YyonX実行モデル(2) — Yy-clientの出力の解析 —

7E−2

古坂孝史[†1]，井田晶之[†2]，田中啓介[†2]
青山学院大学情報科学研究センター研究教育開発室

1 はじめに

Lispで構築されたYyonXのクライアントであるYy-clientでの入出力の応答時間は、利用者にとって重要な要素である。この応答時間を調べ、実行モデルに対して最適な機能分担を検討する必要がある。

この検討データを得るためYyonX version1.0 (Yy-client version1.0、Yy-server version1.0、Yy-protocol 1.1で動作している)を利用して実験を行なった。実験では、[1]で述べられているPage実行モデルと、Viewport実行モデルに対して、全ての処理をYy-clientで行った場合の処理時間を測定した。また、比較のためにYy-serverでの出力実験を行なった。

2 実験モデル

YyonXは、Yy-serverとYy-clientで構築されている。また、Yy-serverはXサーバと通信して、画面の制御を行なっている。従って、処理時間はYy-client、Yy-serverとXサーバの処理プロセスにおける処理時間の各々の合計となる。ここでは、この時間を計測した。

YyonXでは、連続して文字や図形を出力する場合、つまり、Yy-protocolでのコマンドだけが連続して実行されても、Yy-clientとYy-serverとの間に一定のタイミングで同期をとる間欠同期が導入されている。また、これとは独立してYy-serverは出力処理の途中でXサーバと同期をとっている。従って、出力処理を十分に繰り返せば、全てのプロセスが同期していると考えても差し支えない。そこで実験では、Yy-client側で同じ出力処理を繰り返した時の実行時間を計測した。一回あたりの処理時間は、その処理回数で割れば算出できる。

Yy-client中で、Yy-server側にコマンドを下す処理の流れは、大別すると2つの部分に分割できる。

1 ユーザの指示した描画やウィンドウ等の整形に対して、ウィンドウの選択、引き数部の準備や不足情報の生成等を行う純粋にLispだけで記述された部分(Lisp処理部)

2 Yy-protocolのパケットを生成し、Yy-serverにパケットを送り出すLispとC言語で記述された部分(通信処理部)

実際のYy-clientでは、例えば文字列をウィンドウに出力する場合でも上述で示した処理の流れより複雑である。そこで、時間計測に直接影響の無い部分をYy-clientから削除して、上述の1、2の処理に近付けた実験モデルを想定する。この実験モデルを図1に示す。図1の実験モデルを作り、各々の処理部の所要時間を計測することで、どこに処理の比重がかかっているか判る。また、Lisp処理部と通信処理部の実行時間を足し合わせることで、画面への出力時間が求まり、実行モデルの指標と比較可能である。

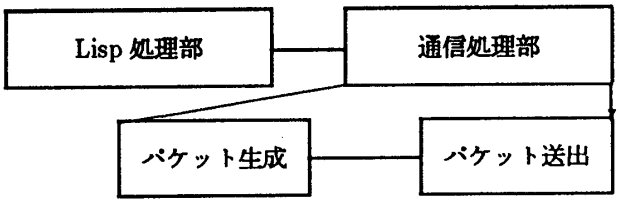

図1: 実験モデル

3 実験概要

実験は、4種類8項目を実施した。実験の目的と、実験方法を下記に記す。

- 実験1

目的: Page/Viewport実行モデルにおいて、Yy-clientで入力に対するエコー処理だけを行なった場合の実行時間を求める。

方法: 文字を一文字づつ出力する処理を100回繰り返し、一文字当りの処理時間を求める。

- 実験2

目的: Viewport実行モデルにおいて、Yy-clientで多量の描画を行なった場合の実行時間を求める。

方法: 文字列を一度に出力する処理を100回繰り返し、一文字当りの処理時間を求めた。この実験は、文字列が10,50,100,500,1000バイトである場合について、各々実施した。

- 実験3

目的: Page/Viewport実行モデルにおいて、Yy-clientで入力に対するエコーを含む一連の編集処理を行なった場合の実行時間を求める。

方法: 予め入力される文字列をバッファに挿入しておき、このバッファから一文字づつ読み取り、エコー処理に見立てた文字出力と編集処理を50回繰り返した時の一文字当りの処理時間を求めた。

- 実験4

目的: Page実行モデルにおいて、実験1と比較するためにYy-serverで入力に対するエコー処理だけを行なった場合の実行時間を求める。

方法: サーバ側で処理した場合の一文字づつ出力する処理で、一文字当り処理時間を求める。

実験1と2では、図1のLisp処理部、通信処理部、それと通信処理部中のパケット生成、パケット送出の各々の処理時間を測定した。

The analysis of data output on Yy-client
Takashi KOSAKA[†1], Masayuki IDA[†2], Keisuke TANAKA[†2]
[†1]Aoyama Gakuin Univ./CSK Corp. [†2]Aoyama Gakuin Univ.

表 1: 1 文字当たりの出力時間 (単位はミリ秒)

実験番号 文字数	*Lisp* 処理部	通信処理部 *A* / *B*	出力時間 *C*
1 20 文字	19.00	12.48 5.46 / 7.02	31.48
2 10 文字	2.77	1.40 0.35 / 1.05	4.17
2 50 文字	1.96	0.36 0.21 / 0.15	2.32
2 100 文字	1.52	0.28 0.15 / 0.13	1.80
2 500 文字	1.12	0.05 0.02 / 0.03	1.17
2 1000 文字	0.82	0.10 0.01 / 0.09	0.92
3	3.52	100.22	103.74
4	7.00	—	7.00

[表注: 通信処理部の時間は、A と B の和である。尚、A は パケット生成、B はパケット送出の時間である。C は、Lisp 処理部と通信処理部の和で、一文字当りの出力時間である。]

実験 3 では、文字エコーの処理が複数の描画コマンドや、ウィンドウの整形コマンドを複雑に呼び出しているため、単純に通信処理だけを抜き出してパケットの生成時間やパケットの送出時間を測定できない。従って、Lisp 処理部の処理時間とエコー処理を全て通信処理とした時の処理時間を測定した。

実験 1,2,3 における実行時間は Lisp の `time` 関数を用いて測定した。このとき、通信処理部の時間測定中は、パケット生成やパケット送出の時間の測定は行っていない。パケット生成やパケット送出の時間の測定は、各々独立して行なった。このため、完全に同一の条件で、測定を実施したわけではない。しかし、そのことに対しての影響は少ないと考えられるので、無視した。実験 4 については、UNIX の時刻摘出関数 `gettimeofday` を用いて測定した。

尚、実験に使用したハードウエアは、SUN4/280HM、ソフトウエアは、SunOS 4.0.3、X-Window Ver. 11 Release 4、Lucid Common Lisp4.0.0 Beta-1 である。

4 実験結果と解析

出力実験の結果を表 1 に示す。実験 1 より文字出力という一つのコマンドの場合、通信処理と Lisp 処理とは同じオーダであることが判った。次に、実験 2 において、1000 文字の場合を除いて、文字数と通信処理を掛け合わせた結果は、

1.40 msec × 10 = 14.0 msec
0.36 msec × 50 = 18.0 msec
0.28 msec × 100 = 28.0 msec
0.05 msec × 500 = 25.0 msec

というように、同じオーダであり、文字数の差ほど処理時間の差はないことが判った。更に、この結果は、実験 1 の通信処理時間と同じオーダである。

上記より、一回の文字の出力には最低限必要な時間があり、文字数が少なければ、費やす時間の比重は大きいと言える。このことは、文字を出力するコマンド以外のコマンドにも当てはめられる。従って、小さな処理を連続して多量に行なう処理、例えば入力編集のエコー処理は、通信処理の負荷により *Yy-client* で処理するのに向かないと考えられる。

また実際に、実験 3 の結果で明白なように一文字当りの処理速度は 103.74 msec である。この結果では、[1] で示された Page 実行モデルの指標値に対して、大きく離れている。従って、この結果からも Page 実行モデルにおいて、入力編集処理を *Yy-client* 単独で行なう方式は適当でない。

1000 文字出力の結果は、表 1 で示されるように、一文字当たりの出力時間は、0.92 msec で、[1] で示された Viewport 実行モデルの指標値を満足している。このことで、文字の出力における Lisp の `write-string` や `write-char` で出力したい文字列を予め作っておき、`force-output` で一度に出力するならば、実験 2 の 1000 文字出力の結果が反映され、高速に表示される。

一般の描画処理のように 表示処理を連続して多量に行なう場合は、全ての表示処理を完了するのに、最低でも通信処理 × コマンド数の処理時間がかかってしまう。ここで、実験 2 において、文字数とパケット生成を掛け合わせた結果は、

0.35 msec × 10 = 3.5 msec
0.21 msec × 50 = 10.5 msec
0.15 msec × 100 = 15.0 msec
0.02 msec × 500 = 10.0 msec
0.01 msec × 1000 = 10.0 msec

となり、10 文字以降の処理時間は文字数に比べて差がないことが判る。このことは、パケット生成時のデータ量にあまり依存しないことを意味する。従って、*Yy* -protocol 上の命令やコマンドにある程度複雑な属性を持たせても、通信時間の増加につながらない。表示処理を連続して多量に行なう場合は、それらの表示機能をまとめるような機能の高いコマンドを持つ *Yy*-protocol を検討する必要がある。

5 終わりに

Yy-client の出力実験を実験モデルを用いて行なった。この結果、入力エコーを含む編集処理は、Page 実行モデルにおいて、*Yy-client* 単独で行なう方式が適当でないことが判った。また、Viewport 実行モデルにおいて、文字の出力ならば *Yy-client* で行なっても十分対応できることが判った。更に Viewport 実行モデルにおいて、異なる表示コマンドを大量に発行される場合の検討項目が見つかったので、*YyonX* version2.0 のための *Yy* -protocol version2.0 の検討を進めている。

また、実験結果として本論中には述べていないが、間欠同期の回数を変えた実験も行なった。その結果、コマンド毎に同期をとるのは、全体としての処理時間の増加につながる。また間欠同期の間隔があまり大き過ぎても、全体としての処理時間の増加につながることも判った。これらの結果は、上記の本論の目的に直接影響しなかったので、ここには、その詳細を含めていない。

参考文献

[1] 井田昌之、他 : *YyonX* 実行モデル (1), 情報処理学会第 41 回全国大会, Sept. 1990.