Secretos del generador de perfiles de ejecución

Sebastián Argüello
Universidad de Costa Rica, Escuela de Ciencias de la Computación e Informática,
San José Costa Rica
sebastian.arguello@gmail.com

Enrique Valverde Chacón
Universidad de Costa Rica, Escuela de Ciencias de la Computación e Informática,
San José Costa Rica
evc.vlpv@gmail.com

Resumen:

Durante el proceso de implementación o mantenimiento de programas informáticos, una de las inquietudes más frecuentes por parte de un buen programador; es la de optimizar el tiempo que tarde la aplicación durante su ejecución. Por lo que se propone el estudio de una herramienta muy importante que facilita esta tarea; como lo es el *generador de perfiles de ejecución*.

Los *analizadores de rendimiento* como también se les conoce (denominados comúnmente ***profilers*** en inglés), son herramientas que permiten analizar el comportamiento de programas en tiempo de ejecución. Este análisis permite observar cuales son las partes del programa que consumen más tiempo de ejecución, y así saber cuales zonas son criticas para optimizarlas.

Este tipo de herramientas permiten crear programas más eficientes a raíz de aspectos como lo son: el descubrir partes no utilizadas del código fuente, cuales son las secciones de la aplicación utilizadas con mayor frecuencia, en que funciones se invierte la mayor parte del tiempo de computo en la CPU; entre otras.

Dentro de estas herramientas destacan ***gprof*** y ***gcov***, las cuales se pueden acceder mediante el compilador ***gcc***; el cual sumado a su gran potencia y versatilidad destaca por ser software de código abierto lo que lo hace totalmente gratuito. Como también están sus versiones de software propietario de compañías como Borland con su ***Turbo Profiler***, o bien ***Source Profiler*** de Microsoft.

Palabras Clave: perfil de ejecución, gprof, gcov, perfil plano, grafo de llamadas.

1.Introducción

Procesadores de Lenguaje es el término general con el que se le conoce a una gran variedad de programas informáticos, de

los cuales una de sus características principales es que su entrada corresponde a un lenguaje. Este concepto envuelve en su definición a una amplia gama de programas, dentro de los cuales se debe destacar a los g*eneradores de perfiles de ejecución* también conocidos como *analizadores de rendimiento*.

El análisis de rendimiento es la investigación del comportamiento de un programa usando información recopilada en tiempo de ejecución del programa. Esto se hace, usualmente, con el objetivo de determinar cuales partes del programa se deben optimizar para aumentar la velocidad u optimizar el uso de memoria.

Lo anterior permite asegurar que se están optimizando secciones del código que suponen un gasto importante de recursos, asegurando que se esté trabajando en las partes del programa, que al optimizarlas, suponen una mayor ganancia. Esto gracias a que los *profilers* permiten recoger estadísticas sobre el comportamiento de un determinado programa durante su tiempo de ejecución. De igual forma pueden ser reutilizados por el propio compilador para la optimización a la hora de generar código intermedio.

La forma en que estas herramientas permiten al usuario conocer este tipo de comportamiento es su aplicación es mediante a la generación de un fichero externo con información detallada del perfil de su programa; como lo es el caso de gmon.out que se analiza con *gprof*. Entre información que se puede obtener mediante este tipo de perfilado de programas destacan datos como lo son la distribución del tiempo de ejecución a lo largo de programa, en detalle del tiempo de ejecución de sus funciones, numero de llamadas a cada instrucción o función, entre otras.

Además hay que diferenciar que dependiendo de la herramienta, y/o de los parámetros que se le den, este fichero tendrá un flujo con los eventos grabados (rastro o grafo de llamada) o un resumen estadístico de los eventos observados (perfil plano) Para hacer esto los perfiladores utilizan una gran variedad de técnicas para recolectar información, incluyendo interrupciones de hardware, instrumentación de código y contadores de rendimiento. Esta diferencia crea los dos grandes grupos de profilers: los llamados perfiladores planos (Flat profiler) y los Perfiladores con Grafo de llamadas (Call-Graph profiler) ([1])

La generación de un perfil es proporcional a tamaño del código, mientras que la de un rastro es proporcional al tiempo de ejecución del programa. Haciendo ineficiente generar rastros de programa secuenciales y necesarios para comprender la relación de los problemas de rendimiento en un programa paralelo (la espera de mensajes o problemas de sincronización) ([1])

2. Perfiles de ejecución

Los tres pasos importantes para la utilización de un *generador de perfiles de ejecucion* son:

1. Compilación
2. Ejecución
3. Análisis

pasos en los que se profundiza a continuación.

2.1Compilación:

La compilación usando *gcc* en Linux para la generación de *perfiles de ejecución* es muy sencilla y se logra obtener simplemente agregando el parámetro de compilación pg (o variaciones del mismo parámetro según en tipo de perfil que se desee generar) como comando en consola para la utilización de gprof. De igual forma se pueden utilizar los parámetros *-fprofile-arcs* y *-ftest-coverage* para realizar un análisis todavía mas exhaustivo con *gcov*. Esto añade en el programa código que permite guardar la información necesaria con respecto al perfil del programa. Al igual que *gcov* y *gprof* existen gran numero más de *profilers*.

Ejemplo de utilización de gprof en consola de Linux:
gcc -xpg nombre_del_programa.cpp

En windows ya sea si se quiere utilizar Borland como los compiladores de Microsoft, se debe de especificar al compilador que se quiere hacer usa de esta herramienta accediendo alas opciones de compilación en alguna de las ventanas en la interfaz gráfica del editor de texto que utiliza el compilador, y posteriormente acceder el llamado especifico al profiler.

De igual forma encontramos *profilers* en casi la totalidad de compiladores de la actualidad como lo es en el caso de Perl, JavaScript, Python, Java

2.2 Ejecución:

Los programas se compilan como se hace usualmente, sin embargo existen dos grandes diferencias en este proceso. El primer cambio se ve reflejado en el tiempo que tarda el programa en ejecutar, ya que este tiende a tardar aproximadamente un 30% más en su ejecución debido a la gran cantidad de instrucciones *linkeadas* al programa original con el fin de poder crear el perfil del programa. La segunda diferencia y menos sutil que la primera es el directorio que se crea después de la ejecución del programa, la carpeta *gmon.out* para el caso en que se utilice *gprof* o ficheros con extensiones *.bb* o *.bbg* para el caso en que se use gcov; de igual forma se crearán archivos específicos según cada profiler de cada compilador.

2.3 Análisis:

Después de la ejecución del profiler se debe interpretar la información solicitada acerca del perfil del programa. Lo cual se logra ejecutando los archivos que se encuentran en los ficheros generados anteriormente mediante el uso del *profiler* que hayamos seleccionado.

Específicamente hablando de la salida producida por *gprof* se compone de tres componentes. El primero conocido como *perfil plano* en el cual encontramos la información del tiempo consumido por cada función. Como segunda salida encontramos al *grafo de llamadas*, el cual nos muestra el análisis del tiempo de ejecución de cada función desde su invocación inicial hasta que culmine su todas sus instrucciones, tiempo durante el cual estuvo el proceso en ejecución. Este análisis toma en cuenta que funciones fueron las que realizaron las invocación, al igual que funciones fueron invocadas desde esta. Estas dos primeras salidas se detallaran más adelante. Y como tercer salida aparece un listado alfabético de las funciones numeradas, para poder comprender mejor el *grafo de llamadas*.

i. *Perfil plano*:

El *perfil plano* es la primera parte de la salida del *gprof*. El cual muestra información del tiempo consumido por cada función sin entrar en detalle en el *grafo de llamadas*. En algunos de los casos basta con el *perfil plano* para hacerse una idea en que segmentos de nuestro programa se comienzan a originar problemas. Esta salida es una tabla en la que se se muestra una lista de rutinas ordenadas de mayor a menor según el consumo de la CPU. Esta tabla tiene las siguientes columnas e información:

- **%time**: es un porcentaje del tiempo total consumido en la función, sin tomar en cuenta los llamados hechos a funciones externas.
- **cumulative seconds**: es el acumulado del tiempo total en segundos de esta funcion sumado con las funciones que aparecen listadas en la tabla antes de ella. No incluye el llamado a otras funciones.
- **self seconds**: es el tiempo en segundos consumidos por la función, sin tomar en cuenta llamados a otras funciones.
- **calls**: numero de veces que se llama a la función.
- **self us/ call**: tiempo promedio en microsegundos consumido en llamados a la función. Sin tomar en cuenta llamados a otras funciones.
- **total us/ call**: tiempo promedio en microsegundos consumido en llamados a la función. Tomando en cuenta llamados a otras funciones.
- **name**: nombre de la función.

  De estar interesado únicamente en el *perfil plano* se debe de ejecutar el programa *gprof* únicamente con el parámetro *-p*.

ii. *Grafo de llamadas*:

Existen casos en los que el uso del *perfil plano* no nos muestra toda la información que necesitamos conocer para poder realizar las mejoras que buscamos en nuestro código, en estos casos *gprof* nos presenta una *grafo de llamadas* en el cual podemos saber que funciones fueron invocadas por determinada función, o bien conocer que funciones son las responsables del llamado a una función. O de igual forma puede ser relevante conocer el tiempo que tardan en realizarse las llamadas a otras funciones. Esta información se nos muestra nuevamente en forma de tablas, únicamente que estas son mas detalladas y se despliega una tabla completa por función con la siguiente información en cada columna:

- **index**: es el indice que se le asigno a la función. Por encima de ella se encuentran las funciones que hacen llamados a la función y por debajo de esta se encuentran las que son llamadas por ella.
- **%time**: es el porcentaje del tiempo de ejecución que utilizo esta función, junto con los llamados a funciones

desde la misma.

- **self**: es el tiempo en segundos consumidos por la función, sin tomar en cuenta llamados a otras funciones.
- **children**: es el tiempo en segundos consumidos por las funciones que son llamadas desde esta función.
- **called**: indica el numero de veces que otras funciones llamaron a la función que se analiza en la tabla, para el caso de las filas superiores. Para el caso de la función actual se muestra la cantidad de veces que ha sido llamada. Y para las filas inferiores muestra la cantidad de veces que se han llamado desde la función actual.
- **name**: nombre de la función junto con su indice el cual se le fue asignado.

Cabe tomar en cuenta que conforme se ejecute gprof se van acumulando archivos e información, por lo que es bueno estar borrando salidas anteriores si se le realizan cambios al código.

Hablemos ahora un poco de salidas que se nos brinda con una herramienta como *gcov*. Si lo que se busca es un análisis más profundo instrucción por instrucción esta es la herramienta a elegir. El análisis de cobertura como se le conoce, nos brinda la oportunidad de conocer el numero de veces que se ejecuta cada linea con lo que conocemos que instrucciones de nuestro código son usadas con menor frecuencia o si es el caso no son utilizadas del todo. Al ejecutar el gcov se nos genera un archivo de salida matriz.c.gcov en el cual se ve cada linea del programa junto con el numero de veces que se ejecuto al lado suyo, o bien un ###### lo que indica que una sección nunca se ejecutó.

3. Instrumentación:

También es posible hacer que un programa se convierta en su propia herramienta de análisis. Es decir un programa se analice a si mismo. Esto se logra insertando, en tiempo de compilación, código dentro del programa a ser analizado. Está técnica se conoce como "instrumentación". Esta técnica tiene diversas aproximaciones entre las cuales tenemos:

1. Manual: el código es introducido por el programador
2. Asistido por el compilador. Como por ejemplo el gcc
3. Inyección en tiempo de ejecución: el código es modificado en tiempo de e ejecución para crear brincos a funciones auxiliares.

4. Implementación de un generador de perfiles de ejecución

Un profiler, como toda aplicación, puede implementarse de maneras muy diferentes y especificas que usualmente producen programas muy distintos. Además, como ya se dijo existen distintos tipo de profilers (los de grafo de llamadas y los planos) y obviamente la implementación es distinta para cada tipo. Por esto se describirán brevemente dos implementaciones para un profiler.

Para implementar un profiler basado en estadística básicamente se debe manejar el hilo o proceso a optimizar. Esto implica tomar muestras, es decir ver la información que referente al contexto del hilo y el estado del procesador, a intervalos definidos.

La cantidad de muestras es variable y depende de que tan exhaustivo se quiera hacer el análisis de la aplicación, esto ya que entre mayor sea el número de muestras se puede lograr mejores resultados. Las muestras se logran interrumpiendo la ejecución del proceso, durmiendolo o pausandolo y almacenando la información acerca del estado del mismo para ese momento. Luego se permite que el proceso se siga ejecutando durante un intervalo (por ejemplo 1 milisegundo) y se vuelve a tomar una muestra (figura 1).

Se pausa el programa
Se recopilan los datos
Se reanuda la ejecución
Sigue ejecutando el programa
Proceso en ejecución

Figura 1. Representación del proceso de toma de muestras de un generador de perfiles de ejecución

La otra aproximación sería que al final de cada instrucción se incremente el contador para cada línea. No es el análisis de las líneas de un programa, sino que se debe hacer un listado de las instrucciones e insertar la instrucción que incrementa el contador. Durante el proceso se deben ignorar los comentarios, líneas en blanco y directivas de cada lenguaje que no son instrucciones. Esto se logra en insertando la instrucción que incrementa el contador en el código fuente del programa en tiempo de compilación. La manera usual de hacerlo es insertando una instrucción para llevar la cuenta justo después de cada línea(Figura 2), sin importar que haya más de una instrucción del programa en cada línea. Esto incrementa considerablemente el tiempo de compilación y también, pero a menor escala, el tiempo de ejecución del programa. Pero a cambio el resultado obtenido es estadísticamente perfecto (esto porque no se están tomando muestras del programa, sino que se esta contando cada instrucción) Por ejemplo:

```
//Descripción de la función
void funcionDePrueba(){
            incrementarContador(linea_actual);
            int c = 0;
            incrementarContador(linea_actual);
            for(int z=0; z<100; ++z){
            //Comentario [debe ignorado por el profiler]
            incrementarContador(linea_actual);
                c += z;
            }
}
```

Donde las líneas en negrita representan el código insertado por de un generador de perfiles de ejecución

Los resultados para la primera implementación dependen de la cantidad de muestras y el tiempo que se de al programa para tomar la siguiente muestra. Pero si se hace un muestreo amplio genera perfiles del programa que brindan información muy precisa, lo suficiente como para tener la certeza de que se estarán encontrando cual es el uso promedio de las funciones del programa. La segunda implementación permite esto pero no sólo en promedio, sino que genera, como ya se dijo, un perfil estadísticamente perfecto con el conteo de ejecución de cada función y línea del programa. Por esto es que lo generadores de perfiles de ejecución funcionan sin importar su implementación.
Nada más cabe destacar que se debe que tener en claro que tipo de perfil es el más conveniente según el tipo de programa que se quiere analizar y también el enfoque que se quiere tener de dicho análisis. Es decir que tan certera debe ser la información del perfil. A la hora de seleccionar la herramienta que se utilizará para generar el perfil.

Conclusión:

Los generadores de perfiles de ejecución son parte de la tecnología de las herramientas de análisis de rendimiento, que, como se dijo, data desde los años 70. Por lo tanto no es una tecnología muy novedosa pero, a pesar de su potencial, es una tecnología no muy utilizada.

En cuanto a la implementación de este tipo de procesadores de lenguaje, las posibilidades son muy grandes, sin embargo se ve cómo el correcto manejo de los hilos y su estudio, es el tema de cuidado.

Depende en gran parte el lenguaje de implementación, ya que son estos mismo los que nos van a proporcionar las herramientas que nos permitan realizar los estudios de los tiempos de uso de la CPU al igual que los ciclos del reloj, a través de sus librerías. De igual forma las herramientas para pausar o suspender la ejecución de ciertos bloques de código.

Por estas razones se puede ver a un profiler como un programa con una dificultad de implementación media o alta, y para el cual se deben de dominar gran cantidad de conceptos. Sin embargo su uso mediante a herramientas ya disponibles es muy sencillo.

Referencias

- Biblioteca en línea Wikipedia, http://www.wikipedia.org, 2008
- N, Chapman (2005) *Implementación de un profiler simple*
- A, Pyster (1980) *Compiler design and constructive*