# Telecomunicaciones y Sistemas Distribuidos

Proyecto final

2017

## 1. Descripción

El objetivo de este proyecto es el desarrollo de una simple estructura de datos y primitivas de sincronización que permita implementar el modelo de programación *Partitioned Global Address Space (PGAS)* en una arquitectura en red (multicomputadora), ocultando los detalles de comunicación entre procesos.

El modelo *PGAS* presenta al programador un sistema multicomputador, donde cada proceso/procesador cuenta con su propia memoria local o privada y aporta bloques a una memoria compartida.

De esta manera permite al programador un estilo de programación de *memoria compartida* aún en un ambiente de memoria distribuida (particionada) donde el acceso a la memoria remota se realiza por medio de mecanismos de comunicación (mensajes).

A modo de ejemplo, se muestra el siguiente pseudo-código que ordena los elementos de un arreglo en forma distribuida (en paralelo):

```
// Set of process 0, 1, ..., N-1
init()                          // initialize system
distributed int[N*1000] a       // distribute array on N nodes
bool finish = false             // one copy in each processor
int p = pid()                   // local (private) variable

while not finish:
  finish = true

  // sort local block
  bubble_sort(a.lowerindex(p), a.upperindex(p))
  barrier()

  if not Iam(N - 1):
    if a[a.upperindex(p)] > a[a.lowerindex(p+1)]:
      swap(a.upperindex(p), a.lowerindex(right))
      finish = false          // update local copy

  // reduce finish by and, then replicate result
  finish = and_reduce(finish)
```

Notas:

- `a.lowerindex(p)` obtiene el índice menor del bloque o partición local de `a` en el proceso `p`.
- `a.upperindex(p)` obtiene el índice mayor del bloque o partición local de `a` en el proceso `p`.
- `barrier()` sincroniza (*rendezvous*) los procesos.
- `and_reduce(v)` genera una reducción de los valores de las copias locales de `v` aplicando la operación lógica *and*. También actúa como un punto de sincronización.

Este modelo de programación se conoce como *one program multiple data* ya que es el mismo programa corriendo en paralelo en todos los nodos pero operando sobre diferentes datos.

Las primitivas `barrier()` y `reduce` se conocen como operaciones de *comunicación global* ya que requieren el intercambio de mensajes entre todos los procesos.

Se pide el diseeño e implementación de los siguientes componentes:

1. Una estructura de datos de tipo `distributed array`, que distrubuya en bloques uniformes entre los procesos.

   Debería contener operaciones o métodos `read(i)` y `write(i,v)`.

2. Implementar `barrier()`

3. Implementar `and_reduce()`

## 2. Consideraciones de diseño

Para una mejor y simple implementación, se debería definir una capa de abstracción de manejo de mensajes que corra en el contexto de un thread de cada proceso y provea las facilidades de transmisión, recepción y entrega de mensajes al proceso.

Esta capa debería proveer operaciones para que un estructura de datos distribuida registre los tipos de mensajes recibidos (junto con un método o función *callback*) para que puedan ser entregados los mensajes de lectura y escritura de elementos. En este caso, la capa de gestión de mensajes actuaría como un *dispatcher* de operaciones ante el arribo de mensajes.

Los otros tipos de mensajes, requeridos para implementar las primitivas de sincronización y reducción, deberían ser encolados () y entregados bajo demanda por el proceso mismo.

Además, esta capa (*middleware*) debería encapsular el protocolo de transporte utilizado.

La figura 2 muestra el esquema de diseño propuesto.

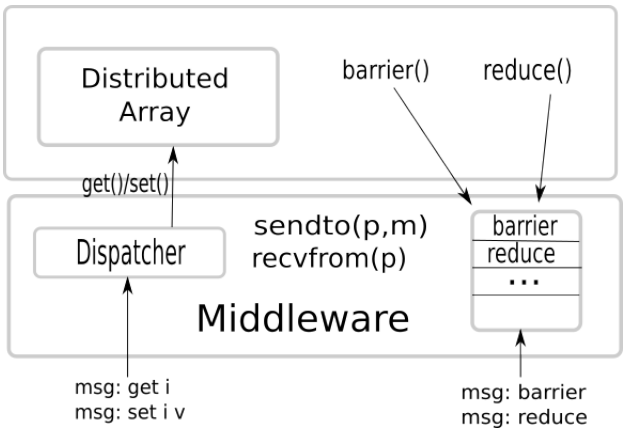


Figura 1: Arquitectura del sistema

## Referencias

[1] Ajay D. Kshemkalyani, Mukesh Shingal. *Distributed Computing. Principles, Algorithms and Systems*. Cambridge University Press. ISBN-13: 978-0-511-39341-9. 2008.

[2] *Partitioned Global Address Space*.

`https://en.wikipedia.org/wiki/Partitioned_global_address_space`.