TP2 - Algoritmos en sistemas distribuidos

Sistemas Operativos - Segundo cuatrimestre de 2017

Límite de entrega: Miércoles 25 de Octubre, 23:59 hs.

1. Introducción

A través de este trabajo, se busca ahondar en el conocimiento de los sistemas distribuidos mediante el envío de mensajes. La implementación se hará en el lenguaje C++ utilizando la interfaz MPI, de la que luego daremos más detalles.

Deberán implementar un cómputo distribuido que será llevado adelante por un conjunto de $n \geq 1$ procesos, que podrán ejecutar en $m \geq 1$ máquinas. A estos procesos los llamaremos **nodos**.

2. Problema a resolver

Se pide desarrollar un *DistributedHashMap*. Este no es más que una versión *MPI* del archiconocido *ConcurrentHashMap* del TP2.

Para esto, contarán con un nodo distinguido (la **consola**), que se encargará de la entrada y salida de los comandos y sus resultados.

El resto de los nodos no serán distinguibles y cada uno deberá inicializar su propio *HashMap* local para realizar las tareas pedidas por la consola. Para esto les brindaremos una implementación completa de un *HashMap*, que podrán encontrar en los archivos `HashMap.hpp` y `HashMap.cpp`. El código base de los nodos se encuentra en `nodo.cpp`.

Justo antes de enviar cualquier mensaje, los nodos deben realizar tareas de procesamiento durante una cantidad de tiempo no determinística. Deberán simular este comportamiento realizando llamadas a la función `trabajarArduamente()` donde corresponda.

Por su parte, el código base de la consola se encuentra en `consola.cpp`. La consola debe responder a los siguientes comandos:

```
* load <arch_1> <arch_2> ... <arch_n>
* addAndInc <string>
* member <string>
* maximum
* q|quit
```

A continuación se explica el funcionamiento de cada uno de ellos:

- `void load(list<string> params)`: Toma como parámetro una lista de archivos de texto y carga las palabras de estos archivos en el *DistributedHashMap*. Deberá utilizar un nodo para cada archivo. Si hay más archivos que nodos, deberá enviarle un nuevo archivo a cada nodo que vaya liberándose.

- `void addAndInc(string key)`: Si *key* existe en el *DistributedHashMap*, incrementa su valor. Si no existe, crea el par $(key, 1)$ en algún nodo. No tiene por qué ser procesada siempre por el mismo nodo, ni por uno que ya contenga la clave. Deberá enviarse el comando a todos los nodos; el primer nodo que lo tome deberá avisar que lo ha hecho, y el resto no deberá ejecutarlo.
- `void member(string key)`: Dado un *string*, determina si el mismo existe o no en el sistema.
- `void maximum()`: Busca el *string* que más veces aparece y su valor. Luego, imprime el resultado por pantalla. La consola tendrá que pedir a cada nodo las palabras allí almacenadas, y deberá poder recibirlas de distintos nodos, sin que necesariamente todas las de un mismo nodo lleguen juntas. El *HashMap* dado provee un iterador (`HashMap::iterator`) que permite obtener una a una las palabras almacenadas. Una vez que la consola tenga el *HashMap* completo podrá calcular el resultado.

  Si un mismo nodo utiliza mútiples mensajes para enviar sus palabras a la consola, debe realizar una única llamada a `trabajarArduamente()`, en lugar de una por cada mensaje.
- `void quit()`: Avisa a todos los nodos que deben terminar y liberar sus recursos.

3. Implementación

Para implementar el TP deberán utilizar MPI[1]. Entre las funciones de MPI que podrían resultarles útiles, se encuentran:

- `MPI::COMM_WORLD.Send()`: Envío bloqueante de mensajes.
- `MPI::COMM_WORLD.Recv()`: Recepción bloqueante de mensajes.
- `MPI::COMM_WORLD.Probe()`: Espera de manera bloqueante la llegada de un mensaje.
- `MPI::COMM_WORLD.Isend()`: Envío no bloqueante de mensajes.
- `MPI::COMM_WORLD.Irecv()`: Lectura no bloqueante de mensajes.
- `MPI::COMM_WORLD.Iprobe()`: Indica si hay o no mensajes, de manera no bloqueante.
- `MPI::COMM_WORLD.Wait()`: Espera de manera bloqueante que se complete el envío o la recepción de un mensaje.

Para encontrar una descripción más detallada de ellas, puede consultar https://computing.llnl.gov/tutorials/mpi.

4. Ejecución

El sistema se ejecutará indicando por parámetro cuántos procesos se utilizarán. Luego, uno de ellos será el encargado de la entrada y salida por consola. Ese será el único nodo distinguido; el resto serán indistinguibles para nosotros (excepto para debug).

Para ejecutar el programa compilado debe usarse `mpiexec -np N ./dist_hashmap`, donde `N` es la cantidad de procesos a lanzar.

[1] http://www.mpi-forum.org/docs/docs.html

5. Requerimientos

El objetivo principal del trabajo es implementar correctamente el *DistributedHashMap*.

La solución no debe violar la exclusión mutua ni exhibir *deadlock*, *livelock* o inanición para ninguna combinación de parámetros y `-np` razonable. Tampoco debe introducir componentes centralizados fuera de los especificados en el apartado anterior ni suponer la existencia de relojes globalmente sincronizados. Otros problemas menores no serán causal de reentrega, siempre y cuando estén documentados en el informe que acompañará el trabajo, listando limitaciones conocidas y conjeturando posibles causas.

Para el desarrollo puede usarse MPICH2 [3], OpenMPI [4] o cualquier otra implementación del estándar MPI-2 [1] en versión no-obsoleta.

Deberán incluir un conjunto de casos de prueba que permita verificar que el código cumple con los requerimientos del enunciado. Estos deberán ir acompañados de instrucciones acerca de cómo ejecutarlos, y deberán explicarse sus objetivos y resultados esperados.

Además, deberán elaborar un informe de caracter breve donde expongan, a grandes rasgos, las ideas generales detrás de su implementación, así como los puntos ya mencionados sobre sus limitaciones y sobre los casos de *testing* elaborados.

6. Entregable

Para entregar el TP2, envien un mail a la lista de docentes: so-doc@dc.uba.ar

Con la siguiente informacion:

únicamente:

- El código fuente debidamente documentado, **con Makefile** (**NO** incluir ejecutables).
- Los casos de prueba que hayan realizado.
- El documento del informe (en **PDF**).

Al recibir correctamente una entrega, el sistema enviará un mail de confirmación a todos los integrantes del grupo. Si es necesario, podrán realizar múltiples envíos, en cuyo caso el corrector solo tendrá en cuenta el último envío que se haya realizado antes de la fecha límite de entrega.

Referencias

[1] *MPI: A Message Passing Interface Standard.*
http://www.mpi-forum.org/docs/docs.html

[2] MPI Tutorial @ LNLL (muy recomendable)
https://www.llnl.gov/computing/tutorials/mpi/

[3] MPICH2, http://www.mcs.anl.gov/research/projects/mpich2/

[4] OpenMPI, http://www.open-mpi.org/