5J-04

シェルスクリプトにおける純粋なコマンドに対する入出力キャッシングによる増分計算の適用

佐藤碧[*1]　新美礼彦[*2]

公立はこだて未来大学大学院 システム情報科学研究科[*1]
公立はこだて未来大学 システム情報科学部[*2]

1　はじめに

近年，前処理としてシェルスクリプトを変形してから実行することで，シェルスクリプトに対して並列処理化やリモートアクセスの最適化といった処理を適用する研究が行われている．これらは，いずれもシェルスクリプトの処理速度を高速化させることを目的としている．そこで本研究では，別のアプローチとして，シェルスクリプトに増分計算を適用することで処理速度を高速化させることを目指す．シェルスクリプトの再実行あるいはパラメータを一部修正した上での再実行，同様のコマンドをほかのシェルスクリプト上で実行するなどの場合に，増分計算によって新規に計算が必要な部分のみを計算することで，シェルスクリプトの処理速度を高速化できると考える．

本研究では，シェルスクリプトを構成する各 UNIX コマンドのうち，シェルスクリプトにおける純粋なコマンドに対して入出力のキャッシュを使用した増分計算を適用する手法を提案する．ここでの純粋なコマンドとは，PaSh[1] で提案された UNIX コマンドの入出力に関する分類のうち，コマンドに対する入力データが同一なら出力も同一になる性質を持つ UNIX コマンドである．入出力のキャッシングにより，同一に入力に対して再計算をすることなく出力を返せる．この性質を利用することで，コマンドの処理内容に関係なく増分計算を適用可能である．また，シェルスクリプト全体に直接ではなく，それを構成する個々の UNIX コマンドに対して増分計算を適用することで，処理の文脈を維持しつつ増分計算を適用できる．

2　提案手法

本手法では，コマンドを実行する際に，コマンドラインとそれに対する入出力を特定して，その出力をキャッシュとして保存しておく．そして，次に UNIX コマンドおよびパイプラインを実行する際に，実行するコマンドラインと入出力を特定し，それらに対応するキャッシュに保存されていれば，そのコマンドラインを実行する代わりにキャッシュを出力する．同様の処理をシェルスクリプト内の各コマンドおよびパイプラインに適用することで，シェルスクリプト全体の再実行を高速化できる．

3　実装

PaSh[1] のソースコードを修正して提案手法を実装する．

本手法におけるキャッシュは，ファイルシステム上にファイルとして保存される．ファイルの内容は，対応するコマンドと入力の組み合わせによって得られる出力である．ファイルパスは，コマンドラインと入出力から算出されるハッシュ値で構成される．よって，キャッシュの探索は，そのファイルパスにファイルが存在するかを確認である．

PaSh は，入力となるスクリプトを解析して，スクリプト内の各コマンドやパイプラインを，スクリプトと相互変換が可能な内部表現に変換する．その際にパイプラインは，シェルの制御オペレータである|ではなく，UNIX の名前付きパイプとコマンドの非同期実行を指示するオペレータである&を使って，並列に実行されるコマンド群に変換される．よって，その内部表現では，UNIX コマンドのみを考慮すればよい．その内部表現からは，スクリプト内の各 UNIX コマンドごとに，コマンド名やコマンドライン引数といったコマンドライン，それに対する入出力先を取得できる．それらの情報からキャッシュの探索に必要となるハッシュ値を算出し，キャッシュが存在するか確認する．存在する場合，その部分を対応するキャッシュを読み出す処理で置換する．キャッシュの読み出しは cat コマンドで行う．存在しない場合，コマンドはそのまま実行するが，その際に出力を元の出力先ではなく一旦 tee コマンドに名前付きパイプ経由で流し，その tee コマンドが元の出力先に出力する．その際，tee コマンドはキャッシュ探索の際に利用したファイルパスを引数に指定して実行される．このようにコマンドの出力を tee コマンド経由で出力することで，その出力をファイルとしてキャッシュしながらコマンドを実行できる．

4　実験と評価

提案手法を実装した PaSh である CaSh によっていくつかシェルスクリプトを実行して処理時間を計測する．実行するシェルスクリプトは，増分計算の効果が比較的顕れやすいと思われる処理時間が長いコマンドを含んでいる．このとき，CaSh によって事前に対象とするシェルスクリプトを実行しておき，増分計算のための完全なキャッシュを作成しておく．これにより，次に CaSh によって対象とするシェルスクリプトを実行した場合，スクリプト内の全てのコマンドとその入

Apply Incremental Computation to Pure Commands on Shell Script by Input and Output Caching
[*1] Aoi Sato · Future University Hakodate
[*2] Ayahiko Niimi · Future University Hakodate

Copyright ©2023 Information Processing Society of Japan.
All Rights Reserved.

```
FROM=2015 TO=2015 IN='http://ndr.md/data/noaa/'
↪ fetch='curl -L -s'
seq $FROM $TO | sed "s;^;$IN;" | sed 's;$;/;' |
↪ xargs -r -n 1 $fetch | grep gz | tr -s ' n'
↪ | cut -d ' ' -f9 | sed
↪ 's;^(.*)(20[0-9][0-9]).gz;2/12.gz;' | sed
↪ "s;^;$IN;" | head -n144 | xargs -n1 $fetch |
↪ gunzip | cut -c 89-92 | grep -v 999 | sort
↪ -rn | head -n1
```

図 1　ネットワークアクセスが発生するパイプライン

```
grep -v '[1-6]' <random_number_100GB.txt | sed
↪ -E 's/^0+//g' | wc
```

図 2　ある程度巨大な入力を受け取るパイプライン

表 1　実験環境

OS	GNU/Linux (5.18.9-arch1-1)
CPU	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Memory	(16GB, LPDDR3)
SSD	WDS500G2B0B (500GB, SATA III)

出力についてキャッシュが作成されているため，全ての処理でキャッシュが有効になる．また比較対象として，対象とするシェルスクリプトをそのまま実行した場合の処理時間と，キャッシュがまったく存在しない場合に提案手法を実装した PaSh でシェルスクリプトを実行した場合の処理時間も計測する．前者は，シェルスクリプトを通常実行した場合と比較して，提案手法を適用することでどれだけ処理時間が変化するかを比較するためである．後者は，提案手法によってシェルスクリプト内の各コマンドについてキャッシュが作成される際に発生するオーバーヘッドがどの程度のものになるのかを確認するためである．実行時間は，それぞれの試行で 10 回ずつ実行して計測した時間の平均値同士で比較する．ディスクキャッシュの影響を回避するために，各試行前にディスクキャッシュをクリアしておく．実行するスクリプトは図 1 と図 2 の 2 種である．図 1 は，PaSh で使用された評価用のスクリプトを修正したものである．実行環境は表 1 である．

図 1 のスクリプトについての実験結果を表 2 に，図 2 のスクリプトについての実験結果を表 3 にそれぞれ示す．どちらの場合も，提案手法/キャッシュありで使用した場合の実行時間が，通常実行の場合と比較して，提案手法によって処理時間が高速化していることがわかる．またどちらの場合も，提案手法/キャッシュなしの場合の実行時間が，通常実行の場合と比較して，平均処理時間でほぼ差がないことがわかる．

5　関連研究

function caching[2] は，本手法と同様にキャッシュを使った増分計算の手法である．本手法は，コマンドに関する事前知識を利用して，シェルスクリプトに増分計算を適用したが，POSH[3] は，コマンドに関する事前知識を利用して，シェルスルクプトにリモートアクセスに関する最適化を適用する．

表 2　ネットワークアクセスが発生するパイプラインについての実験結果

条件	平均 (sec)
通常実行	413.099
提案手法/キャッシュあり	0.5104
提案手法/キャッシュなし	414.215

表 3　ある程度巨大な入力を受け取るパイプラインについての実験結果

条件	平均 (sec)
通常実行	327.739
提案手法/キャッシュあり	0.6079
提案手法/キャッシュなし	328.945

6　おわりに

今回実行した，増分計算の効果が比較的顕れやすいと思われる処理時間が長いコマンドを含んでいるシェルスクリプトについて，提案手法による増分計算によって処理速度を高速化できることが確認できた．また，提案手法のワーストケースである，キャッシュが一切存在しない場合に提案手法を適用した場合の処理時間についても，通常実行とほとんど差がないことを確認できた．

参考文献

[1] N. Vasilakis, K. Kallas, K. Mamouras, A. Benetopoulos, and L. Cvetković, “Pash: Light-touch data-parallel shell processing,” in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 49–66.

[2] W. Pugh and T. Teitelbaum, “Incremental computation via function caching,” in *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL ’89, Austin, Texas, USA: Association for Computing Machinery, 1989, pp. 315–328, ISBN: 0897912942. DOI: 10.1145/75277.75305. [Online]. Available: https://doi.org/10.1145/75277.75305.

[3] D. Raghavan, S. Fouladi, P. Levis, and M. Zaharia, “POSH: A Data-Aware shell,” in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, USENIX Association, Jul. 2020, pp. 617–631, ISBN: 978-1-939133-14-4. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/raghavan.

Copyright ©2023 Information Processing Society of Japan.
All Rights Reserved.