Influencia de las Imágenes en la Eficiencia de los Contenedores

Universidad Politécnica de Valencia
Máster Universitario en Computación en la Nube y de Altas Prestaciones
Asignatura Cloud Computing

Autor:
Juan González Caminero

Contenidos

Lista de Tablas

1 Abstract

Los contenedores Linux son la tecnología de virtualización ligera más usada en la actualidad. Nos permiten ejecutar grupos de procesos aislados del sistema host y del resto de contenedores, y limitar su consumo de recursos.
Docker [1] es un sistema de gestión de contenedores, entre otras cosas, define un estándar de imágenes para los contenedores [2]. Estas imágenes nos permiten definir el contenido del sistema de archivos del contenedor, y lanzar múltiples instancias del mismo.

Docker emplea en los contenedores sistemas de ficheros de tipo union, ofreciendo varias implementaciones [3]. Las imágenes se componen de una serie de "capas", cada una de las cuales contiene una serie de ficheros, el UnionFS nos presenta estas capas como un único sistema de ficheros. Las capas de una imágen son de sólo lectura, y pueden ser usadas por varios contenedores al mismo tiempo, Docker proporciona a los contenedores una capa de lectura/escritura que pueden utilizar como almacenamiento persistente. También se proporciona la opción de compartir zonas del sistema de ficheros del host con los contenedores.

El objetivo de este trabajo es determinar cómo impacta el uso de este tipo de sistemas de ficheros al rendimiento de los contenedores, así como las diferencias de rendimiento en función del número de capas, el uso que se haga del sistema de ficheros, o si se usa el sistema de ficheros del host en lugar de la capa de lectura/escritura de Docker. También se explora la diferencia entre distintas implementaciones del UnionFS soportadas por Docker.

Se plantean una serie de experimentos que prueban los principales parámetros que se considera que pueden marcar una diferencia de rendimiento, y se ejecutan sobre distintas implementaciones del UnionFS. En vista de los resultados obtenidos, se llega a la conclusión de que en las implementaciones más modernas del UnionFS no se aprecia un impacto en el rendimiento al usar contenedores. En una de las implementaciones del sistema, sin embargo, sí podemos ver una diferencia a peor en el rendimiento.

2 Objetivos

El objetivo principal del proyecto es determinar el impacto del uso de sistemas de ficheros de tipo Union en el rendimiento de los contenedores, para ello, se plantean los siguientes objetivos:

- Determinar si existen diferencias entre el uso de la capa de lectura/escritura y el uso de un directorio compartido con el host.
  - Determinar, si existen diferencias, si el número de capas de la imágen tiene algún efecto.
- Determinar si la profundidad a la que se encuentre una capa tiene algún efecto en el rendimiento.
- Determinar si existe una diferencia de rendimiento entre escribir en un nuevo fichero en la capa de lectura/esritura y modificar un fichero preexistente.
- Identificar los principales parámetros que puedan causar una diferencia de rendimiento.
- Determinar si existen diferencias significativas entre las implementaciones del UnionFS que proporciona Docker.

3 Experimentación

3.1 Planificación

Se realiza toda la experimentación sobre las mismas imágenes de Docker, basadas en la imágen de Ubuntu y que añaden:

- Python3.
- 10 y 20 capas respectivamente, generadas usando el comando ADD.
- Ficheros utilizados en las pruebas, dentro de la primera y última de estas capas.

Las pruebas consisten en lanzar cada contenedor desde el host cinco veces, cada lanzamiento del contenedor ejecuta una serie de pruebas de lectura y escritura a través de un script python almacenado en un directorio compartido con el host, y almacena el tiempo empleado por cada prueba en ese mismo directorio compartido.

Se observó, en pruebas sobre el sistema host, que los datos en caché afectan a los resultados obtenidos en las pruebas. Por ejemplo, la primera lectura de un fichero lleva mucho más tiempo que las siguientes. Para eliminar este efecto, se vacían las cachés a través del fichero /proc/sys/vm/drop_caches entre cada ejecución de la batería de pruebas.

Después de cada lectura/escritura dentro del script de pruebas se hace una llamada a fsync, que fuerza a sincronizar el estado del fichero en caché con el estado en el dispositivo de almacenamiento, lo que asegura que los tiempos se miden cuando todas las operaciones han terminado.

Se plantean los siguientes experimentos para las pruebas:

- Lectura completa de un único fichero a memoria, usando ficheros de 1GB y de 100MB.
- Escritura de datos generados a través de /dev/urandom a un fichero nuevo, con tamaños de 1GB y 100MB.
- Escritura de datos generados a través de /dev/urandom a un fichero existente, con tamaños de 1GB y 100MB.

- Lectura completa de varios ficheros a memoria, usando diez ficheros de 100MB, para un total de 1GB.
- Escritura de datos generados a través de /dev/urandom a varios ficheros nuevos, diez ficheros de 100MB, para un total de 1GB.
- Escritura de datos generados a través de /dev/urandom a varios ficheros existentes, diez ficheros de 100MB, para un total de 1GB.

Estos experimentos se ejecutan, en cada lanzamiento del contenedor, sobre un directorio compartido con el host, y sobre la primera y última de las capas generadas para la prueba. Además de esto, se ejecuta la experimentación sobre el propio host para tener una serie de datos de control.

En cuanto a las distintas implementaciones del UnionFS que ofrece docker [3], se usan overlay2 [4], por ser la implementación por defecto en las versiones más recientes de docker, overlay, la primera versión de OverlayFS, y fuse-overlayfs [5], el driver recomendado para usar docker en modo rootless en algunos sistemas. Lo ideal sería probar también AUFS [6], ya que era el driver por defecto hasta la versión 18.06 de Docker, sin embargo el sistema operativo con el que se realizan las pruebas no ofrece soporte para este sistema de ficheros.

La experimentación planteada prueba los siguientes parámetros:

- Dependencia del rendimiento del tamaño escrito/leído, realizando operaciones de lectura/escritura sobre ficheros de 100MB y 1GB.
- Dependencia del rendimiento del número de ficheros utilizados, realizando operaciones de lectura/escritura sobre ficheros de 1GB, y 1GB repartido en 10 ficheros de 100MB.
- Diferencia entre el uso de un fichero nuevo y la modificación de uno preexistente, realizando las escrituras descritas anteriormente en ambos escenarios.
- Diferencia entre el uso de la capa de lectura/escritura y un directorio compartido con el host.
- Diferencia entre capas a distintos niveles de profundidad.
- Diferencia entre imágenes con distinto número de capas.
- Diferencia entre distintas implementaciones del UnionFS.

3.2 Entorno experimental

Las pruebas se realizan sobre una máquina con las siguientes características:

- CPU: Intel Core i5-4690 @ 3.50GHz
- RAM: 7798MB DDR3 @ 1333Mt/s
- Almacenamiento: Seagate ST1000DM003-1SB102 1TB @ 7200RPM
- Sistema de ficheros: ext4
- SO Host: Arch Linux, versión del kernel 5.4.66-1-lts

4 Resultados

A continuación se presentan en tablas los resultados obtenidos, en función de la implementación del UnionFS utilizada en cada caso. Para cada experimento y directorio sobre el que se actúa, aparecen la media de tiempos de las cinco ejecuciones realizadas, así como el intervalo de confianza al 95% para la media. Todos los tiempos aparecen en segundos.

4.1 Overlay2

En la tabla 1 vemos los resultados obtenidos con Overlay2, la implementación por defecto del UnionFS en las versiones más recientes de Docker. Utilizando estos datos, se llega a estas conclusiones:

- Dependencia del rendimiento del tamaño escrito/leído: No parece haber una diferencia significativa entre leer/escribir 100MB o 1GB, los tiempos de lectura/escritura permanecen alrededor de 10 segundos para todos los casos con 1GB, y alrededor de 1 segundo con 100MB.
- Dependencia del rendimiento del número de ficheros utilizados: En este caso, sí parece haber una cierta tendencia, donde leer/escribir 1GB es ligeramente más lento al usar 10 ficheros. Este efecto es de esperar, ya que cada vez que se cambia de fichero las cabezas del disco tienen que moverse a esa nueva zona.
- Diferencia entre el uso de un fichero nuevo y la modificación de uno preexistente: Vemos que con esta implementación del UnionFS no parece haber ninguna diferencia.
- Diferencia entre el uso de la capa de lectura/escritura y un directorio compartido con el host: De nuevo, no parece haber diferencia, tanto las medias como los intervalos de confianza son muy similares en todos los casos.
- Diferencia entre capas a distintos niveles de profundidad: Ya se usen 10 o 20 capas, los resultados son muy similares, no parece que la profundidad de las capas afecte al resultado.
- Diferencia entre imágenes con distinto número de capas: Tampoco parece haber diferencia en este caso, tanto en las lecturas/escrituras de un directorio compartido, como si se usan capas de la imágen.

4.2 Overlay

En la tabla 2 se muestran los resultados de Overlay, la anterior implementación del OverlayFS y, a priori, menos eficiente. Sin embargo, vemos que las conclusiones obtenidas con Overlay2 se mantienen también con esta implementación del UnionFS, incluso al comparar los tiempos al usar capas con los de un directorio compartido o con los del host. Tampoco se aprecia una diferencia clara entre el rendimiento de Overlay y Overlay2.

4.3 Fuse-overlayfs

Por último, la tabla 3 muestra los resultados para Fuse-overlayfs, en este caso sí que se aprecian mayores diferencias entre los distintos casos de prueba:

- Diferencia entre el uso de un fichero nuevo y la modificación de uno preexistente: En este caso, vemos grandes diferencias entre el uso de un directorio compartido y las capas de la imágen a la hora de modificar un fichero preexistente en el caso de la escritura de 10 ficheros de 100MB, donde los tiempos llegan a ser de casi el doble a la hora de modificar ficheros existentes. La diferencia es menor cuando se escriben ficheros de 1GB, y no parece haber diferencia con un fichero de 100MB.

- Diferencia entre el uso de la capa de lectura/escritura y un directorio compartido con el host: Vemos una clara diferencia entre los tiempos del directorio compartido y las capas para todas las escrituras, en el caso de las lecturas no parece haber diferencia.
- Diferencia entre capas a distintos niveles de profundidad/distinto número de capas: Aunque las lecturas de las capas son más lentas que con los otros sistemas, seguimos sin ver una diferencia de tiempos al variar la profundidad y el número de capas de la imágen.

4.4 Conclusiones

Con los resultados obtenidos, podemos decir que no parece haber diferencia entre las implementaciones del UnionFS Overlay y Overlay2, aunque, si el sistema donde se lanzan los contenedores soporta Overlay2 siempre será preferible usar el sistema más moderno, ya que Docker dejará de dar soporte a Overlay en un futuro.

Con Overlay y Overlay2 no se aprecia diferencia al variar parámetros relacionados con Docker como el uso de la capa de lectura escritura o un directorio compartido, el uso de más o menos capas, o la profundidad a la que se encuentren. Tampoco vemos diferencia entre escribir sobre un fichero nuevo, o modificar uno existente (Que implica copiarlo a la capa de lectura/escritura).

Fuse-overlayfs es el sistema en el que vemos una mayor diferencia entre usar las capas de la imágen y un directorio compartido, y entre escribir en un fichero nuevo y modificar uno existente. Vemos que el rendimiento empeora especialmente cuando estamos modificando varios ficheros, en lugar de un único fichero con el mismo tamaño.
Esta implementación es la recomendada [5] para ejecutar Docker en modo rootless en algunos sistemas operativos. El modo rootless nos permite ejecutar el daemon y los contenedores sin permisos de superusuario, lo cual puede mitigar algunas vulnerabilidades del sistema. En base a la diferencia clara con Overlay y Overlay2, parece recomendable usar un sistema operativo en el que se pueda utilizar el modo rootless con Overlay2.

Overlay2

10 Layers	Host	Bind Mount	Layer 0/10	Layer 9/10
Read 1GB	10.01 (9.77 - 10.26)	9.59 (9.52 - 9.66)	10 (9.96 - 10.03)	10.02 (10.01 - 10.02)
Write 1GB new	9.97 (9.81 - 10.13)	10.24 (9.96 - 10.52)	10.29 (10.13 - 10.45)	10.36 (10.14 - 10.57)
Write 1GB existing	9.96 (9.82 - 10.09)	10.28 (9.99 - 10.58)	10.4 (10.06 - 10.75)	10.37 (10.2 - 10.54)
Read 10*100MB	9.98 (9.68 - 10.27)	9.82 (9.76 - 9.88)	10.39 (10.17 - 10.62)	10.23 (10.21 - 10.24)
Write 10*100MB new	10.62 (10.44 - 10.79)	10.79 (10.58 - 10.99)	10.76 (10.53 - 10.98)	10.89 (10.69 - 11.09)
Write 10*100MB existing	10.51 (10.4 - 10.61)	10.81 (10.56 - 11.07)	10.81 (10.59 - 11.04)	10.88 (10.59 - 11.17)
Read 100MB	0.99 (0.96 - 1.03)	0.99 (0.97 - 1)	1.01 (1 - 1.02)	1.02 (1.01 - 1.04)
Write 100MB new	1.04 (1.02 - 1.06)	1.05 (1.02 - 1.08)	1.11 (1.07 - 1.15)	1.07 (1.04 - 1.11)
Write 100MB existing	1.07 (1.03 - 1.11)	1.08 (1.05 - 1.11)	1.08 (1.06 - 1.09)	1.08 (1.05 - 1.11)

20 Layers	Host	Bind Mount	Layer 0/20	Layer 19/20
Read 1GB	10.01 (9.77 - 10.26)	9.84 (9.7 - 9.97)	9.99 (9.97 - 10.02)	10.22 (10.21 - 10.23)
Write 1GB new	9.97 (9.81 - 10.13)	10.24 (9.76 - 10.71)	10.09 (9.79 - 10.38)	10.03 (9.62 - 10.44)
Write 1GB existing	9.96 (9.82 - 10.09)	10.23 (9.93 - 10.54)	10.05 (9.79 - 10.31)	10 (9.82 - 10.18)
Read 10*100MB	9.98 (9.68 - 10.27)	10.03 (9.82 - 10.25)	10.17 (10.14 - 10.2)	10.58 (10.54 - 10.62)
Write 10*100MB new	10.62 (10.44 - 10.79)	10.63 (10.34 - 10.93)	10.55 (10.34 - 10.76)	10.48 (10.3 - 10.66)
Write 10*100MB existing	10.51 (10.4 - 10.61)	10.73 (10.3 - 11.15)	10.5 (10.32 - 10.68)	10.64 (10.3 - 10.98)
Read 100MB	0.99 (0.96 - 1.03)	1.05 (0.97 - 1.14)	1.01 (1.01 - 1.01)	1.08 (1.05 - 1.1)
Write 100MB new	1.04 (1.02 - 1.06)	1.05 (1.02 - 1.07)	1.05 (1.03 - 1.06)	1.03 (1.02 - 1.04)
Write 100MB existing	1.07 (1.03 - 1.11)	1.04 (1.01 - 1.06)	1.02 (1.01 - 1.03)	1.05 (1.02 - 1.08)

Tabla 1: Tiempos de overlay2

Overlay

10 Layers	Host	Bind Mount	Layer 0/10	Layer 9/10
Read 1GB	9.79 (9.54 - 10.05)	9.93 (9.64 - 10.21)	9.66 (9.64 - 9.69)	9.62 (9.6 - 9.64)
Write 1GB new	10.56 (10.29 - 10.83)	10.27 (10.04 - 10.49)	10.19 (9.89 - 10.5)	10.33 (9.94 - 10.72)
Write 1GB existing	10.73 (10.43 - 11.04)	10.42 (10.1 - 10.74)	10.1 (9.84 - 10.37)	10.43 (10.19 - 10.67)
Read 10*100MB	10.09 (9.79 - 10.39)	10.15 (9.93 - 10.37)	10.01 (9.76 - 10.27)	9.9 (9.87 - 9.92)
Write 10*100MB new	10.85 (10.65 - 11.05)	10.92 (10.65 - 11.19)	10.64 (10.39 - 10.89)	10.9 (10.5 - 11.29)
Write 10*100MB existing	10.89 (10.72 - 11.07)	11.26 (10.86 - 11.65)	10.73 (10.43 - 11.03)	10.74 (10.45 - 11.03)
Read 100MB	0.99 (0.95 - 1.03)	1.01 (0.97 - 1.06)	1.02 (1.01 - 1.03)	0.99 (0.97 - 1)
Write 100MB new	1.07 (1.05 - 1.1)	1.09 (1.07 - 1.12)	1.13 (1.03 - 1.23)	1.06 (1.03 - 1.09)
Write 100MB existing	1.06 (1.02 - 1.1)	1.11 (1.04 - 1.19)	1.06 (1.04 - 1.09)	1.09 (1.04 - 1.14)

20 Layers	Host	Bind Mount	Layer 0/20	Layer 19/20
Read 1GB	9.79 (9.54 - 10.05)	9.82 (9.65 - 10)	9.64 (9.61 - 9.68)	10.16 (10.13 - 10.18)
Write 1GB new	10.56 (10.29 - 10.83)	10.52 (9.82 - 11.23)	10.5 (10.03 - 10.98)	10.33 (9.96 - 10.69)
Write 1GB existing	10.73 (10.43 - 11.04)	10.57 (10.22 - 10.92)	10.42 (9.95 - 10.88)	10.15 (9.84 - 10.47)
Read 10*100MB	10.09 (9.79 - 10.39)	10 (9.68 - 10.31)	9.86 (9.76 - 9.97)	10.29 (10.26 - 10.32)
Write 10*100MB new	10.85 (10.65 - 11.05)	10.91 (10.66 - 11.17)	10.72 (10.39 - 11.04)	10.91 (10.42 - 11.4)
Write 10*100MB existing	10.89 (10.72 - 11.07)	10.7 (10.42 - 10.99)	10.75 (10.42 - 11.09)	10.84 (10.38 - 11.3)
Read 100MB	0.99 (0.95 - 1.03)	1.02 (0.98 - 1.06)	1.02 (1.01 - 1.02)	1.03 (1 - 1.07)
Write 100MB new	1.07 (1.05 - 1.1)	1.03 (1.01 - 1.05)	1.06 (1.03 - 1.09)	1.06 (1.02 - 1.1)
Write 100MB existing	1.06 (1.02 - 1.1)	1.05 (0.98 - 1.13)	1.08 (1.05 - 1.12)	1.06 (1.02 - 1.11)

Tabla 2: Tiempos de overlay

fuse-overlayfs

10 Layers	Host	Bind Mount	Layer 0/10	Layer 9/10
Read 1GB	9.79 (9.54 - 10.05)	10.03 (9.75 - 10.3)	10.05 (9.96 - 10.14)	10.17 (10.16 - 10.19)
Write 1GB new	10.56 (10.29 - 10.83)	10.46 (10.02 - 10.9)	11.6 (10.87 - 12.34)	11.46 (10.54 - 12.38)
Write 1GB existing	10.73 (10.43 - 11.04)	10.38 (10.01 - 10.75)	12.33 (11.43 - 13.23)	12.28 (11.77 - 12.79)
Read 10*100MB	10.09 (9.79 - 10.39)	10.03 (9.73 - 10.32)	10.31 (10.22 - 10.39)	10.38 (10.35 - 10.4)
Write 10*100MB new	10.85 (10.65 - 11.05)	10.57 (10.33 - 10.8)	13.84 (13.69 - 14)	13.68 (13.3 - 14.05)
Write 10*100MB existing	10.89 (10.72 - 11.07)	10.61 (10.46 - 10.76)	23.9 (23.66 - 24.15)	24.32 (24.17 - 24.48)
Read 100MB	0.99 (0.95 - 1.03)	1.01 (0.97 - 1.05)	1.03 (1.03 - 1.03)	1.06 (1.05 - 1.06)
Write 100MB new	1.07 (1.05 - 1.1)	1.12 (1.02 - 1.22)	1.29 (1.25 - 1.33)	1.28 (1.25 - 1.31)
Write 100MB existing	1.06 (1.02 - 1.1)	1.07 (1.05 - 1.09)	1.38 (1.17 - 1.58)	1.28 (1.25 - 1.32)

20 Layers	Host	Bind Mount	Layer 0/20	Layer 19/20
Read 1GB	9.79 (9.54 - 10.05)	9.76 (9.55 - 9.97)	10.05 (10 - 10.1)	9.55 (9.52 - 9.58)
Write 1GB new	10.56 (10.29 - 10.83)	10.59 (10.1 - 11.09)	11.4 (11.19 - 11.62)	11.45 (11.34 - 11.57)
Write 1GB existing	10.73 (10.43 - 11.04)	10.38 (9.99 - 10.76)	12.18 (11.57 - 12.78)	12.01 (11.5 - 12.52)
Read 10*100MB	10.09 (9.79 - 10.39)	9.89 (9.62 - 10.17)	10.25 (10.23 - 10.27)	9.75 (9.72 - 9.78)
Write 10*100MB new	10.85 (10.65 - 11.05)	10.75 (10.46 - 11.04)	13.6 (13.21 - 14)	13.47 (13.17 - 13.76)
Write 10*100MB existing	10.89 (10.72 - 11.07)	10.96 (10.53 - 11.39)	24.02 (23.31 - 24.72)	23.77 (23.06 - 24.48)
Read 100MB	0.99 (0.95 - 1.03)	1.01 (0.99 - 1.04)	1.03 (1.03 - 1.04)	0.96 (0.96 - 0.97)
Write 100MB new	1.07 (1.05 - 1.1)	1.07 (1.02 - 1.11)	1.35 (1.3 - 1.39)	1.31 (1.25 - 1.36)
Write 100MB existing	1.06 (1.02 - 1.1)	1.19 (1.08 - 1.3)	1.33 (1.3 - 1.36)	1.3 (1.26 - 1.33)

Tabla 3: Tiempos de fuse-overlayfs

Referencias

[1] Docker docs. https://docs.docker.com/.

[2] Docker docs. Images and layers. https://docs.docker.com/storage/storagedriver/#images-and-layers.

[3] Docker docs. Storage drivers. https://docs.docker.com/storage/storagedriver/select-storage-driver/.

[4] Docker docs. OverlayFS. https://docs.docker.com/storage/storagedriver/overlayfs-driver/.

[5] Docker docs. Rootless mode. https://docs.docker.com/engine/security/rootless/.

[6] Docker docs. AUFS. https://docs.docker.com/storage/storagedriver/aufs-driver/.