

秋ですね。ぶどう狩りやシェルスクリプトで夕涼み。

USP

for the sophisticated shell scripters
日本で唯一のシェルスクリプト総合誌

MAGAZINE

特集

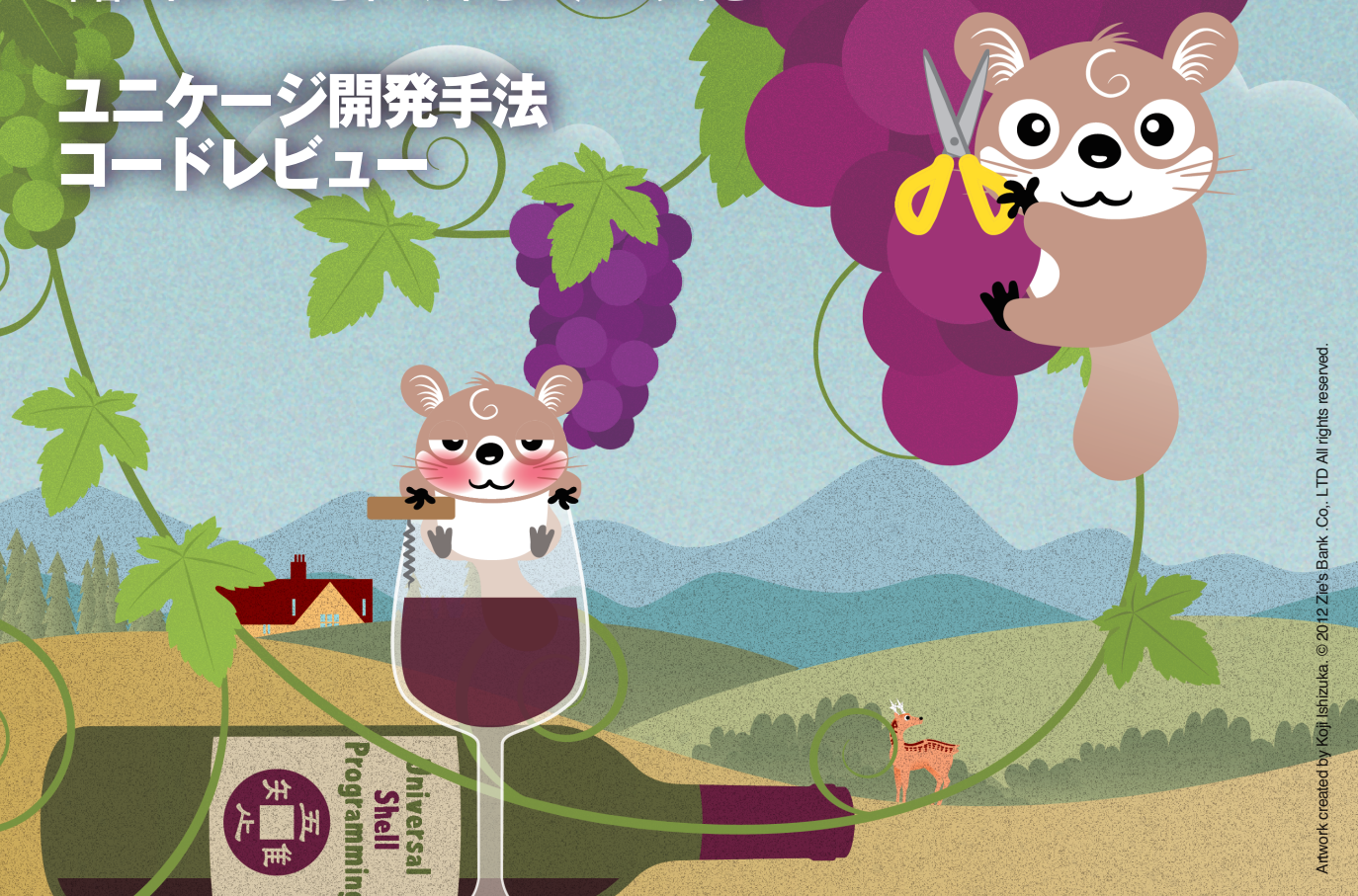
軽量言語

Lightweight Language が 世界を楽しくする

TechLION 再録

小山哲志×角谷信太郎×鈴木たかのり×
前島有貴×和田裕介
本音バトル! LL とイベントとコミュニティと!

ユニケース開発手法 コードレビュー



Universal
Shell
Programming

C o n t e n t s

04 特集

軽量言語

Lightweight Language が
世界を楽しくする

16 ユニケーシ開発手法
コードレビュー 第6回

大内智明

21 未来に生きる! 現場で使える!
データモデリング 第7回

熊野憲辰

24 「シェル芸」に効く
AWK 処方箋 第6回

齊藤博文

28 人間とコンピュータの可能性
第17回

大岩元

30 シェル芸勉強会後追い企画
Haskellで
やってはいかんのか? 第6回

上田隆一

34 Linux 女子部誌上講座
リナ女のたしなみ 第2回

平愛美

36 漢のUNIX 第14回

後藤大地

40 縁の木、育てよう 第4回

白羽玲子

42 スズラボ通信 第9回

すずきひろのぶ

47 Open usp Tukubai
おためし on AWS 第2回

坪和樹

50 組織文化を変える汗と涙の物語
アジャイル改善塾
第5回

山海一剛

52 ユニケーシエンジニアの作法
第11回

松浦智之

58 りゅうちの作業員日記 第6回

りゅうちてつや

62 「医は仁術なり」 シェル魔人
／編集後記



```
<tr bgcolor="FFFFFF">
  <td align="left" >SV1</td>
  <td align="left" >192.168.1.1</td>
  <td align="left" >/dev/sda1</td>
  <td align="right">2,361,152</td>
  <td align="right">479,354</td>
  <td align="right">1,759,000</td>
  <td align="right">22</td></tr>
```

輕量



AWK
ECMAScript
ActionScript
JavaScript
JScript
Lua
Perl
PHP
Python
Ruby
ShellScript



言語



「軽量言語」(Lightweight Language、LL)とは、ソースコードの作成、修正、実行が容易とされるプログラミング言語の総称です。この10数年の間にバージョンアップを重ね、国内外でユーザー数を増やしています。言語系イベントの数や規模の拡大がそれを物語っているでしょう。

軽量言語の定義や該当する言語の種類については諸説あります。しかし、「比較的短期間に習得することが可能」「アプリやシステムの開発・運用現場で活用され、利用者の仕事の負担を軽減するのに役立つ」「利用するユーザーのコミュニティがあり、デバッグやローカライズへの貢献がなされている」「メンバー間で効果的な学習方法や使いこなすコツについて日夜研究が進められている」――

などの共通する特徴が存在します。

本特集では始めに、6/28に開催されたTechLION vol.17をレポート。言語系コミュニティやイベントの舞台裏に迫ります。それぞれの自負とコミュニティへの愛、相互のリスペクトを感じ取ってください。また今回、日本UNIXユーザ会の法林浩之さんの協力を得てLLの歴史、いま注目株の言語をまとめました。LLが支持される本当の理由とは何か、そして、その将来像は。本記事から見てくるはずです。

_JIKOKU###<

e="###DATE#

tmp-calsed.1st

>

e="20131210" name="DAY_SELECT">

084,580 479,356,224 1,759,881,436 22 / 正

250,000 1,511,331,296 281,350,740 85 / 80

>

>

>

>

>

>

>

>

>

>

>

>

>

>

>

>

>

>

>

Tcl
VBScript
Clojure
Curl
Go
Groovy
Haskell
JavaFX Script
PowerShell
Scala
Scheme

上危険</td>

<td>8.1.1</td>

<td>da2</td>

<td>084,580</td>

<td>6,224</td>

<td>881,436</td>

>

Lightweight Languageの歴史

written by 法林浩之 (日本 UNIX ユーザ会)

Lightweight Language (LL) という概念や、それに類する言語はどこで生まれ、どのような発展を遂げてきたのだろうか。本稿ではLLの起源から日本への伝来、さらに各種言語および言語コミュニティの歴史を概観する。

LL の起源

Lightweight Language という単語が初めて使われたのは、2001 年 11 月に米国の MIT で行われた「LL1: Lightweight Languages Workshop」¹ であるとされている。

翌年に行われた LL2² では Ruby 作者のまつもとゆきひろさんも講演している。

イベントの Web サイトには LL についての厳密な定義は記述されていないが、まつもとさんは講演の中で「Lightweight Language とは、人間がより少ない『脳力』でプログラミングできる言語である」という見解を述べている。

つまり、無駄な労力を費やすことなくプログラムを書くことができ、高い生産性をもたらす言語、というような意味合いであろう。

日本への輸入と定着

日本における Lightweight Language の初出は、2003 年 8 月に行われた LL Saturday³ である。もともとは jus とアスキーで Ruby のイベントを企画していたのだが、国内には Ruby だけでなく Perl も PHP も Python もユーザコミュニティがあるので、声を掛けて合同でイベントをやろうという話になったものである。

ここで、このイベントにふさわしい名前を考えていたところに米国版 LL イベントの話聞き、近い方向性を感じたのでイベント名称に Lightweight Language を冠することにした。

米国版 LL イベントは 2 回で終了し、LL という言葉

も定着しなかったが、日本の LL イベントは毎年コンスタントに開催を続け、またそれまでになかった「プログラミング言語の枠を超えた交流」を生み出したことで多くの注目を集めるムーブメントとなり、概念として定着した。

LL イベントに登場する言語の変遷

LL イベント開始以降の LL の歴史は、イベントに登場した言語や実施されたセッションを振り返ることで、ある程度概観できるだろう。ここではまずイベントに登場した言語を振り返る。

初年度 (2003 年) は Perl、Python、PHP、Ruby の 4 言語でスタートした LL イベントだが、評判を聞きつけたのか 2 年目以降は俺も俺も的に出場言語が増加した。記録を見ると、2004 年は 12 言語、2005 年も 11 言語、2006 年は 16 言語、2007 年に至っては 23 言語も登場していて、この年がもっとも出場言語数が多かったようである。

2008 年以降は、出場言語数という観点では徐々に沈静化に向かっている。その要因のひとつとして、RubyKaigi⁴ や YAPC⁵ のような各言語コミュニティによるイベントが始まり、各言語に特化した話題はそちらに移行したこと、小規模なコミュニティや勉強会であれば簡単に実施できるようになり、LL イベントのような場を利用する必然性がなくなったのではないかと推測している。

LL イベントのセッションにみる技術の推移

表 1: 各年の技術を象徴するセッション

年	タイトル	解説
2003	LLとオブジェクト指向	Perl, Python, PHP, Rubyのオブジェクト指向プログラミング手法や実装を比較した。
2004	LLとblog	この頃からブログが流行し始めた。
2005	フレームワーク対決	フレームワークもこの年ぐらいからよく使われるようになった。
2006	LLで関数プログラミング	Haskellなどの関数型言語に注目が集まったことで実施された。
2007	VM魂	Java VMや.NET環境でのLLの利用法を紹介した。
2008	古い言語、新しい言語	LLVMとECMAScriptの処理系について議論した。
2009	渡る世間は雲ばかり	Google App EngineやWindows Azureなどクラウド上でのプログラミングについて議論した。
2010	LLでフィジカル コンピューティング	GainerやArduinoなどのボードをLLで制御する手法を解説した。
2011	JavaScript八面六臂	あらゆる分野で使われるようになったJavaScriptについて議論した。
2012	俺たちの継続的hoge hogeは始まったばかりだ!	継続的インテグレーションなどの取り組みを紹介した。
2013	Infrastructure as LL	ChefやPuppetなどLLを用いたインフラ構成管理について議論した。

過去の LL イベントのプログラムを眺めてみると、その時代の最新技術やプログラマー達の関心事が反映されていることがわかる。

ここでは各年のプログラムから当時の技術動向を象徴するセッションを列挙すること（表 1）で、LL 関連の技術の変遷をたどる。

十数年の間に変わったこと

LL イベントを運営して 12 年になるが、その間に LL を取り巻く状況は大きく変化した。

中でも顕著と思われることを 2 つ取り上げたい。

★ LL の業務ツール化

LL イベント開始当初、業務における LL の用途としては、Perl による CGI や PHP による簡単な Web アプリケーション、およびシェルや Perl などによるシステム管理作業の自動化やデータ処理といったところが中心で、分野が限られていた。当時、「LL を仕事に」という、どうすれば業務で LL を利用できるかを議論するセッションを行ったこともある^{*6}。

ところが、Ruby on Rails などのフレームワークを用いた Web サービス開発が一般化すると、LL を業務で利用することが普通に行われるようになった。

特にそれまで業務での利用シーンがなかなか増えなかった Ruby の普及が一気に進み、今では Ruby で仕事をする人が普通に存在するようになったのは隔世の感がある。

★ JavaScript の台頭

もうひとつの大きな変化は JavaScript の台頭である。LL イベントでは初年度から登場しユーザーも多

い Perl, PHP, Python, Ruby を「四大言語」と称してきた。

その頃の JavaScript はブラウザの動作に少し動的な変化をつける程度の言語という扱いだったのだが、2005 年ぐらいからの Ajax の流行により新たな価値を見出され、Web サービスに欠かせない言語としての地位を確立した。さらに近年は Node.js のようなサーバー側での利用や、スマートフォン用のアプリケーション開発にも使われるようになった。

おそらく今では四大言語を超えてもっともユーザー数の多い LL であろう。

これからの LL は

LL イベントは「明日のプログラミングが楽になるように」というのがひとつのテーマであると私は考えている。

そのために新しい言語やツールを使ってみたりするわけだが、ここで、10 年前の LL と今の LL は同じなのか、そして今の LL はこの先も LL で在り続けられるのか、というのが最近の私の疑問である。10 年も経てば新しい言語が台頭し、取って代わられることもあるだろうと思っていたのだが、今のところはまだそこまでには至っていないようである。しかし 10 年後はどうだろうか。

このイベントに登場する言語に新陳代謝があるのかどうか、これからも注目していきたい。

*1:<http://ll1.ai.mit.edu/>

*2:<http://ll2.ai.mit.edu/>

*3:<http://lljus.or.jp/lls2003/>

*4:<http://rubykaigi.org/>

*5:<http://yapcasia.org/>

*6:<http://lljus.or.jp/llw2004/program.html>



第十四回 UFS ファイルシステムの特徴を知る FAT32と比較する



フラグメントとシリンダグループはSSDでは逆効果？ 1

登場初期の UFS は、大雑把に捉えると 3 つの区切りしか存在しなかった。ディスクの情報を納めた「スーパーブロック」、inode のデータを納めた「inode ブロック」、実際のデータを納めた「データブロック」だ。ストレージのサイズが大きくなってくると性能を引き上げるためにブロックサイズを大きくするようになった。そうすると小さいファイルが巨大なブロックに収まることになり、実際には使われていない領域が増えることになる。これを回避するためにブロックの書き込み単位として「フラグメンテーション」が導入されることになる。

UFS ではメタデータの更新 (inode の書き換え) とデータの書き換えを実施するわけだから、ハードディスクのヘッドは「inode ブロック」と「データブロック」の間を行ったり来たりすることになる。ストレージのサイズが大きくなってくると、この行き来にかかる時間が無視できないものになってきた。そこで UFS は「シリンダグループ」という概念を導入する。一組しか存在しなかった「inode ブロック」と「データブロック」を、小さいサイズの『「inode ブロック」「データブロック」』の集まりとして捉えるようになる。この 1 つの集まり (単位) を「シリンダグループ」と呼ぶ。ハードディスク上の「inode ブロック」と「データブロック」の位置を、ヘッドの物理的な移動経路に近いところへ集約し、ヘッドの移動に費やされる時間を短縮する高速化を狙った。

つまり、現在の UFS が実装している次の 2 つの機能

は、HDD の「物理特性」に合わせて導入された機能ということになる。

●フラグメンテーション：転送速度を引き上げた場合でもディスクのスペース面での使用効率が下がらないようにする。

●シリンダグループ：ヘッドが移動することで発生する遅延を回避するために、小さいグループをたくさん作ってヘッドの動きがなるべく近場で収まるようにする。

SSD は上記の制約を受けないため、こうした機能はむしろ動作を遅くする原因になっていると考えられる。SSD というデバイスに対しては、むしろ登場初期のシンプルな UFS の方が高速化が期待できる可能性がある。今回はそれを検証してみよう。



FAT32ファイルシステム 2

「スーパーブロック」「inode ブロック」「データブロック」しか存在しない頃の UFS の構造は、FAT ファイルシステムによく似ている。FAT ファイルシステムの構造はさらに単純で、「FAT」と「データブロック」のみだ。後に「スーパーブロック」に相当するブロックも追加され、細かい構造は違うが大枠の区切り方は近い。

シリンダグループを 1 つにした UFS を作ってベンチマークを実施したいところだが、現在の UFS2 の実装はシリンダグループの使用が前提になっているため、シリンダグループのサイズを変更する、個数を変えろといったことが簡単にはできない。そこで、こうした回転円盤向けの機能が入っていない状態で、これまで実

表 3.1

ブロックサイズ	フォーマットコマンド
4KB	<code>newfs -O 2 -a 32 -b 4096 -d 4096 -e 512 -f 4096 -g 16384 -h 64 -i 8192 -k 492 -m 8 -o time -s 482344960 /dev/gpt/s</code>
4KB (SU)	<code>newfs -O 2 -U -a 32 -b 4096 -d 4096 -e 512 -f 4096 -g 16384 -h 64 -i 8192 -k 492 -m 8 -o time -s 482344960 /dev/gpt/s</code>
32KB	<code>newfs -O 2 -a 4 -b 32768 -d 32768 -e 4096 -f 32768 -g 16384 -h 64 -i 65536 -k 4167 -m 8 -o time -s 482344960 /dev/gpt/s</code>
32KB (SU)	<code>newfs -O 2 -U -a 4 -b 32768 -d 32768 -e 4096 -f 32768 -g 16384 -h 64 -i 65536 -k 4167 -m 8 -o time -s 482344960 /dev/gpt/s</code>

表 3.2

セクタサイズ	クラス あたりの セクタ数	フォーマットコマンド
512B	1	<code>BytesPerSec=512 SecPerClust=1 ResSectors=32 FATs=2 Media=0xf0 SecPerTrack=63 Heads=16 HiddenSecs=0 HugeSectors=272629781 FATsecs=2097152 RootCluster=2 FSInfo=1 Backup=2</code>
1KB	1	<code>BytesPerSec=1024 SecPerClust=1 ResSectors=16 FATs=2 Media=0xf0 SecPerTrack=63 Heads=16 HiddenSecs=0 HugeSectors=241172442 FATsecs=934777 RootCluster=2 FSInfo=1 Backup=2</code>
2KB	1	<code>BytesPerSec=2048 SecPerClust=1 ResSectors=8 FATs=2 Media=0xf0 SecPerTrack=63 Heads=16 HiddenSecs=0 HugeSectors=120586221 FATsecs=234604 RootCluster=2 FSInfo=1 Backup=2</code>
4KB	1	<code>BytesPerSec=4096 SecPerClust=1 ResSectors=4 FATs=2 Media=0xf0 SecPerTrack=63 Heads=16 HiddenSecs=0 HugeSectors=60293079 FATsecs=58766 RootCluster=2 FSInfo=1 Backup=2</code>
8KB	1	<code>BytesPerSec=8192 SecPerClust=1 ResSectors=4 FATs=2 Media=0xf0 SecPerTrack=63 Heads=16 HiddenSecs=0 HugeSectors=30146508 FATsecs=14706 RootCluster=2 FSInfo=1 Backup=2</code>
16KB	1	<code>BytesPerSec=16384 SecPerClust=1 ResSectors=4 FATs=2 Media=0xf0 SecPerTrack=63 Heads=16 HiddenSecs=0 HugeSectors=15073254 FATsecs=3679 RootCluster=2 FSInfo=1 Backup=2</code>
32KB	1	<code>BytesPerSec=32768 SecPerClust=1 ResSectors=4 FATs=2 Media=0xf0 SecPerTrack=63 Heads=16 HiddenSecs=0 HugeSectors=7536627 FATsecs=920 RootCluster=2 FSInfo=1 Backup=2</code>

験してきたようなブロックサイズ / フラグメントサイズを変更した場合にどのように性能が変化するかを FAT ファイルシステムを使って実験してみたいと思う。

FAT32 では、UFS のブロックに相当する（わけではないのだが、似たような変更ができる）ものにクラスタがある。クラスタは 1 つ以上のセクタで構成される。クラスタとセクタがブロックとフラグメンテーションのような関係になっている（厳密にはだいぶ違う）ので、この部分を変更しながらベンチマークを実施する。クラスタを複数のセクタで構成する方法は実装が安定していなかったため、ここでは 1 セクタ 1 クラスタの構成で、セクタサイズを変更しながらベンチマークを実施した。

フォーマット 3

まず比較として UFS2 で特徴的だったブロックサイズが 4KB のケースと 32KB のケースについて、それぞれ Soft Updates 機能が有効になった場合と無効になった場合、フラグメントのサイズがブロックの 8 分の 1 の場合とブロックサイズと同じだった場合のベンチマーク結果を取得しておく。それぞれフォーマット時のオプションは表 3.1 のとおり（`dumpfs`

-m デバイスファイルで確認）。

FAT32 でのフォーマットには `newfs_msdos(8)` コマンドを使う。セクタサイズは -S で、クラスタごとのセクタの数は -c で指定できる。`newfs_msdos(8)` コマンドは FAT12、FAT16、FAT32 のフォーマットが可能で、-F でどのフォーマットを使うかを指定できる。この場合は「`newfs_msdos -F 32 -S 512 -c 1 デバイスファイル`」のように指定することでフォーマットを実施できる。フォーマット時のオプション結果を表 3.2 にまとめておく。

ベンチマーキング 4

これまで同様のベンチマークを実施すると次のページのような表とグラフが得られた。

FAT32 でフォーマットされたファイルシステムは、UFS2 において最も性能が高いケースに匹敵、もしくはそれを超える性能を発揮していることがわかる。

結果と考察 5

SSD はその特性上、フラグメンテーションやシリンググループを必要としないため、よりシンプルな

表 4.1

ブロック/ フラグメントサイズ、 セクタサイズ (FAT32)	シーケンシャル 書き込み速度 [バイト/秒]	シーケンシャル 読み込み速度 [バイト/秒]	ランダムファイル 書き込み時間[秒]	ランダムファイル 読み込み時間[秒]	ランダムファイル 削除時間[秒]
4KB/512B	154258522	348362603	44.33	44.75	24.20
4KB/512B (SU)	74865830	294362477	27.82	45.17	2.22
4KB/4KB	215529491	354176868	53.66	40.27	31.88
4KB/4KB (SU)	221060928	343450398	23.15	40.25	2.55
32KB/4KB	61224462	351800407	82.04	39.78	39.78
32KB/4KB (SU)	48110026	364483242	30.65	39.91	1.64
32KB/32KB	47813115	325646743	162.53	41.46	97.51
32KB/32KB (SU)	200185031	529667001	39.26	41.84	2.01
512B	83504106	2613903	25.99	50.51	8.72
1KB	178737607	5211556	23.10	45.04	6.38
2KB	261453169	10562707	25.80	42.23	9.01
4KB	261624874	530035959	32.59	38.99	10.93
8KB	236628939	535213592	50.52	38.49	20.16
16KB	195569248	535294759	69.54	39.22	31.75
32KB	211745885	539412048	105.68	41.80	54.22

図 4.2 シーケンシャル書き込み速度 ↑高いほど高性能

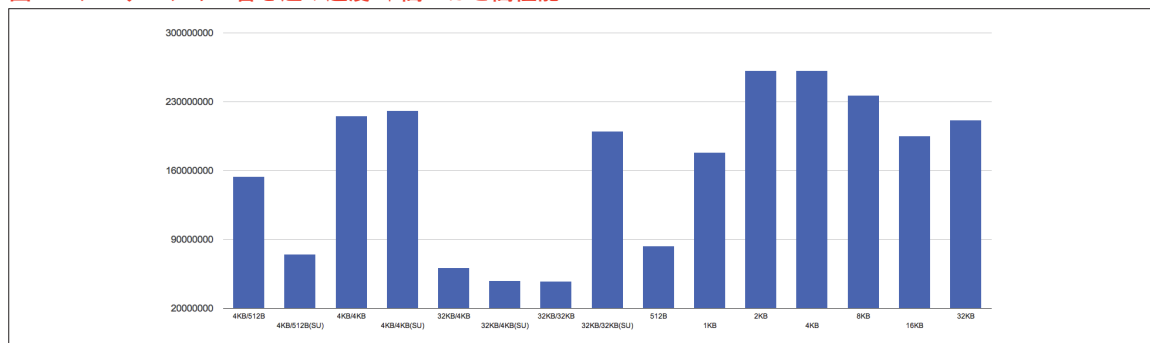
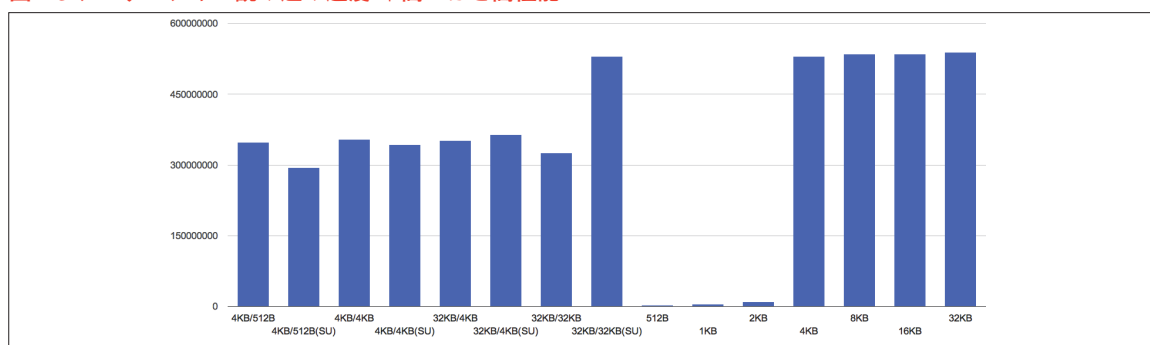


図 4.3 シーケンシャル読み込み速度 ↑高いほど高性能



ファイルシステムの方がパフォーマンスを発揮しやすいというのが、今回の結果から推測できる。今回のケースでは特に FAT32 のセクタを 4KB、クラスタにおけるセクタの数を 1 にした場合、シーケンシャルな処理でも、小さいファイルを大量に処理するようなケースでも、どちらでも高い性能が出ている。UFS と FAT では実現している機能の違いゆえにまったく同じレベルでパフォーマンスについて比較するのは意

味がないのだが、興味深い結果といえる。

データ処理などに特化したシステムを開発する場合、FAT32 もひとつの候補として検討できることになる。特に SSD をベースとしたストレージで高速データ処理を実施したいとなった場合、FAT32 は興味深い検討対象といえる。FAT32 では 1 ファイルあたりの最大サイズが 4GB という制限があるためビッグデータの扱いに難があるが、なかなかおもしろい。

図 4.4 ランダムファイル書き込み時間 ↓低いほど高性能

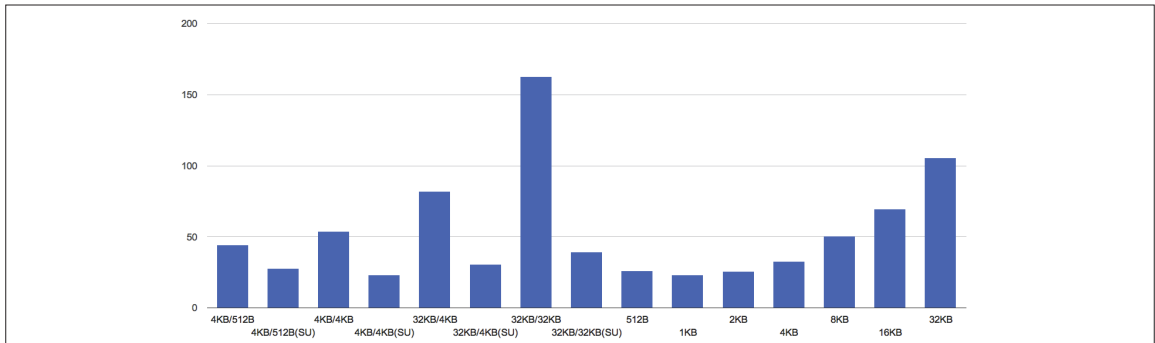


図 4.5 ランダムファイル読み込み時間 ↓低いほど高性能

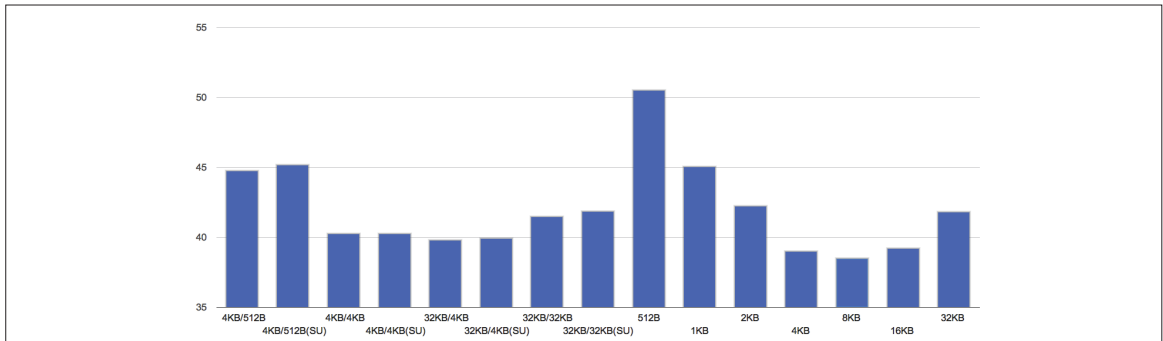
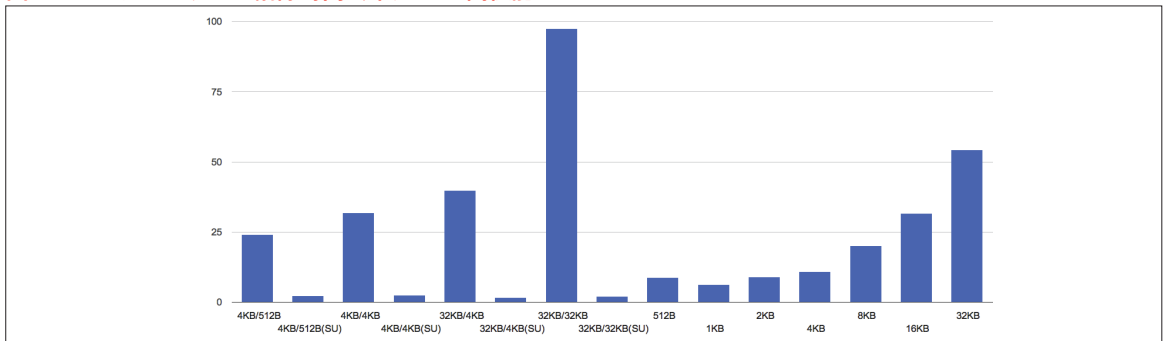


図 4.6 ランダムファイル削除時間 ↓低いほど高性能



逆の視点から今回の結果を捉えると、UFS2 の提供するさまざまな機能を実現しながら、FAT32（4KB セクタ）に匹敵する性能を発揮している UFS 32KB ブロック /32KB セクタ +Soft Updates 機能は良くできているといえる。FAT32 には Soft Updates 機能に相当する機能が存在しないため削除処理では UFS+Soft Updates の性能には及ばない。UFS からフラグメントとシリンダグループの機能をそぎ落とした状態で、Soft Updates 機能を有効にした場合、UFS ベースでも FAT32 を超える性能を発揮する可能性がある。



クリティカルパスと最適化

6

ソフトウェアのパフォーマンスを引き上げようとし

て特定の状況やハードウェアに特化した変更を続けると、ハードウェアの構成が変わったときにすべてが役立たずになることがある。このため極端な最適化はせずに、次のような指針でソースコードを記述するのがベターと言われる。「シンプルな設計」、「クリティカルパスを明確に」、「行き過ぎた最適化はしない」だ。

UFS は、回転円盤という物理特性に対応するために「シリンダグループ」と、そのデータ転送特性に対応するために「フラグメンテーション」という概念を導入したが、この概念を捨てて元に戻すと、SSD 向きのファイルシステムを作ることができる。その場合でも Soft Updates 機能や各種最適化機能はそのまま適用できる。

～便利なコマンドをクラウドで使おう～

Open usp Tukubai

おためし on AWS

第2回

AWS 性能評価 ～利用する前にコストを考えよう！

written by 坪 和樹

7月号から開始した本連載の第2回は、パフォーマンスの測定として「AWS+Open usp tukubai環境」の性能評価を行いたいと思います。第1回で“使える”環境ができあがったのに、性能評価を急ぐのはなぜ？ とお考えの方もいるかもしれません。実は、AWSを利用するコストを考える上で、性能評価が最も大切だと思われるからです。

なぜ性能評価が大切なのか？

Open usp Tukubai を利用する方々は、大規模なデータの集計やソートなどを行う機会が少なくないと思います。例えば、UEC のサイト (usp engineers' community site: <https://uec.usp-lab.com/>) には、「商品部門 / 地域毎売数データの帳票作成」というシェルスクリプトのサンプルが公開されています。これは、商品の売り上げレコードを集計するというサンプルレシピになっており、実に 300 万行のテキストデータでも難なく集計できることが示されています。

また、ご存知の通り AWS は 1 時間単位の時間課金となっています。つまり、起動直後に停止しても、1 時間ぎりぎりまで使い切っても料金は同じです。EC2 インスタンスは性能によって料金が設定されており、その中でも高価な EC2 インスタンスには、大容量の SSD 一時ディスクや高速なネットワーク帯域など有用な機能が付いてきます。つまり必要なときに高性能な EC2 インスタンスを使えば、それだけ処理が早く終わるということを意味します。

一般的には、分散処理を行おうとすると対応に手間がかかるかと思うので、1 台のインスタンスだけで処理を行うスクリプトであれば、すぐに検証することが可能です。そのためにも、手持ちのデータ量に対して、どのくらいの性能を割り当てる必要があるのかを知っておくことが大切なのです。

EC2 インスタンスの費用対効果を高めるために、性能試験は必須だご理解頂けたでしょうか。

●早速、試験結果を……の前に

手続き型になってしまい恐縮ですが、いくつか前

提や tips をお伝えさせて頂きたいと思います。

★試験環境

● OS は Amazon Linux

Redhat 系列の OS なので、Redhat や CentOS と同等の性能が出ることを期待できると思います。また、AWS 環境を便利に使うための CLI ツールが標準で用意されている等のメリットがあるため、EC2 インスタンス上における OS は Amazon Linux をお勧めします。

● EBS(Elastic Block Storage) は、SSD タイプ (General Purpose) を利用

2014 年 6 月に発表された、SSD タイプの EBS である General Purpose を利用して検証を行いました。このタイプではブースト機能があり、ブーストが無くなると I/O 性能が低下します。リリースされた直後に検証を行っているため、考察において十分ではない部分もあるかもしれません。本番環境で利用する場合などは必ず先に検証を行ってください。

★ tips

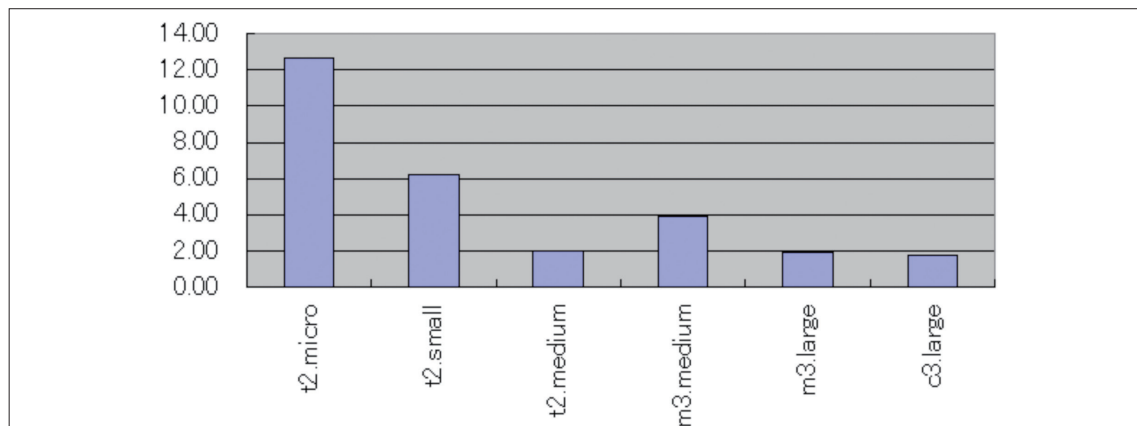
● AWS 環境は使い終わったら停止もしくは削除！

AWS ではインスタンスの起動時間で課金されるため、できれば使わないときは落としてください。また、ディスクや固定 IP も課金体系が設定されているため、再作成で良いものは都度削除して頂くのが無難です。

性能の比較

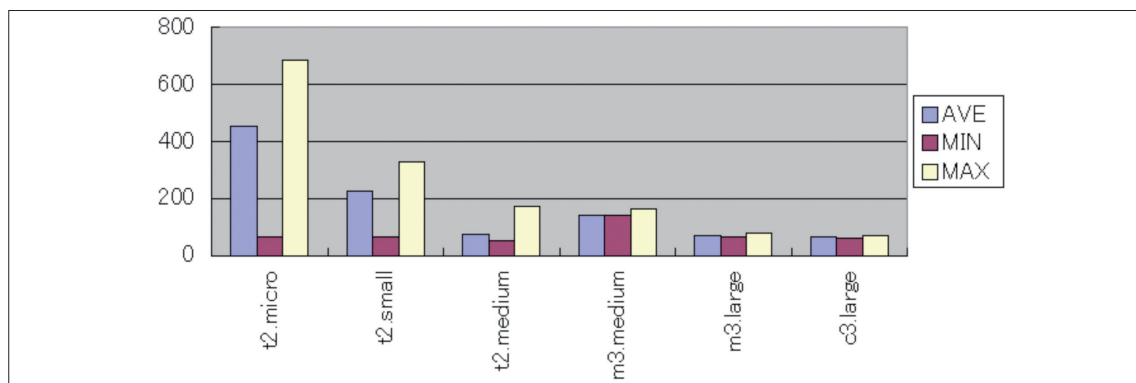
それでは、実際に比較を行った結果を図示します。

図 1: スクリプト処理を 100 回繰り返した時の合計時間



グラフの縦軸の単位は「時間」(h)

図 2: 処理 1 回あたりにかかる平均、最小、最大時間



グラフの縦軸の単位は「秒」

比較のために、紹介させて頂いた「商品部門 / 地域毎
売数データの帳票作成」という UEC のサンプルレシ
ピを利用して頂きました。サンプルレシピの中では
comma や keta といった Open usp Tukubai のコマ
ンドが利用されており、実際の利用シーンに近いの
ではと考えて選択しております。

★各 EC2 インスタンス毎の性能評価

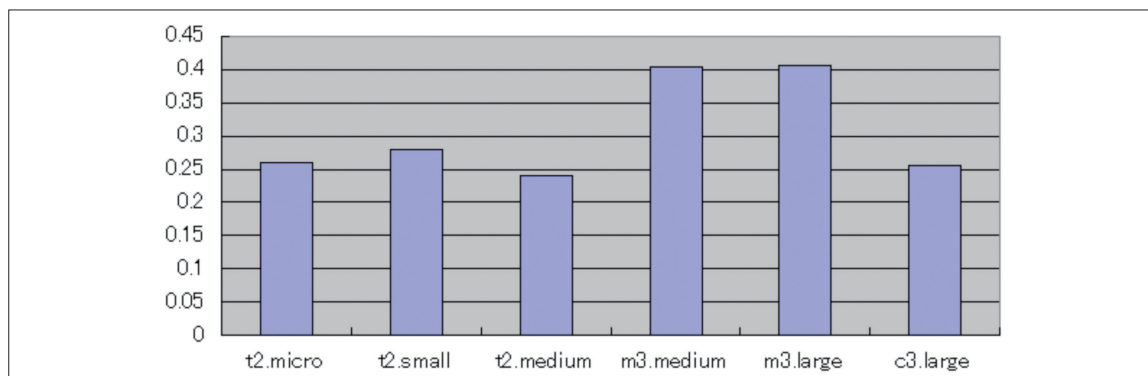
グラフは、各インスタンスの種類 (T2、M2、C3)
におけるサイズ (small、medium、large など) を比
較したものです。EC2 インスタンスで、起動直後に必
要なアップデートやインストール、試験環境の構築を
自動で行った後、サンプルレシピを 100 回繰り返し実
行しました。

なお、全ての EC2 インスタンスに対して複数回の
検証を実施し、いずれの検証でも値に乖離が無いこ

とを確認した上で記載しておりますが、必ずこうし
た時間で処理が終わることを保証するものではありません。何卒ご了承ください。

まず、100 回の処理を繰り返し実行したときの
合計時間 (図 1) です。t2.micro → t2.small → t2.
medium と処理時間が約半分になっていることが
分かります。これは割り当てられている CPU 性能
がインスタンスタイプに応じて約 2 倍ずつ増えてい
ることから見てても妥当な結果と言えます。一方で
m3.medium の性能が大きく t2.medium に劣ってい
るように見えますが、T2 には無いが M3 で提供され
る機能 (一時ディスクの割り当て等) もあるため、安
定した処理を提供する M3 が必要となる場面も多々
あるはずです。また、今回のサンプルは CPU がボト
ルネックになっていたものと考えられ、コンピュート
リソースを多く使える c3.large において最も早い結

図 3：各処理にかかったコスト



グラフの縦軸の単位は「ドル」

果となりました。

図 2 では、100 回実行した処理の各回について、処理時間の平均 (AVE)、最小 (MIN)、最大 (MAX) をグラフ化してみました。グラフからもお分かりのように、T2 インスタンスは処理速度にばらつきがあり、ブーストが無い場合に処理時間が長くなる傾向が見られました。

最後に、各インスタンスを利用した今回の検証で消費したコストを計算し、比較してみました。図 3 は、各インスタンスで検証を行った際のコストを示したものです。T2 インスタンスは安価な価格設定のため、処理が長かった場合でも安く済んでいることが分かります。また、適切なインスタンスを選ぶことでコストが 2 倍近くも違うことがあることが読み取れます。

今回の検証で、T2 系の EC2 インスタンスでは、ブースト機能により短時間で処理が終わる時もありますが、反面でブーストがなくなった後に処理時間が長くなる傾向があることが分かりました。M3 インスタンス、C3 インスタンスでは安定した処理時間になっていることも確認することができました。

UEC のサンプルレシピを 100 回実行したときのコストは、図 3 から分かる通り t2.medium が最も安価でした。ただし、無料利用枠がある場合には数時間以上かかっても t1.micro で実行した方がコストがかからずお得かもしれません。

また、今回の検証においては C3 系のインスタンスを利用すれば高速に処理が終わることが確認できました。注意点としては、コストが高いインスタンスを利用すれば簡単に高速化することができますが、一定以上の高速化を求める場合には工夫が必要です。例えば

EBS ボリュームの PIOPS や最適化オプションがパフォーマンスチューニングに役立つかもしれません。

まとめ

T2 インスタンスは安価な価格設定となっており、コスト面で優れています。ユースケースによっては処理時間を厳密に計算する必要があるなど、T2 が使えない場面もあるかもしれませんが、検証目的の場合にはかなり助かる局面もありそうです。

なお、今回の検証のために利用したサンプルレシピでは、入力として約 309 万行 (106MB) のファイルを一度シーケンシャルに読み込みますが、出力は標準出力に吐き出すため、ディスクへの書き込みを行っていません。

また、マルチコア環境でも性能が十分に発揮されているように見える理由として、スクリプト内では帳票を加工するために、実に 25 ものコマンドをパイプラインで実行していることが挙げられると思います。パイプラインで繋がれたコマンドはそれぞれ入出力を受け渡ししながら独立したプロセスで動くため、マルチコア環境でも良いパフォーマンスが得られたものと思います。

本稿の検証はあくまで一例ですが、指針なくクラウド環境に一步を踏み出すのはとても勇気が要ることです。今回ご紹介した検証結果が参考になり、読者の皆様の不安が軽減されれば、また、Open usp Tukubai 環境と AWS を利用する一助となれば幸いです。



HANDS LAB

ユニケージ®エンジニア数 最大級!

ハンズラボはユニケージ®開発手法に特化したITソリューション企業です。東急ハンズの営業システムを刷新したノウハウを駆使し、小売業における「オーダーメイド」のシステム開発を行います。

中途採用 大募集! 詳しくはHPで!

> <http://www.hands-lab.com/>

「ユニケージ®」は有限会社ユニバーサル・シェル・プログラミング研究所の登録商標です。

これであなたもユニケージエンジニア!

ユニケージ開発手法教育講座

「ユニケージ開発手法教育講座」は、ユニケージ開発手法におけるデータ管理の方法や、オリジナルコマンドの使用方法などをハンズオン形式で具体的に学べる講座です。UNIXの基礎からユニケージ開発手法による開発プロジェクトの進め方まで、ユニケージエンジニアとしてのトータルスキルを習得できます。

<http://www.usp-lab.com/LECTURE/CGI/LECTURE.CGI>



続々と新講座も充実中!

- K1 コマンド学習編
- K2 シェルスクリプト学習編 (帳票・バッチ処理)
- K3 シェルスクリプト学習編 (ウェブアプリケーション処理)
- K4 ユニケージアーキテクチャ編
- K5 ユニケージ開発環境セットアップ編
- K6 システム運用・管理編
- K7 プロジェクトマネジメント・人材育成編
- KSQL SQL エンジニアのためのユニケージ活用編
- KSTAT ユニケージにおける統計コマンド編

UNIX 初心者のための講座などもご用意しています。

TechLION
For Independent Engineers

技術の草原で百獣の王を目指す
エンジニアたちの新感覚トークライブ!

<http://techlion.jp/>

上記のサービス内容は2014年8月現在のものです。
最新の情報は弊社ホームページにてご確認ください。

ISBN 978-4-9048-0711-8
C3455 ¥500



9784904807118

USP 研究所
定価 (本体 500円+税)



1923455005002