Implementacion

Se necesita una implementación en C/C++ que use OpenMP . El sistema donde se ejecutará es un Lubuntu 20.04 LTS, se usarán configuraciones que harán variar la cantidad de CPUs (entre 1 a 8 CPU) y la memoria RAM3 (entre 1 y 16 GB). La ejecución del programa no puede exceder de 60 minutos para el caso de 3 cpu con 6 GB de ram.

https://es.wikipedia.org/wiki/OpenMP
Intel(R) Core(TM) i5-4440 CPU @ 3.10GHz
DIMM DDR3 Synchronous 1600 MHz

Notas

Intel(R) Core(TM) i5-4440 CPU @ 3.10GHz tiene 4 procesadores lógicos, no se entiende que se usen 8 a no ser que se implemente una arquitectura de dos procesadores compartiendo la placa madre, y en el caso de ser una implementación en ambiente virtual, se haría necesario utilizar dos PC's distintos, osea intercomunicación entre PC's.

Asumiendo los requerimientos mínimos, se cumple que el procesador es capaz de utilizar hasta 4 procesadores lógicos, dentro de un ambiente con procesador AMD-fx 8320 @ 3.5 GHz de 8 procesadores lógicos (CPU's), y 8GB de RAM en kingston blu ddr3 1066MHz 2x4 8gb ram. Para esto se hará una estimación en la equivalencia del procesador y ram para medir los requerimientos mínimos.

Archivo

La parte más complicada del examen consiste en procesar el archivo de log. La empresa entregó un archivo de 21GB, lo cual dificulta procesarlo en RAM, por lo que parte de la evaluación es encontrar una manera óptima de procesar este archivo.

*El formato del log es: **%h %^[%d:%t %^] "%r" %s %b "%R" "%u" %D***
Dónde:

- ***%h** Dirección ip del cliente.*
- ***%^[%d:%t %^]** Fecha Hora y Zona horaria del requerimiento.*
- ***"%r"** "Verbo HTTP Requerimiento y versión del protocolo"*
- ***%s** Código de Estado HTTP*
- ***%b** Tamaño del recurso*
- ***"%R"** Referer de petición*
- ***"%u** Agente de usuario (identificación del browser)*
- ***"%D"** Tiempo de ejecución en microsegundos.*

Ejemplo:

190.100.148.8 - - [30/Aug/2020:06:44:35 -0400] "GET / HTTP/1.1" 200 11673 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Safari/605.1.15" 38923

Notas

En el ejemplo que se da, cabe señalar que no es claro el por qué después de la ip hay dos guiones “- - “ (guión espacio guión espacio). El último de estos es atribuible a la expresión **%^[%d:%t %^]** donde **%^** representa la zona horaria “-0400” o UTC -04:00, de ahí que se pueda explicar el origen del guión (posiblemente) “- [fecha:hora zona horaria]”, sin embargo no es claro la razón del primer separador “-” ni el espacio que le sigue a ambos.

Para el desarrollo del programa sin embargo, es fácil omitir esta particularidad, puesto que la información realmente importante se encuentra en *[30/Aug/2020:06:44:35 -0400].* Donde se hace necesario determinar si la fecha es dependiente de la zona horaria o el archivo.log registro los horarios de acceso al sistema con una zona horaria fija, o el agregado de la zona horaria corresponde al usuario, por lo cual en el archivo log existirian mas de una zona horaria.

Concluyendo, lo más probable es que la zona horaria no varíe en el archivo log y esto corresponda a la zona horaria del servidor y no del usuario. En caso contrario habría que ordenar las fecha por zona horaria y no parece ser plausible.

Desarrollo

Ya entendiendo que la información crucial se encuentra en *[30/Aug/2020:06:44:35 -0400],* En la lectura del archivo hay que buscar en la línea el inicio y final del corchete. Guardarlo en un array y procesarlo posteriormente.

Debido a la dificultad que presenta procesar un archivo tan grande en ram, uno de los caminos a seguir es dividir el archivo en varias partes que no superen la cantidad mínima de 6GB, y establecer un rango de tamaño de archivo de acuerdo a la velocidad del procesador, la cantidad de archivos a procesar y el espacio que ocupasen en la memoria RAM. Por ejemplo 42 archivos de 500mb, o 84 de 250mb. Establecer el balance de acuerdo al requerimiento mínimo y la variedad de CPUs y RAM que pueden ser testeadas se transformara en la tarea principal. De ahí que existe la posibilidad de optimizar la tarea y hacer uso del paralelismo en OMP.

Metodo:

Uno de los métodos de paralelismo I/O que se descartan es el disk stripping, puesto que el sistema de archivos debe ser conocido, donde hay más de un disco donde distribuir la lectura o escritura de archivos.

El siguiente método consiste en leer el archivo por `chunks`, especificando un buffer y procesando cada paquete de datos sin saturar la memoria. Para esto es necesario conocer la estructura del archivo a leer, y asumiendo que el formato dado está separado por un salto de línea, u alguna otra estructura es cómo se implementará la lectura y conversión del texto.

Una vez el archivo se encuentre procesado en un `array`, el siguiente paso consistirá en agrupar las fechas y horas para conocer cuales son los horarios de mayor tráfico de usuarios. Esta parte es la que necesitará hacer uso del paralelismo y ver si se obtiene ventaja al procesar los datos de esta manera en contraste con un método secuencial.

Debido a la naturaleza del problema, el programa estará en un esqueleto de OMP con el proceso maestro orquestando los procesos esclavos y entrando en acción al momento indicado, esto es, una vez realizada la lectura del archivo.log.

Recursos:

1. https://uh-ir.tdl.org/bitstream/handle/10657/468/MEHTA-DISSERTATION-2013.pdf?sequence=1&isAllowed=y
2. https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture32.pdf
3. https://itqna.net/questions/85137/how-loop-openmp-count-rows-text-file
4. https://www.cpu-world.com/CPUs/Core_i5/Intel-Core%20i5-4440.html
5. https://www.youtube.com/watch?v=5wzmEKjNqiU
6. https://en.wikipedia.org/wiki/Multi-core_processor
7. https://www.howtogeek.com/194756/cpu-basics-multiple-cpus-cores-and-hyper-threading-explained/
8. https://en.wikipedia.org/wiki/C%2B%2B/CLI