情報ネットワーク学演習 2
課題：cbench の高速化

秋下 耀介
情報ネットワーク学専攻
学籍番号；33E16001
E-mail: y-akishita@ist.osaka-u.ac.jp

2016 年 10 月 18 日（火）

1 問題

1.1 Cbench のボトルネック調査

Ruby のプロファイラで Cbench のボトルネックを解析しよう．以下に挙げた Ruby のプロファイラのどれかを使い，Cbench や Trema のボトルネック部分を発見し遅い理由を解説してください．

- profile
- stackprof
- ruby-prof

これ以外にもいろいろあるので、好きな物を使ってかまいません。

1.2 発展課題：Cbench の高速化

ボトルネックを改善し Cbench を高速化しよう．

2 解答

2.1 Cbench のボトルネック調査

ruby-prof によって cbench.rb を解析した結果のうち，上位は以下のようになっている．

%self	total	self	wait	child	calls	name
2.71	26.020	3.686	0.000	22.334	316359	BinData::Struct#instantiate_obj_at
2.28	12.592	3.097	0.000	9.495	456350	*BinData::BasePrimitive#_value
2.21	3.155	3.003	0.000	0.152	283781	BinData::Base#get_parameter
1.97	2.677	2.677	0.000	0.000	196898	String#sub
1.81	3.854	2.457	0.000	1.397	276225	Kernel#define_singleton_method
1.73	2.350	2.350	0.000	0.000	196890	String#to_sym

```
  1.72      2.333     2.333     0.000     0.000   196890   Array#index
  1.72      8.735     2.329     0.000     6.406   196890   BinData::Struct#base_field_name
  1.71      6.781     2.328     0.000     4.453   126489   Kernel#dup
  1.71      2.317     2.317     0.000     0.000   216173   Hash#[]=
  1.69     23.173     2.295     0.000    20.878   196890   BinData::Struct#find_obj_for_name
  1.63      2.214     2.214     0.000     0.000   381726   Symbol#to_s
  1.62      2.206     2.206     0.000     0.000   355532   BasicObject#!
  1.55      4.148     2.108     0.000     2.040   187288   BinData::Struct::Snapshot#[]=
```

上記の結果より， *BinData::BasePrimitive#_value はメソッドの呼び出し回数が最も多いということが読み取れる．1つのメソッドの実行時間はさほど差がないため，その実行回数がプログラムの速さに大きく影響を与えると考え， *BinData::BasePrimitive#_value がボトルネックになっていると判断した．

2.2 発展課題：Cbench の高速化

fast_cbench.rb は，cbench.rb を改良したプログラムとなっている．具体的には，cbench.rb のメソッド packet_in において，

```
def packet_in(datapath_id, packet_in)
    send_flow_mod_add(
      datapath_id,
      match: ExactMatch.new(packet_in),
      buffer_id: packet_in.buffer_id,
      actions: SendOutPort.new(packet_in.in_port + 1)
    )
  end
```

となっている部分を，インスタンス変数@flow_mod を用いて，

```
def packet_in(dpid, packet_in)
    @flow_mod ||= create_flow_mod_binary(packet_in)
    send_message dpid, @flow_mod
  end
```

のようにして，メソッド create_flow_mod_binary を呼び出す形に変更している．”||=”のように記述することによって，インスタンス変数の中が偽か未定義ならば，create_flow_mod_binary メソッドが呼び出される．すなわち，初めて packet_in メソッドが呼び出されたときのみ create_flow_mod_binary メソッドが実行されるが，それ以降はインスタンス変数@flow_mod の中が定義されているので，create_flow_mod_binary メソッドは実行されずインスタンス変数@flow_mod を使い回す．これによって，フィールド，バッファ ID，アクションといったオプションの指定をクラス全体として一度しか行うことなく packet_in メソッドが施行され，結果としてプログラムの高速化が実現されている．

fast_cbench.rb を解析した結果のうち，上位は以下のようになっている．

```
 %self      total      self      wait     child     calls  name
  3.37     26.263     3.758     0.000    22.505   249686   BinData::Struct#instantiate_obj_at
  3.36      3.876     3.742     0.000     0.134   263124   BinData::Base#get_parameter
  2.52      2.804     2.804     0.000     0.000   161511   String#sub
  2.41      8.760     2.680     0.000     6.080   161503   BinData::Struct#base_field_name
  2.35     16.102     2.616     0.000    13.486   161503   BinData::Struct#find_obj_for_name
  2.31      4.405     2.575     0.000     1.831   256530   Kernel#define_singleton_method
  2.23      3.940     2.485     0.000     1.455   218620  *BinData::BasePrimitive#_value
  2.11      2.347     2.347     0.000     0.000   187818   Hash#[]=
```

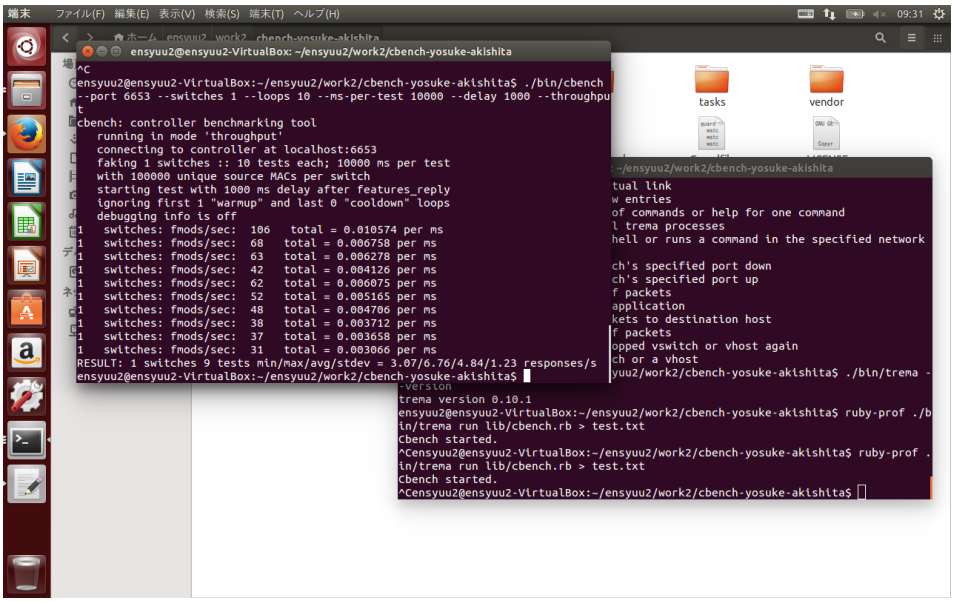

図 1　cbench.rb

1.98	2.209	2.209	0.000	0.000	161503	Array#index
1.96	2.185	2.185	0.000	0.000	161503	String#to_sym
1.86	19.885	2.070	0.000	17.815	130237	BinData::SanitizedPrototype#instantiate
1.76	4.041	1.958	0.000	2.084	156237	BinData::Struct::Snapshot#[]=
1.67	1.858	1.858	0.000	0.000	333834	Symbol#to_s
1.64	1.831	1.831	0.000	0.000	256531	BasicObject#singleton_method_added
1.63	1.818	1.818	0.000	0.000	130251	Kernel#initialize_copy
1.57	1.749	1.749	0.000	0.000	302269	BasicObject#!

*BinData::BasePrimitive#_value の呼び出し回数が半数ほどとなり，またトータルの実行時間も $\frac{1}{3}$ ほどになっていることが読み取れる．また，図 1 と図 2 を比較すると，Flow Mod の送信速度が fast_cbench.rb において 2 倍程増加していることからも，fast_cbench.rb は Cbench の高速化が実現されていると言える．

参考文献

[1] 情報ネットワーク学演習 2 事前準備 https://github.com/handai-trema/syllabus#事前準備
[2] Trema で OpenFlow プログラミング http://yasuhito.github.io/trema-book/#cbench

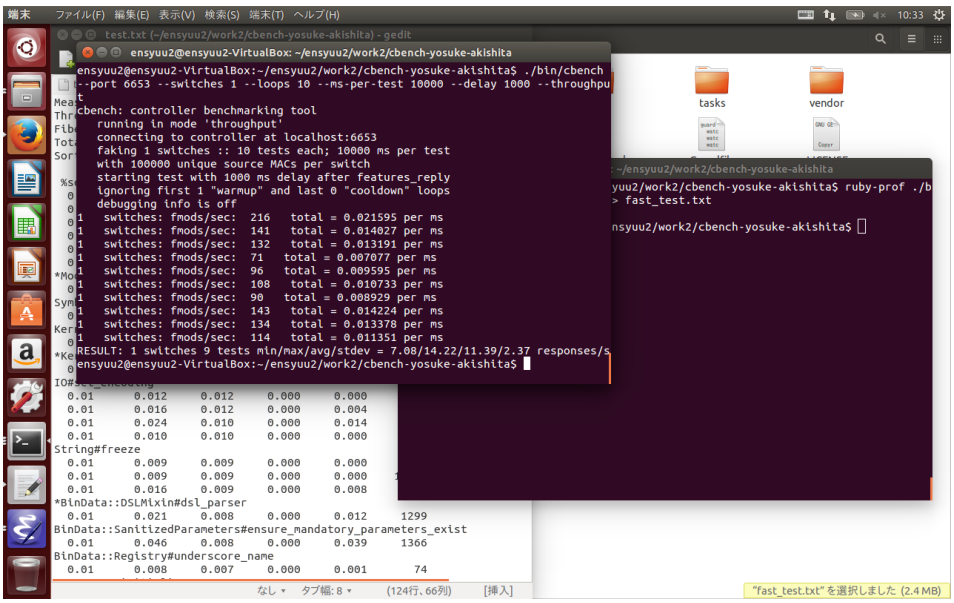

図 2　fast_cbench.rb