VM HW3 Report

- 虛擬機 2023 年作業 3
- 姓名: 林俊佑
- 學號: R11922114

VM performance comparison different configurations

> You are responsible for tweaking the VM and KVM configurations, and answer the following questions about VM performance.

1. Compare VM performance of SMP VM (2 virtual cores) versus UP VM (1 virtual core)
2. Compare VM performance using transparent huge page versus regular page (enable transparent huge page on KVM)
3. Compare VM performance with vhost versus without vhost (toggle vhost using vhost=on/off in run-kvm.sh)

> Please make sure the number of vCPUs is strictly less than the number of pCPUs for each virtualizing layer. For example, a setup can be a 16-core x86 machine running QEMU that emulates a 4 vCPU ARM system for KVM host to run, with a 2 vcpu VM running within it (16 > 4 > 2). You should first provide the measurement results using different configurations, and explain what causes the performance difference you observe. You should use the same v5.15 kernel and config for compiling your KVM host and the guest kernel, except for the VHOST_NET configuration.

Compare VM performance of SMP VM (2 virtual cores) versus UP VM (1 virtual core)

此測試項目針對 CPU 的個數不同進行比較, 因此我針對這兩種虛擬機進行了下列兩種主要與 CPU 相關的跑分測試

1. hackbench
2. kernbench

其中 hackbench 的結果如下

```
# for 1 cpu
Running with 50*40 (== 2000) tasks.
Time: 120.622
# for 2 cpu
Running with 50*40 (== 2000) tasks.
Time: 57.819
```

可以看出來 2 cpu 的執行時間約為 1 cpu 的 50% 非常符合 2 倍 cpu, 2 倍算力的直覺。 但在此時我發現 kernel 有警告如下

```
[  247.919034] rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
```

看到 RCU 就知道是 scheduling 的部分有問題, 大膽猜測是 task 太多導致 cpu stall 為此, 我想要試著調整 task 數量, 讓這個警告不要發生

```
# for 1 cpu : ./hackbench 15 process 50
Running with 15*40 (== 600) tasks.
Time: 32.488
# for 2 cpu : ./hackbench 15 process 50
Running with 15*40 (== 600) tasks.
Time: 17.177
```

可以發現即使沒有 stall 還是約略一半的時間差異, 可以推測該類任務和 schedule 沒有太大關聯 且雖然直覺上是 tasks 是 IO bound, 好像和 cpu 數量沒有直接關係, 但實務上卻會因為 cpu 數量呈現加速的趨勢 我又做了 4 cpu 作為比較

```
# 4cpu
./hackbench 50 process 50
Running with 50*40 (== 2000) tasks.
Time: 42.383
```

發現雖然加速了但並不像 1/2 cpu 間, 近乎線性的加速, 這反而合理不少, 畢竟一定存在可以平行和不可平行的部分, 所以不可能無限線性的加速下去

---

kernbench 的結果如下

```
# 1cpu
Elapsed Time 5410.880000
User Time 3454.530000
System Time 690.540000
Percent CPU 76.000000
Context Switches 522004.000000
Sleeps 32792.000000
# 2cpu
Elapsed Time 2823.770000
User Time 3906.190000
System Time 792.380000
Percent CPU 166.000000
Context Switches 421765.000000
Sleeps 32201.000000
```

透過 Elapsed Time 可以得到近似 hackbench 的結果, 2 倍的算力, 約為 2 倍的速度提升。 此外可以觀察的是 Percent CPU 和 Context Switches 兩倍的 CPU 帶來約兩倍的 Percent CPU , 有足夠的 present CPU 可以減少

Context Switches , 減少 content switch 可以提高 User Time 的時間, 但這裡反直覺的點在於 system time 反而 2 cpu 更多。 我猜測可能是因為多核心系統多了不少 scheduling 的成本, 因此雖然整體減少了切換的成本, 但核心計算的時間卻會少許提升。

Compare VM performance using transparent huge page versus regular page

此測試項目針對記憶體管理配置不同進行比較, 我針對此項目做了完整四項測試

1. hackbench
2. kernbench
3. netperf
4. apache bench

其中前三項差異在幾 % 內, 可以視為誤差, 以下略過較有明顯差別的是 apache bench , 結果如下

```
# With THP
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   12  41.8      3    1053
Processing:     6  856 476.7    734    9292
Waiting:        3  691 362.7    610    4760
Total:          8  868 485.5    742    9293

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   14  56.4      3    1406
Processing:     5  853 453.8    757    7649
Waiting:        4  682 371.8    612    5091
Total:          8  867 470.0    766    7651

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   12  32.8      3     663
Processing:     4  819 400.9    732    6105
Waiting:        3  651 318.0    603    4457
Total:          6  830 407.8    740    6173
# No THP
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   13  40.2      3    1242
Processing:     6  926 466.8    813    9324
Waiting:        4  733 370.8    659    5274
Total:          8  938 473.9    822    9335

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   15  72.4      3    1836
Processing:     5  833 440.2    766    6933
Waiting:        4  651 355.5    601    4942
Total:          7  849 458.7    774    7030

Connection Times (ms)
```

```
               min  mean[+/-sd] median   max
Connect:         1   14  49.7      3    1204
Processing:      5  816 498.6    732    8857
Waiting:         3  529 386.7    488    8468
Total:           6  830 507.5    742    8858
```

可以發現 apache bench 在啟用 THP 下處理時間明顯可以在每次循環後有所降低,推測原因是 apache bench 的項目是獲取網頁靜態頁面, 該項目有大量時間是在等待磁盤讀取,所以有效的快取政策可以明確的提升速度, 此時 THP 可以提高 TLB hit 的可能性, 且每輪測試目標相同, 所以只要抓過一次的靜態頁面, 在次抓取的速度就能有所提升。此時 normal 的政策無法提供太好的快取效果的原因推測是因為, 雖然每輪的分頁重複, 但一個分頁很大, 一般分頁裝不下, 導致即便多輪存取, 快取效果依然不佳。

其餘測試則應該是因為與快取機制關係不大, 所以無明確的差異。

Compare VM performance with vhost versus without vhost

Vhost 為網路相關的測試, 所以我將聚焦在比較以下兩項測試

1. netperf
2. apache bench

以下列出 netperf 的結果(僅取具代表性的筆數)

```
# vhost off
MIGRATED TCP STREAM
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
131072  16384  16384    10.04     914.60
...skip
MIGRATED TCP REQUEST/RESPONSE
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec
16384  131072 1        1       10.01     413.50
16384  131072
# vhost on
MIGRATED TCP STREAM
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
131072  16384  16384    10.06     779.52
...skip
MIGRATED TCP REQUEST/RESPONSE
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec
16384  131072 1        1       10.01     616.62
16384  131072
```

可以發現啟用 vhost 時 TCP STREAM 的速度明顯較慢, 然而 TCP REQUEST/RESPONSE 的速度卻又明顯較快。 這樣的差異卻是反直覺的, 直觀上開啟了 vhost 應該都要有速度上的提升才對。 我覺得這個狀況應該要從 vring 開始理解, vhost 之所以可以提高傳輸流量的原因是在 kernel 中創建了額外模塊用來處理 IO 事件, 模塊和 guest VM 之間利用 vring 完成直接傳輸, 這個過程相對於不使用 vhost 減少了許多冗長的模式切換途徑。 我猜測使用 stream 時因為只要建立連線就可以持續傳輸, 所以在不啟用 vhost 下也沒有太多的模式切換, 使用了 vhost 反而新增了 vring 作為媒介, 整體反而更慢。 反而 REQUEST/RESPONSE 有明確的許多 IO 事件, 每次事件都要包含一組模式切換, 透過 vhost 可以大量的減少模式切換次數, 同時 vhost 前端的 virtio 利用 event driven 的 loop 作為驅動, 針對這種使用場景, 效能也會有所優化。整體來說, 流量自然更大。

接著比較 apache bench 的結果

```
# vhost on
...skip
Server Software:        Apache/2.4.41
Server Hostname:        192.168.0.105
Server Port:            80

Document Path:          /gcc/index.html
Document Length:        278 bytes

Concurrency Level:      100
Time taken for tests:   830.918 seconds
Complete requests:      100000
Failed requests:        0
Non-2xx responses:      100000
Total transferred:      45800000 bytes
HTML transferred:       27800000 bytes
Requests per second:    120.35 [#/sec] (mean)
Time per request:       830.918 [ms] (mean)
Time per request:       8.309 [ms] (mean, across all concurrent requests)
Transfer rate:          53.83 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   12  32.8      3     663
Processing:     4  819 400.9    732    6105
Waiting:        3  651 318.0    603    4457
Total:          6  830 407.8    740    6173

Percentage of the requests served within a certain time (ms)
  50%    740
  66%    860
  75%    942
  80%    997
  90%   1210
  95%   1570
  98%   2149
  99%   2548
 100%   6173 (longest request)
# vhost off
```

```
...skip
Server Software:        Apache/2.4.41
Server Hostname:        192.168.0.105
Server Port:            80

Document Path:          /gcc/index.html
Document Length:        278 bytes

Concurrency Level:      100
Time taken for tests:   658.589 seconds
Complete requests:      100000
Failed requests:        0
Non-2xx responses:      100000
Total transferred:      45800000 bytes
HTML transferred:       27800000 bytes
Requests per second:    151.84 [#/sec] (mean)
Time per request:       658.589 [ms] (mean)
Time per request:       6.586 [ms] (mean, across all concurrent requests)
Transfer rate:          67.91 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2   28  30.0     17     580
Processing:     6  630 495.0    519    7338
Waiting:        5  537 415.9    443    6512
Total:         13  657 495.1    544    7570

Percentage of the requests served within a certain time (ms)
  50%    544
  66%    686
  75%    791
  80%    867
  90%   1177
  95%   1533
  98%   2063
  99%   2576
 100%   7570 (longest request)
```

可以發現和上面結論不相符合的地方是未開啟 vhost 的 appach2 server 反而擁有更高的 Transfer rate, 理論上根據前一段論述 vhost 配合 TCP 應該要速度更快啊, 但實際看 apache bench 會發現這是一個壓力測試的工具, 考慮到這麼大量的流量平行進入, server 可以同時處理的併發數與平均性能或許才是關鍵。 因此我認為若要關注 vhost 對 IO 的影響, 應該先聚焦在各指標的分佈, 此時可以發現啟用 vhost 時, 最小值顯然更小, 可以知道, 最少在還不需要考慮其他條件時, 因為 vhost 可以節省一些流程, 所以可以得到最高的流量。 為了驗證想法, 我試著更改測試方法, 排除掉平行能力的影響, 想看看是否真的讓有 vhost 比沒 vhost 傳輸更快。 使用指令 `ab -n 1000 -c 1 http://ip.for.server/gcc/index.html` 一次只有一個 process 會打 API, 得到以下結果

```
# vhost on
...skip
Transfer rate:          40.19 [Kbytes/sec] received
```

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1    2   0.9      2      12
Processing:     5    8   4.6      7      40
Waiting:        3    6   3.9      5      36
Total:          7   11   4.7      9      44
# vhost off
...skip
Transfer rate:          34.66 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2    4   1.6      4      24
Processing:     4    8   4.1      7      47
Waiting:        3    7   3.9      6      44
Total:          6   12   4.6     11      52
```

可以明顯發現有使用 vhost 的 server 擁有更大的流量。 所以統整所有發現可以整合成以下兩點

1. 使用 stream 時無法有效發揮 vhost 的能力, 因為原本就不需要那麼多模式切換
2. 使用 req/res 時可以有效發揮 vhost 的能力, 但當平行度過大時, 若是整個系統沒做好設置, 反而會產生雍塞, 因此更慢

VM performance comparison with KVM host

以下先列出 KVM host 的數據

```
# hackbench
Running with 20*40 (== 800) tasks.
Time: 25.319
# kernbench
Elapsed Time 2823.770000
User Time 3906.190000
System Time 792.380000
Percent CPU 166.000000
Context Switches 421765.000000
Sleeps 32201.000000
# netperf (僅放具代表性數據)
Measuring netperf performance of 192.168.0.101
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.0.101 () port 0 AF_INET : demo
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
131072  16384  16384    10.02     213.18
...skip
MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET
to 192.168.0.101 () port 0 AF_INET : demo : first burst 0
Local /Remote
```

```
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

16384  131072 1        1       10.01     588.80
16384  131072
# apache bench (僅放具代表性數據)
Server Software:        Apache/2.4.41
Server Hostname:        192.168.0.101
Server Port:            80

Document Path:          /gcc/index.html
Document Length:        41288 bytes

Concurrency Level:      100
Time taken for tests:   1398.901 seconds
Complete requests:      100000
Failed requests:        0
Total transferred:      4156200000 bytes
HTML transferred:       4128800000 bytes
Requests per second:    71.48 [#/sec] (mean)
Time per request:       1398.901 [ms] (mean)
Time per request:       13.989 [ms] (mean, across all concurrent requests)
Transfer rate:          2901.41 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2  431 326.6    365    7610
Processing:    84  964 639.6    833    9460
Waiting:        2  447 351.6    380    7463
Total:        352 1395 838.3   1221   10317
```

以下是 guest VM 的數據

```
# hackbench
Running with 20*40 (== 800) tasks.
Time: 28.772
# kernbench
Elapsed Time 4288.510000
User Time 5276.660000
System Time 1278.660000
Percent CPU 152.000000
Context Switches 510560.000000
Sleeps 35847.000000
# netperf (僅放具代表性數據)
Measuring netperf performance of 192.168.0.105
MIGRATED TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.0.105 () port 0 AF_INET : demo
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec
```

```
131072  16384  16384    10.06     779.52
...skip
MIGRATED TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET
to 192.168.0.105 () port 0 AF_INET : demo : first burst 0
Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec
16384  131072 1        1       10.01     612.39
16384  131072
# apache bench (僅放具代表性數據)
Server Software:        Apache/2.4.41
Server Hostname:        192.168.0.105
Server Port:            80

Document Path:          /gcc/index.html
Document Length:        278 bytes

Concurrency Level:      100
Time taken for tests:   830.918 seconds
Complete requests:      100000
Failed requests:        0
Non-2xx responses:      100000
Total transferred:      45800000 bytes
HTML transferred:       27800000 bytes
Requests per second:    120.35 [#/sec] (mean)
Time per request:       830.918 [ms] (mean)
Time per request:       8.309 [ms] (mean, across all concurrent requests)
Transfer rate:          53.83 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   12  32.8      3     663
Processing:     4  819 400.9    732    6105
Waiting:        3  651 318.0    603    4457
Total:          6  830 407.8    740    6173
```

首先從 CPU bound 相關數據開始分析, hackbench 並沒有太大落差, 硬要說就是 KVM host 有略快一些, 我想畢竟 guest VM 是在 KVM host 中執行, 需要多一些轉換。

至於 kernbench 主要有落差的部分如下

1. Context Switches 在 guest VM 多 20%
2. 在 user / kernel mode 的執行時間 KVM host 等比例減少 執行速度相關的差異我想跟 hackbench 的原因相似, 而且從點 2 "兩個 mode 執行時間等比例減少" 可以發現一個特性, 即兩個 VM 都具有相似的 scheduling mechanism。差異僅在各 mode 的執行速度。 為了驗證以上想法, 我在兩個 VM 做了 unixbench 想知道各個項目的跑分

```
# KVM host
System Benchmarks Index Values               BASELINE       RESULT
INDEX
```

```
Dhrystone 2 using register variables         116700.0    4328921.7
370.9
Double-Precision Whetstone                       55.0        822.7
149.6
Execl Throughput                                 43.0         89.5
20.8
File Copy 1024 bufsize 2000 maxblocks          3960.0      24794.4
62.6
File Copy 256 bufsize 500 maxblocks            1655.0       6443.0
38.9
File Copy 4096 bufsize 8000 maxblocks          5800.0      94531.7
163.0
Pipe Throughput                               12440.0      29445.2
23.7
Pipe-based Context Switching                   4000.0       8192.6
20.5
Process Creation                                126.0        324.8
25.8
Shell Scripts (1 concurrent)                     42.4        304.5
71.8
Shell Scripts (8 concurrent)                      6.0         35.5
59.2
System Call Overhead                          15000.0      25021.7
16.7

========
System Benchmarks Index Score
52.2
# guest VM
System Benchmarks Index Values               BASELINE       RESULT
INDEX
Dhrystone 2 using register variables         116700.0    2606649.4
223.4
Double-Precision Whetstone                       55.0        630.1
114.6
Execl Throughput                                 43.0         23.6
5.5
File Copy 1024 bufsize 2000 maxblocks          3960.0     114696.5
289.6
File Copy 256 bufsize 500 maxblocks            1655.0      30677.4
185.4
File Copy 4096 bufsize 8000 maxblocks          5800.0     310816.3
535.9
Pipe Throughput                               12440.0     172081.1
138.3
Pipe-based Context Switching                   4000.0       6675.3
16.7
Process Creation                                126.0        144.6
11.5
Shell Scripts (1 concurrent)                     42.4        123.4
29.1
Shell Scripts (8 concurrent)                      6.0         13.9
23.1
System Call Overhead                          15000.0      90394.5
```

```
60.3

========
System Benchmarks Index Score
63.5
```

此時如果單看跑分, 會得出和前兩個 bench 相反的結果, guest VM 擁有更高的分數, 但如果詳細的依據項目解釋就會合理了

- 前三項, 計算類項目, KVM host 有明顯優勢, 可以猜測透過一層 VM 調用 CPU 確實存在性能流失
- File Copy 類的項目反而 guest VM 大幅領先, 我猜測是因為 host VM 的 `qemu-system-aarch64` 是使用軟體模擬硬體環境, 所以在該項目上沒有辦法調用硬體特性所以產生 guest 比 host 更快的現象

總合來說, 從 hackbench 和 kernbench 可以看出 host KVM 在 CPU 的運算速度, 併行的 overhead 上都有所領先, 相對的 guest VM 有對應的性能流失, 同時在 context switch 的策略上兩者沒什麼差異, 同時跟硬體支援有關的任務, guest VM 有可能更快

接著觀察網路相關的 bench

從 netperf 可以看到兩個現象

1. guest 的 stream 遠快於 host 的 stream
2. host 的 TCP req/res 略小於 guest 的 TCP req/res

表格如下

	guest	**host**
TCP req/res	略快	略慢
stream	快	慢

可以做出以下推測, TCP req/res 在 guest 略快是因為可以使用到 vhost 的特性, 但 stream 上超大的速度差異很明顯就不是一般的軟硬體機制可以解釋的, 聯想到 unixbench 上 guest 的 COPY FILE 速度遠遠快於 host 我做了一下猜測。

netperf 主要考慮的是 IO bound 的任務, 所以速度的差異來自於傳輸過程, 其中 guest VM 可以使用 vhost 減少模式切換, 路徑較短, 速度會有些微提升, 但是 stream 時限制速度的重點在把大量的數據搬遷 COPY 來/回 user space, 此時 host VM 或許是因為軟體模擬的缺陷, 效果很差, 所以速度遠慢於 guest VM。

接著討論 apache bench

可以發現 guest VM 的 connection time 明顯更低, 這我猜測是來自 http request/response 被 vhost 加速後的結果。 為了驗證結果, 額外檢視了未開啟 vhost 的 guest VM 和 host VM 的 apache bench 比較, 發現 connection time 雖然還是 guest 較快, 但也已經相對接近。

因此最終通整結論如下

1. 在 CPU 運算方面, guest 因為運行在模擬硬體中, 所以有相對明顯的效能損失
2. 在 COPY 類型指令部分, 或許是由於 host VM 軟體模擬上的缺陷, 所以 guest VM 只要牽涉到 COPY 指令的操作都會快很多。 ex: stream 相關
3. 在 TCP req/res 因為 guest 可以調用 vhost 這樣的特性, 所以有一定的速度提升。