendo de nexo entre la aplicación, donde se encuentran los trabajos concretos y se producen las unidades de trabajo, y los clientes, que se manejan en la capa inferior.

Comunicación : En esta capa existe el único vínculo real entre clientes y servidor, la responsabilidad de esta capa es manejar los clientes conectados al servidor y llevar acabo los procedimientos de comunicación entre ambas aplicaciones.

Diagrama de clases

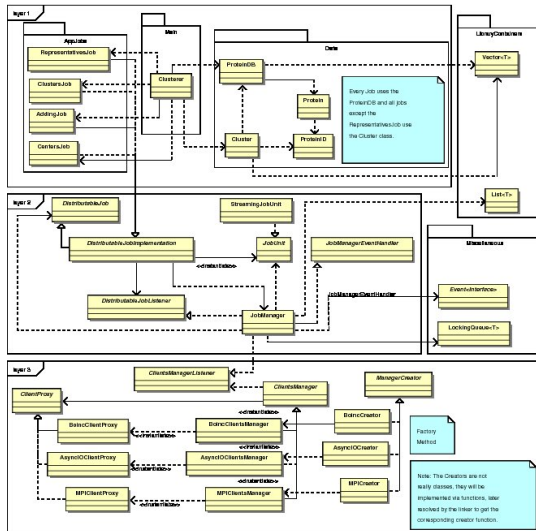
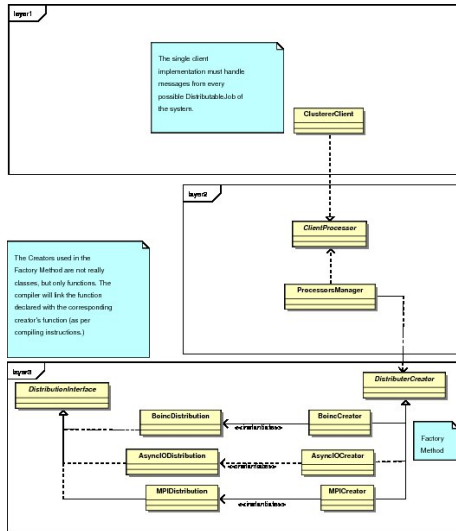


Diagrama de clases (Cliente)



¿Cómo funciona una aplicación que usa **FuD**

La aplicación principal que usa a **FuD** interacciona con el framework de la siguiente manera:

- 1 Crea instancias concretas de trabajos (`DistributableJob`).
- 2 Utiliza los métodos provistos por esta interfaz:
 - `run()` : Ejecuta o inicia el trabajo.
 - `wait_completion()` : Espera (bloqueando) que el trabajo termine.

Además, es la responsabilidad del implementador de la aplicación:

- Definir el nombre de un trabajo (para logging).
- Definir el estado de un trabajo.
- Definir cómo se subdivide un trabajo.
- Definir cómo se incorporan resultados de una unidad de trabajo.

¿Cómo funciona una aplicación que usa **FuD**

La aplicación principal que usa a **FuD** interacciona con el framework de la siguiente manera:

- 1 Crea instancias concretas de trabajos (`DistributableJob`).
- 2 Utiliza los métodos provistos por esta interfaz:

`run()` : Ejecuta o inicia el trabajo.

`wait_completion()` : Espera (bloqueando) que el trabajo termine.

Además, es la responsabilidad del implementador de la aplicación:

- Definir el nombre de un trabajo (para logging).
- Definir el estado de un trabajo.
- Definir cómo se subdivide un trabajo.
- Definir cómo se incorporan resultados de una unidad de trabajo.

¿Cómo funciona una aplicación que usa **FuD**

La aplicación principal que usa a **FuD** interacciona con el framework de la siguiente manera:

- 1 Crea instancias concretas de trabajos (`DistributableJob`).
- 2 Utiliza los métodos provistos por esta interfaz:
 - `run()` : Ejecuta o inicia el trabajo.
 - `wait_completion()` : Espera (bloqueando) que el trabajo termine.

Además, es la responsabilidad del implementador de la aplicación:

- Definir el nombre de un trabajo (para logging).
- Definir el estado de un trabajo.
- Definir cómo se subdivide un trabajo.
- Definir cómo se incorporan resultados de una unidad de trabajo.

¿Cómo funciona una aplicación que usa **FuD**

La aplicación principal que usa a **FuD** interacciona con el framework de la siguiente manera:

- 1 Crea instancias concretas de trabajos (`DistributableJob`).
- 2 Utiliza los métodos provistos por esta interfaz:
 - `run()` : Ejecuta o inicia el trabajo.
 - `wait_completion()` : Espera (bloqueando) que el trabajo termine.

Además, es la responsabilidad del implementador de la aplicación:

- Definir el nombre de un trabajo (para logging).
- Definir el estado de un trabajo.
- Definir cómo se subdivide un trabajo.
- Definir cómo se incorporan resultados de una unidad de trabajo.

¿Cómo funciona una aplicación que usa **FuD**

La aplicación principal que usa a **FuD** interacciona con el framework de la siguiente manera:

- 1 Crea instancias concretas de trabajos (`DistributableJob`).
- 2 Utiliza los métodos provistos por esta interfaz:
 - `run()` : Ejecuta o inicia el trabajo.
 - `wait_completion()` : Espera (bloqueando) que el trabajo termine.

Además, es la responsabilidad del implementador de la aplicación:

- Definir el nombre de un trabajo (para logging).
- Definir el estado de un trabajo.
- Definir cómo se subdivide un trabajo.
- Definir cómo se incorporan resultados de una unidad de trabajo.

¿Cómo funciona una aplicación que usa **FuD**

La aplicación principal que usa a **FuD** interacciona con el framework de la siguiente manera:

- 1 Crea instancias concretas de trabajos (`DistributableJob`).
- 2 Utiliza los métodos provistos por esta interfaz:
 - `run()` : Ejecuta o inicia el trabajo.
 - `wait_completion()` : Espera (bloqueando) que el trabajo termine.

Además, es la responsabilidad del implementador de la aplicación:

- Definir el nombre de un trabajo (para logging).
- Definir el estado de un trabajo.
- Definir cómo se subdivide un trabajo.
- Definir cómo se incorporan resultados de una unidad de trabajo.

¿Cómo funciona una aplicación que usa **FuD**

La aplicación principal que usa a **FuD** interacciona con el framework de la siguiente manera:

- 1 Crea instancias concretas de trabajos (`DistributableJob`).
- 2 Utiliza los métodos provistos por esta interfaz:
 - `run()` : Ejecuta o inicia el trabajo.
 - `wait_completion()` : Espera (bloqueando) que el trabajo termine.

Además, es la responsabilidad del implementador de la aplicación:

- Definir el nombre de un trabajo (para logging).
- Definir el estado de un trabajo.
- Definir cómo se subdivide un trabajo.
- Definir cómo se incorporan resultados de una unidad de trabajo.

```
#include "counter.h"
#include "getopt_pp.h"

using namespace fud; using namespace GetOpt;

int main(int argc, char** argv) {
    size_t number(1000); //Default value.
    size_t jobs_n(5);    //Idem.

    GetOpt_pp ops(argc, argv);
    ops >>Option('n', "number", number) >>Option('j', "jobs", jobs_n);

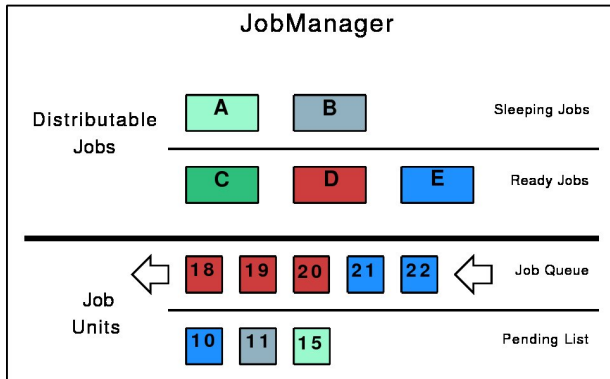
    NumberDatabase* db = new NumberDatabase(number);

    Counter* jobs[jobs_n];
    for (size_t i(0); i < jobs_n; ++i)
        jobs[i] = new Counter(*db,i);

    for (size_t i(0); i < jobs_n; ++i)
        jobs[i]->run();

    jobs[jobs_n-1]->wait_completion();
    std::cout << "Last num: " <<db->current_number() <<std::endl;
}
```

¿Cómo funciona una aplicación que usa **FuD**



Dependencias

FuD depende de los siguientes paquetes para su compilación:

Boost : Una colección muy completa de librerías para C++.

MiLi : Colección sencilla de pequeñas librerías útiles.

```
/* ... */
template <class T>
bostream& operator<< (T x)
{
    _s.append(reinterpret_cast<char*>(&x), sizeof(T));
    return *this;
}

/* Inserting a string inserts its size first. */
bostream& operator<< (const std::string& s)
{
    (*this) << s.size();
    _s += s;
    return *this;
}
```

Dependencias

FuD depende de los siguientes paquetes para su compilación:

Boost : Una colección muy completa de librerías para C++.

MiLi : Colección sencilla de pequeñas librerías útiles.

```
/* ... */
template <class T>
bostream& operator<< (T x)
{
    _s.append(reinterpret_cast<char*>(&x), sizeof(T));
    return *this;
}

/* Inserting a string inserts its size first. */
bostream& operator<< (const std::string& s)
{
    (*this) << s.size();
    _s += s;
    return *this;
}
```

Dependencias

FuD depende de los siguientes paquetes para su compilación:

Boost : Una colección muy completa de librerías para C++.

MiLi : Colección sencilla de pequeñas librerías útiles.

```
/* ... */
template <class T>
bostream& operator<< (T x)
{
    _s.append(reinterpret_cast<char*>(&x), sizeof(T));
    return *this;
}

/* Inserting a string inserts its size first. */
bostream& operator<< (const std::string& s)
{
    (*this) << s.size();
    _s += s;
    return *this;
}
```

Métricas

Files		Line Types			Percentages
Type	Count	Blank	Comment	Source	#Comms./Tot.
C++ source	10	145	360	704	33.83
C++ header	16	228	1223	592	67.38
Total	26	373	1583	1296	54.98

Cobertura de código

	Lines of Code		Percentage
File	Total	Executed	%
job_manager.cpp	117	95	81.2
distributable_job.cpp	40	39	97.5
clients_manager.cpp	43	38	98.37
async_io_clients_manager.cpp	75	56	74.67
job_unit.cpp	15	10	66.6
Total	290	238	82.07

Definición

Dados un conjunto de diferentes posibilidades geométricas para una proteína, representadas por un vector de átomos, agrupe estos elementos en clusters bajo similitudes geométricas de las posiciones de los átomos.

Dado un conjunto de proteínas D y una constante de corte c (en Å). Encuentre una partición $P \subseteq \wp(D)$ de D tal que:

- 1 $\forall C \in P$ (cluster) hay un elemento $r \in C$ (representante), tal que todos los miembros de C están a una distancia menor al corte (usando RMSD) a r y más cercanos a r que a otro representante.
- 2 La distancia entre dos representantes es mayor al corte.
- 3 La partición es mínima.

Definición

Dados un conjunto de diferentes posibilidades geométricas para una proteína, representadas por un vector de átomos, agrupe estos elementos en clusters bajo similitudes geométricas de las posiciones de los átomos.

Dado un conjunto de proteínas D y una constante de corte c (en Å). Encuentre una partición $P \subseteq \wp(D)$ de D tal que:

- 1 $\forall C \in P$ (cluster) hay un elemento $r \in C$ (representante), tal que todos los miembros de C están a una distancia menor al corte (usando RMSD) a r y más cercanos a r que a otro representante.
- 2 La distancia entre dos representantes es mayor al corte.
- 3 La partición es mínima.

Definición

Dados un conjunto de diferentes posibilidades geométricas para una proteína, representadas por un vector de átomos, agrupe estos elementos en clusters bajo similitudes geométricas de las posiciones de los átomos.

Dado un conjunto de proteínas D y una constante de corte c (en Å). Encuentre una partición $P \subseteq \wp(D)$ de D tal que:

- 1 $\forall C \in P$ (cluster) hay un elemento $r \in C$ (representante), tal que todos los miembros de C están a una distancia menor al corte (usando RMSD) a r y más cercanos a r que a otro representante.
- 2 La distancia entre dos representantes es mayor al corte.
- 3 La partición es mínima.

Definición

Dados un conjunto de diferentes posibilidades geométricas para una proteína, representadas por un vector de átomos, agrupe estos elementos en clusters bajo similitudes geométricas de las posiciones de los átomos.

Dado un conjunto de proteínas D y una constante de corte c (en Å). Encuentre una partición $P \subseteq \wp(D)$ de D tal que:

- 1 $\forall C \in P$ (cluster) hay un elemento $r \in C$ (representante), tal que todos los miembros de C están a una distancia menor al corte (usando RMSD) a r y más cercanos a r que a otro representante.
- 2 La distancia entre dos representantes es mayor al corte.
- 3 La partición es mínima.

$$O(2^n)$$

Pero esto es un problema de minimización perteneciente a la clase **NP-C** (demostración no presentada).

Como en nuestro caso, inclusive, el conjunto de datos es potencialmente bastante grande ($n > 10^6$), se debe buscar una aproximación coherente al problema.

La versión aquí presentada se caracteriza por ser computacionalmente eficiente, pero todavía mucho trabajo queda por delante para que los resultados de la misma sean de real interés. Esto se cubrirá en la sección sobre Trabajo a Futuro.

Mientras tanto, a continuación se detalla el algoritmo implementado.

$$O(2^n)$$

Pero esto es un problema de minimización perteneciente a la clase **NP-C** (demostración no presentada).

Como en nuestro caso, inclusive, el conjunto de datos es potencialmente bastante grande ($n > 10^6$), se debe buscar una aproximación coherente al problema.

La versión aquí presentada se caracteriza por ser computacionalmente eficiente, pero todavía mucho trabajo queda por delante para que los resultados de la misma sean de real interés. Esto se cubrirá en la sección sobre Trabajo a Futuro.

Mientras tanto, a continuación se detalla el algoritmo implementado.

$$O(2^n)$$

Pero esto es un problema de minimización perteneciente a la clase **NP-C** (demostración no presentada).

Como en nuestro caso, inclusive, el conjunto de datos es potencialmente bastante grande ($n > 10^6$), se debe buscar una aproximación coherente al problema.

La versión aquí presentada se caracteriza por ser computacionalmente eficiente, pero todavía mucho trabajo queda por delante para que los resultados de la misma sean de real interés. Esto se cubrirá en la sección sobre Trabajo a Futuro.

Mientras tanto, a continuación se detalla el algoritmo implementado.

$$O(2^n)$$

Pero esto es un problema de minimización perteneciente a la clase **NP-C** (demostración no presentada).

Como en nuestro caso, inclusive, el conjunto de datos es potencialmente bastante grande ($n > 10^6$), se debe buscar una aproximación coherente al problema.

La versión aquí presentada se caracteriza por ser computacionalmente eficiente, pero todavía mucho trabajo queda por delante para que los resultados de la misma sean de real interés. Esto se cubrirá en la sección sobre Trabajo a Futuro.

Mientras tanto, a continuación se detalla el algoritmo implementado.

Resumen de la implementación

Los siguientes pasos resumen el algoritmo:

- 1 Encuentra un conjunto de representantes R que cubren D .
- 2 Crea clusters cuyos representantes sean elementos de R .
- 3 Calcula la media geométrica de cada cluster.
- 4 Asigna nuevos representantes a cada cluster como el más cercano a la media.
- 5 Itera entre los pasos 2 (con nuevos representantes) y 4 tantas veces como desee el usuario.

Complejidad: $O(n^2)$ en teoría, pero muy cercano a $O(n)$ en la práctica (por la naturaleza del conjunto de entrada).

Resumen de la implementación

Los siguientes pasos resumen el algoritmo:

- 1 Encuentra un conjunto de representantes R que cubren D .
- 2 Crea clusters cuyos representantes sean elementos de R .
- 3 Calcula la media geométrica de cada cluster.
- 4 Asigna nuevos representantes a cada cluster como el más cercano a la media.
- 5 Itera entre los pasos 2 (con nuevos representantes) y 4 tantas veces como desee el usuario.

Complejidad: $O(n^2)$ en teoría, pero muy cercano a $O(n)$ en la práctica (por la naturaleza del conjunto de entrada).

Resumen de la implementación

Los siguientes pasos resumen el algoritmo:

- 1 Encuentra un conjunto de representantes R que cubren D .
- 2 Crea clusters cuyos representantes sean elementos de R .
- 3 Calcula la media geométrica de cada cluster.
- 4 Asigna nuevos representantes a cada cluster como el más cercano a la media.
- 5 Itera entre los pasos 2 (con nuevos representantes) y 4 tantas veces como desee el usuario.

Complejidad: $O(n^2)$ en teoría, pero muy cercano a $O(n)$ en la práctica (por la naturaleza del conjunto de entrada).

Resumen de la implementación

Los siguientes pasos resumen el algoritmo:

- 1 Encuentra un conjunto de representantes R que cubren D .
- 2 Crea clusters cuyos representantes sean elementos de R .
- 3 Calcula la media geométrica de cada cluster.
- 4 Asigna nuevos representantes a cada cluster como el más cercano a la media.
- 5 Itera entre los pasos 2 (con nuevos representantes) y 4 tantas veces como desee el usuario.

Complejidad: $O(n^2)$ en teoría, pero muy cercano a $O(n)$ en la práctica (por la naturaleza del conjunto de entrada).

Resumen de la implementación

Los siguientes pasos resumen el algoritmo:

- 1 Encuentra un conjunto de representantes R que cubren D .
- 2 Crea clusters cuyos representantes sean elementos de R .
- 3 Calcula la media geométrica de cada cluster.
- 4 Asigna nuevos representantes a cada cluster como el más cercano a la media.
- 5 Itera entre los pasos 2 (con nuevos representantes) y 4 tantas veces como desee el usuario.

Complejidad: $O(n^2)$ en teoría, pero muy cercano a $O(n)$ en la práctica (por la naturaleza del conjunto de entrada).

Resumen de la implementación

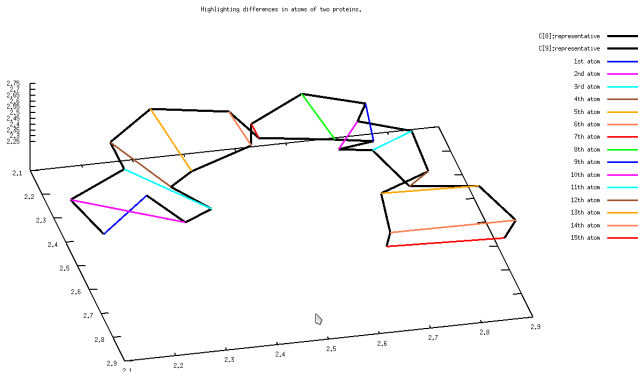
Los siguientes pasos resumen el algoritmo:

- 1 Encuentra un conjunto de representantes R que cubren D .
- 2 Crea clusters cuyos representantes sean elementos de R .
- 3 Calcula la media geométrica de cada cluster.
- 4 Asigna nuevos representantes a cada cluster como el más cercano a la media.
- 5 Itera entre los pasos 2 (con nuevos representantes) y 4 tantas veces como desee el usuario.

Complejidad: $O(n^2)$ en teoría, pero muy cercano a $O(n)$ en la práctica (por la naturaleza del conjunto de entrada).

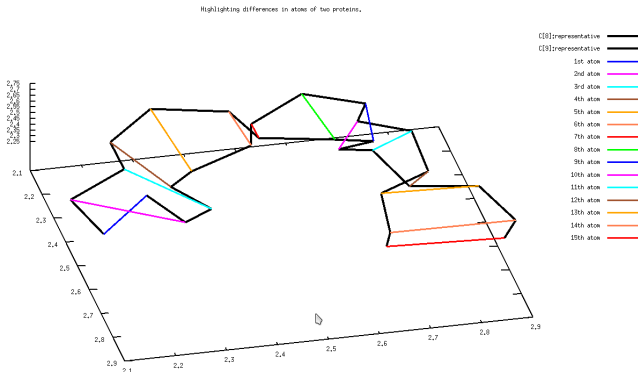
RMSD

La diferencia entre dos proteínas se calcula usando RMSD, mediante una media de las sumas de las diferencias entre átomos análogos de las estructuras[Coutsias et al.,].



RMSD

La diferencia entre dos proteínas se calcula usando RMSD, mediante una media de las sumas de las diferencias entre átomos análogos de las estructuras[Coutsias et al.,].



Alternativas

RMSD con rotación

La comparación utilizada utiliza superimposición rígida de las estructuras. Esto puede ocasionar falsos negativos entre estructuras similares pero rotadas. Es posible rotar de manera que el cálculo rígido de RMSD es mínimo[Li et al., 2006].

Scaled Gauss Metrics

Una buena alternativa es utilizar invariantes de nudos para representar la forma de una proteína mediante 30 valores. Este enfoque es bueno para macro-clasificación[Røgen and Fain, 2002].

Muchos otros enfoques son posibles. En [Koehl, 2001] se encuentra un buen resumen de técnicas de comparación disponibles.

Alternativas

RMSD con rotación

La comparación utilizada utiliza superimposición rígida de las estructuras. Esto puede ocasionar falsos negativos entre estructuras similares pero rotadas. Es posible rotar de manera que el cálculo rígido de RMSD es mínimo[Li et al., 2006].

Scaled Gauss Metrics

Una buena alternativa es utilizar invariantes de nudos para representar la forma de una proteína mediante 30 valores. Este enfoque es bueno para macro-clasificación[Røgen and Fain, 2002].

Muchos otros enfoques son posibles. En [Koehl, 2001] se encuentra un buen resumen de técnicas de comparación disponibles.

Alternativas

RMSD con rotación

La comparación utilizada utiliza superimposición rígida de las estructuras. Esto puede ocasionar falsos negativos entre estructuras similares pero rotadas. Es posible rotar de manera que el cálculo rígido de RMSD es mínimo[Li et al., 2006].

Scaled Gauss Metrics

Una buena alternativa es utilizar invariantes de nudos para representar la forma de una proteína mediante 30 valores. Este enfoque es bueno para macro-clasificación[Røgen and Fain, 2002].

Muchos otros enfoques son posibles. En [Koehl, 2001] se encuentra un buen resumen de técnicas de comparación disponibles.

Implementando el Clusterer con FuD

Para implementar el clusterer, en **FuD** usamos cuatro tipos de trabajos distribuibles:

Representatives : Iteración inicial para buscar los representantes.

Clusters : Arma los clusters con estos representantes.

Adding : Calcula los centros geométricos de un cluster armado.

Centers : Selecciona de un cluster el miembro más cercano a la media geométrica para usar como nuevo representante.

Implementando el Clusterer con FuD

Para implementar el clusterer, en **FuD** usamos cuatro tipos de trabajos distribuibles:

Representatives : Iteración inicial para buscar los representantes.

Clusters : Arma los clusters con estos representantes.

Adding : Calcula los centros geométricos de un cluster armado.

Centers : Selecciona de un cluster el miembro más cercano a la media geométrica para usar como nuevo representante.

Implementando el Clusterer con FuD

Para implementar el clusterer, en **FuD** usamos cuatro tipos de trabajos distribuibles:

Representatives : Iteración inicial para buscar los representantes.

Clusters : Arma los clusters con estos representantes.

Adding : Calcula los centros geométricos de un cluster armado.

Centers : Selecciona de un cluster el miembro más cercano a la media geométrica para usar como nuevo representante.

Implementando el Clusterer con FuD

Para implementar el clusterer, en **FuD** usamos cuatro tipos de trabajos distribuibles:

Representatives : Iteración inicial para buscar los representantes.

Clusters : Arma los clusters con estos representantes.

Adding : Calcula los centros geométricos de un cluster armado.

Centers : Selecciona de un cluster el miembro más cercano a la media geométrica para usar como nuevo representante.

Implementando el Clusterer con FuD

Para implementar el clusterer, en **FuD** usamos cuatro tipos de trabajos distribuibles:

Representatives : Iteración inicial para buscar los representantes.

Clusters : Arma los clusters con estos representantes.

Adding : Calcula los centros geométricos de un cluster armado.

Centers : Selecciona de un cluster el miembro más cercano a la media geométrica para usar como nuevo representante.

Código parcial del main del clusterer

```
ProteinDatabase* db = new ProteinDatabase( input_file.c_str() );  
  
std::vector<Cluster> clusters;  
  
RepresentativesJob* repjob = new RepresentativesJob(*db, cutoff);  
repjob->run();  
repjob->wait_completion();  
  
ClustersJob* clusters_job = new ClustersJob(*db, repjob->  
    get_marked_ids_vector(), clusters, cutoff);  
clusters_job->run();  
clusters_job->wait_completion();  
  
AddingJob* adding_job = new AddingJob(*db, clusters, cutoff);  
adding_job->run();  
adding_job->wait_completion();  
  
CentersJob* centers_job = new CentersJob(*db, clusters, cutoff);  
centers_job->run();  
centers_job->wait_completion();
```

Output del Clusterer

El clusterer permite generar las siguientes salidas:

Texto : Genera un archivo simple de texto con las posiciones de los centros encontrados respecto del archivo original de entrada.

Centros : Genera un .xtc con todos los centros de los clusters.

Centros geométricos : Genera un .xtc con la media geométrica de cada cluster.

Clusters : Genera un .xtc para cada cluster, con el representante como primer elemento.

Estas salidas pueden luego ser interpretadas por las herramientas, desarrolladas con este propósito, de `xtc2pdb` y `xtc2gnuplot`.

Output del Clusterer

El clusterer permite generar las siguientes salidas:

Texto : Genera un archivo simple de texto con las posiciones de los centros encontrados respecto del archivo original de entrada.

Centros : Genera un `.xtc` con todos los centros de los clusters.

Centros geométricos : Genera un `.xtc` con la media geométrica de cada cluster.

Clusters : Genera un `.xtc` para cada cluster, con el representante como primer elemento.

Estas salidas pueden luego ser interpretadas por las herramientas, desarrolladas con este propósito, de `xtc2pdb` y `xtc2gnuplot`.

Output del Clusterer

El clusterer permite generar las siguientes salidas:

Texto : Genera un archivo simple de texto con las posiciones de los centros encontrados respecto del archivo original de entrada.

Centros : Genera un .xtc con todos los centros de los clusters.

Centros geométricos : Genera un .xtc con la media geométrica de cada cluster.

Clusters : Genera un .xtc para cada cluster, con el representante como primer elemento.

Estas salidas pueden luego ser interpretadas por las herramientas, desarrolladas con este propósito, de `xtc2pdb` y `xtc2gnuplot`.

Output del Clusterer

El clusterer permite generar las siguientes salidas:

Texto : Genera un archivo simple de texto con las posiciones de los centros encontrados respecto del archivo original de entrada.

Centros : Genera un .xtc con todos los centros de los clusters.

Centros geométricos : Genera un .xtc con la media geométrica de cada cluster.

Clusters : Genera un .xtc para cada cluster, con el representante como primer elemento.

Estas salidas pueden luego ser interpretadas por las herramientas, desarrolladas con este propósito, de `xtc2pdb` y `xtc2gnuplot`.

Output del Clusterer

El clusterer permite generar las siguientes salidas:

Texto : Genera un archivo simple de texto con las posiciones de los centros encontrados respecto del archivo original de entrada.

Centros : Genera un .xtc con todos los centros de los clusters.

Centros geométricos : Genera un .xtc con la media geométrica de cada cluster.

Clusters : Genera un .xtc para cada cluster, con el representante como primer elemento.

Estas salidas pueden luego ser interpretadas por las herramientas, desarrolladas con este propósito, de `xtc2pdb` y `xtc2gnuplot`.

Métricas I

Files		Line Types			Percentages
Type	Count	Blank	Comment	Source	#Comms./Tot.
C++ source	11	309	469	938	33.33
C++ header	10	151	321	441	42.12
Total	21	460	790	1379	36.42

Métricas II

	Lines of Code		Percentage
File	Total	Executed	%
protein_database.cpp	43	40	93.02
clusterer_processor.cpp	61	59	96.72
representatives_job.cpp	50	49	98
clusters_job.cpp	48	45	93.75
adding_job.cpp	49	49	100
centers_job.cpp	51	51	100
Total	302	293	97.02

Conclusión

Sobre **FuD**, postulamos que el diseño aquí presentado cumple los requisitos originales. Las aplicaciones de prueba que se han desarrollado para utilizar el framework cubren la mayoría de los tipos de uso posibles.

Requisitos

Desarrollar una aplicación que use el framework no requiere pensar en términos de multi-threading o programación paralela.

El clusterer

Sobre el clusterer, postulamos que la solución es una aproximación aceptable. Sin embargo este campo es vasto y un análisis concienzudo del problema de seguro dará a lugares a implementaciones más eficientes y resultados de mayor relevancia. El uso de Scaled Gauss Metrics es una opción prometedora.

Conclusión

Sobre **FuD**, postulamos que el diseño aquí presentado cumple los requisitos originales. Las aplicaciones de prueba que se han desarrollado para utilizar el framework cubren la mayoría de los tipos de uso posibles.

Requisitos

Desarrollar una aplicación que use el framework no requiere pensar en términos de multi-threading o programación paralela.

El clusterer

Sobre el clusterer, postulamos que la solución es una aproximación aceptable. Sin embargo este campo es vasto y un análisis concienzudo del problema de seguro dará a lugares a implementaciones más eficientes y resultados de mayor relevancia. El uso de Scaled Gauss Metrics es una opción prometedora.

Conclusión

Sobre **FuD**, postulamos que el diseño aquí presentado cumple los requisitos originales. Las aplicaciones de prueba que se han desarrollado para utilizar el framework cubren la mayoría de los tipos de uso posibles.

Requisitos

Desarrollar una aplicación que use el framework no requiere pensar en términos de multi-threading o programación paralela.

El clusterer

Sobre el clusterer, postulamos que la solución es una aproximación aceptable. Sin embargo este campo es vasto y un análisis concienzudo del problema de seguro dará a lugares a implementaciones más eficientes y resultados de mayor relevancia. El uso de Scaled Gauss Metrics es una opción prometedora.

Trabajo a Futuro

Las siguientes tareas deberían llevarse a cabo:

- **Compilación con autotools.**
- Uso de BOINC.
- Uso de MPI para clusters.
- Clientes multi-threaded.
- Clientes que se pueden desconectar.
- Capaz de aplicación abstractas.

Trabajo a Futuro

Las siguientes tareas deberían llevarse a cabo:

- Compilación con autotools.
- Uso de BOINC.
- Uso de MPI para clusters.
- Clientes multi-threaded.
- Clientes que se pueden desconectar.
- Capaz de aplicación abstractas.

Trabajo a Futuro

Las siguientes tareas deberían llevarse a cabo:

- Compilación con autotools.
- Uso de BOINC.
- Uso de MPI para clusters.
- Clientes multi-threaded.
- Clientes que se pueden desconectar.
- Capaz de aplicación abstractas.

Trabajo a Futuro

Las siguientes tareas deberían llevarse a cabo:

- Compilación con autotools.
- Uso de BOINC.
- Uso de MPI para clusters.
- Clientes multi-threaded.
- Clientes que se pueden desconectar.
- Capaz de aplicación abstractas.

Trabajo a Futuro

Las siguientes tareas deberían llevarse a cabo:

- Compilación con autotools.
- Uso de BOINC.
- Uso de MPI para clusters.
- Clientes multi-threaded.
- Clientes que se pueden desconectar.
- Capaz de aplicación abstractas.

Trabajo a Futuro

Las siguientes tareas deberían llevarse a cabo:

- Compilación con autotools.
- Uso de BOINC.
- Uso de MPI para clusters.
- Clientes multi-threaded.
- Clientes que se pueden desconectar.
- Capaz de aplicación abstractas.

Bibliografía I



Aas, J. (2005).

Understanding the linux 2.6.8.1 cpu scheduler.



Anderson, D. P. (2004).

Boinc: A system for public-resource computing and storage.

5th IEEE/ACM International Workshop on Grid Computing.



Anderson, D. P. and Fedak, G. (2009).

The computational and storage potential of volunteer computing.



Booch, G., Rumbaugh, J., and Jacobson, I. (2005).

Unified Modeling Language User Guide.

Second edition.



Bourke, T. (2001).

Server Load Balancing.

O'Reilly Media.

Bibliografía II



Brucker, P. (1998).
Scheduling algorithms.
Springer, Heidelberg, second edition.



Chapman, B., Jost, G., and van der Pas, R. (2007).
Using OpenMP. Portable Shared Memory Parallel Programming.
MIT Press, Massachusetts Institute of Technology, Cambridge, MA.



Coutsias, E. A., Seok, C., and Dill, K. A.
Using quaternions to calculate rmsd.
Comput. Chem., pages 1849–1857.



Dean, J. and Ghemawat, S. (2004).
Mapreduce: Simplified data processing on large clusters.



Dijkstra, E. W. (1974).
Self-stabilizing systems in spite of distributed control.
Communications of the ACM, 17:643–644.

Bibliografía III



Dijkstra, E. W. (1988).

On the cruelty of really teaching computing science.



Flynn, M. J. (1972).

Some computer organizations and their effectiveness.

IEEE Trans. Comput., pages 948–960.



Foster, I. (1995).

Designing and Building Parallel Programs.

Addison-Wesley.



Gan, G., Ma, C., and Wu, J. (2007).

Data Clustering, Theory, Algorithms, and Applications.

ASA-SIAM Series on Statistics an Applied Probability.



Herlihy, M. and Shavit, N. (1999).

The topological structure of asynchronous computation.

Journal of the ACM, 46:858–923.

Bibliografía IV



Huang, Z. (2000).

K-means-type algorithms on distributed memory computer.
International Journal of High Speed Computing, 11:75–91.



Karlsson, B. (2005).

Beyond the C++ Standard Library: An Introduction to Boost.
Addison-Wesley Professional.



Knuth, D. E. (1984).

The T_EXbook, Volume A of Computers and Type-setting.
Addison-Wesley, Reading, Massachusetts.



Koehl, P. (2001).

Protein structure similarities.
Curr. Opin. Struct. Biol., pages 348–353.

Bibliografía V



Lamport, L. (1994).

LaTeX: A Document Preparation System.

Addison-Wesley, Reading, Massachusetts.



Lehninger, A. L., Nelson, D. L., and Cox, M. M. (2004).

Lehninger Principles of Biochemistry.

W.H. Freeman, 4th. edition.



Li, Q., Shu, J., Shi, Z., and Zhang, D. (2006).

A general solution for the optimal superimposition of protein structures.

ICIC, pages 352–361.



Macqueen, J. (1967).

Some methos for classification and analysis of multivariate observations.

1:281–297.

Bibliografía VI



Maiorov V.N. and Crippen G.M. (1994).

Significance of root-mean-square deviation in comparing three-dimensional structures of globular proteins.

Molecular Biology, 235:625–634.



Meyer, B. (1997).

Object-Oriented Software Construction, Second Edition.

Prentice Hall Professional Technical Reference, Santa Barbara.



Murray, R. F., Harper, H. W., Granner, D. K., Mayes, P. A., and Rodwell, V. W. (2006).

Harper's Illustrated Biochemistry.

Lange Medical Books/McGraw-Hill, New York.



Oetiker, T., Partl, H., Hyna, I., and Schlegl, E. (2008).

The Not So Short Introduction to L^AT_EX 2_ε.

Bibliografía VII



Pelleg, D. and Moore, A. (1999).

Accelerating exact k -means algorithms with geometric reasoning.
pages 277–281.



Pelleg, D. and Moore, A. (2000).

x -means: Extending k -means with efficient estimation of the number of clusters.
pages 727–734.



Perks, M. (2003).

Best practices for software development projects.



Pressman, R. S. (1982).





Software Engineering - A Practitioner's Approach - Fourth Edition.



Røgen, P. and Fain, B. (2002).

Automatic classification of protein structure by using gauss integrals.

Bibliografía VIII

-  Sasson, O. and Linial, M. (2007).
Protein clustering and classification.
The New Avenues in Bioinformatics, 8:203–226.
-  Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J. (1996).
MPI: The Complete Reference.
MIT Press, Massachusetts Institute of Technology, Cambridge, MA.
-  Stevens, W., Myers, G., and Constantine, L. (1974).
Structured design.
IBM Systems Journal, 13:115–139.
-  Stoffle, K. and Belkonience, A. (1999).
Parallel k-means clustering for large datasets.
Proceedings of EuroPar.

Bibliografía IX



Stroustrup, B. (1997).

The C++ Programming Language, Third Edition.
Addison-Wesley, Murray Hill, New Jersey.



Sussman, B. C., Fitzpatrick, B. W., and Pilato, C. M. (2008).
Version Control with Subversion.
O'Reilly.



Tanenbaum, A. S. (2001).
Modern Operating Systems.
Prentice-Hall, second edition.



Wirfs-Brock, R. and McKean, A. (2003).
Object Design: Roles, Responsibilities, and Collaborations.
Addison-Wesley.

Bibliografía X



Woodhull, A. S. and Tanenbaum, A. S. (1997).
Operating Systems Design and Implementation.
Prentice-Hall, second edition.



Zait, M. and Messatfa, H. (1997).
A comparative study of clustering methods.
Future Generation Computer Systems, 13:149–159.



Zhang, Y., Xiong, Z., Mao, J., and Ou, L. (2006).
The study of parallel k-means algorithm.
Proceedings of the 6th World Congress on Intelligent Control and Automation.

¿Preguntas?

Gracias

[Foster, 1995] [Sussman et al., 2008] [Oetiker et al., 2008] [Knuth, 1984]
[Lamport, 1994] [Meyer, 1997] [Gan et al., 2007]
[Maiorov V.N. and Crippen G.M., 1994] [Zhang et al., 2006]
[Stevens et al., 1974] [Pressman, 1982] [Herlihy and Shavit, 1999]
[Dijkstra, 1974] [Huang, 2000] [Stoffle and Belkonience, 1999]
[Li et al., 2006] [Røgen and Fain, 2002] [Koehl, 2001] [Coutsias et al.,]
[Dean and Ghemawat, 2004] [Flynn, 1972] [Zait and Messatfa, 1997]
[Stroustrup, 1997] [Anderson, 2004] [Anderson and Fedak, 2009]
[Lehninger et al., 2004] [Aas, 2005] [Woodhull and Tanenbaum, 1997]
[Tanenbaum, 2001] [Brucker, 1998] [Bourke, 2001] [Macqueen, 1967]
[Pelleg and Moore, 1999] [Pelleg and Moore, 2000] [Murray et al., 2006]
[Sasson and Linial, 2007] [Booch et al., 2005] [Perks, 2003]
[Snir et al., 1996] [Chapman et al., 2007] [Dijkstra, 1988]
[Wirfs-Brock and McKean, 2003] [Karlsson, 2005]