mmap によるコピーの実行速度

235718G　新里　伊武輝

2025 年 2 月 3 日

1　はじめに

mmap を使ったコピーが、通常の read/write と比べてどのくらいの速度差があるかを検証する。なお、コピー元のファイルのサイズは 3GB とした。ベンチマーク結果は表 1 に示す。

2　3GB ファイルコピーのベンチマーク結果

方法	real (s)	user (s)	sys (s)
2 回連続実行			
cp (1 回目)	4.77	0.00	0.90
cp (2 回目)	4.97	0.00	0.94
mmap_copy (1 回目)	16.05	0.38	2.18
mmap_copy (2 回目)	27.98	0.42	3.19
mmap_write (1 回目)	18.57	0.03	1.50
mmap_write (2 回目)	13.00	0.03	1.40
read_write (1 回目)	4.69	0.02	0.91
read_write (2 回目)	4.65	0.03	0.90
1 回ずつ実行			
cp (1 回目)	4.57	0.00	0.92
cp (2 回目)	4.34	0.00	0.89
mmap_copy (1 回目)	12.96	0.38	2.45
mmap_copy (2 回目)	12.37	0.37	3.50
mmap_write (1 回目)	12.81	0.02	1.33
mmap_write (2 回目)	12.80	0.02	1.34
read_write (1 回目)	4.79	0.03	0.90
read_write (2 回目)	4.61	0.03	0.90

表 1　3GB ファイルコピーの実行時間 (real, user, sys)

3　考察

表 1 から、mmap_copy と mmap_write は cp や read/write よりも処理時間が長くかかることがわかる。これらの違いは主にデータの書き出し方法に起因する。cp と read/write はシステムコールである read および write を使用し、バッファを介してディスク I/O を行う。一方、

mmap_copy と mmap_write はディスク上のファイルを仮想メモリにマッピングし、メモリアクセス時にページフォールトが発生して物理メモリにロードされ、仮想メモリアドレスを介して物理メモリにアクセスする。これにより、mmap を使用する場合、ページフォールトによるオーバーヘッドが発生するため、cp や read/write よりも時間がかかることがある。

また、表 1 から、mmap_copy はコピー先を削除せずに連続してコピーすると処理性能が低下するが、mmap_write は逆に性能が向上する。この違いは、mmap_copy ではファイルキャッシュが更新されず、ページフォールトが増加するためである。一方、mmap_write は同じマッピング先に対して再利用されるキャッシュの効果で性能が向上すると考えられる。さらに、コピー先を削除して再びコピーすると、mmap_copy はキャッシュがクリアされることで性能が向上するが、mmap_write はキャッシュの再利用が少なくなり、性能変化が小さい。

これらの結果から、mmap_copy はコピー先が異なる場合に効果的であり、mmap_write は同じコピー先に対して繰り返し操作を行う場合に適していると考えられる。また、madvise を適切に設定することで、ページキャッシュの最適化が可能となり、さらに効率的なファイル操作が実現できる。

3.1 iostat ログを活用した分析

iostat1.log（2 回連続実行）と iostat2.log（一回ずつ実行）から、cp と read/write は 1 回目と 2 回目の変化はそれほどなかったのに対して、mmap_copy と mmap_write では 1 回ずつ実行よりも 2 回連続実行の方が 2 回目の処理性能が上がっていることがわかる。このことから、mmap_copy と mmap_write はキャッシュが効きやすいので、何度も同じデータにアクセスすることに適しており、一方で、cp と read/write はランダムアクセスに適していることがわかる。また、mmap_copy ではランダムアクセスの場合、ページフォールトをより頻繁に発生させてオーバーヘッドを招く可能性があるので、大規模ファイルのランダムアクセスには mmap_write、同じ大規模ファイルのアクセスには mmap_copy が適していると考える。

3.2 iostat の具体的な数値例

具体的な iostat ログにおいて、cp と read/write の 1 回目と 2 回目の I/O 待機時間（await）に大きな差が見られなかったのに対して、mmap_copy と mmap_write は 2 回目の実行で I/O 待機時間の変化が顕著に現れた。特に、mmap_write の場合、I/O 待機時間の減少が確認でき、キャッシュの再利用が効率的に行われていることが示唆された。一方で、mmap_copy ではページフォールトが頻繁に発生し、I/O 待機時間が増加する傾向が見られた。

これらの結果から、mmap_write はキャッシュ再利用が効率的であり、同じデータに対する繰り返しアクセスに向いていることが確認できた。また、mmap_copy はランダムアクセスにおいて、ページフォールトの影響を受けやすいため、アクセスパターンによってはオーバーヘッドが発生する可能性が高いことがわかる。

3.3 madvise の活用

madvise を適切に設定することで、ページキャッシュの最適化が可能となり、性能向上が期待できる。例えば、MADV_SEQUENTIAL や MADV_WILLNEED の設定を行うことで、メモリのアクセスパターンに基づいたキャッシュ管理が行われ、ページフォールトの頻度を減らすことができる。これにより、mmap_copy や mmap_write の性能をさらに向上させることが可能となる。