システムソフトウェア研究の最前線

慶應義塾大学 河野健二

自己紹介

- 河野健二（慶應義塾大学理工学部情報工学科）
  - 東大助手，電通大講師，慶應義塾大学准教授当を経て，同教授
- 専門分野
  - オペレーティングシステムおよびシステムソフトウェア
    - ディペンダブル・コンピューティングにも少し浮気
- 主な論文
  - IEEE Trans. on Computers, USENIX ATC, ACM EuroSys, IEEE DSN, ACSAC, IEEE ICDCS, ACM VEE, ECOOP...
- 受賞
  - 2000年 情報処理学会山下記念研究賞
  - 2000年，2009年，2010年，2013年 情報処理学会論文賞
  - 2014年 日本ソフトウェア科学会ソフトウェア論文賞
  - 2015年 IBM Faculty Award
  - 2015年 ACM Service Award
  - 2016年 日本ソフトウェア科学会基礎研究賞

システムソフトウェアの研究トピック

- ACM SIGOPS == 国際的な研究コミュニティ
  - SOSP/OSDI には毎年 600 名程度集まる
  - アクティブな研究者は 150名ぐらい？
- 研究トピックは多岐に渡る
  - Many-core/Multi-core 向けのオペレーティングシステム
    - GPU のスケジューリング，GPUの仮想化など
  - ライブラリOS, 仮想化支援ハードウェアによるその支援
  - NVM（不揮発性メモリ）を利用したストレージ，KVS, FS
  - システムソフトウェアのバグ解析
    - Linux や Apache, MySQL 等のバグ解析
  - などなど

システムソフトウェア研究の楽なところ

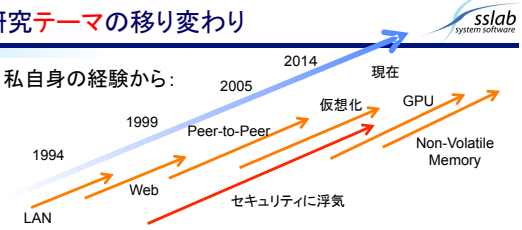
- システムソフトウェアは中間管理職
  - “アプリケーション”という名のわがまま上司
  - “ハードウェア” という名の身勝手な部下
- どこが楽なの？？？

システムソフトウェア研究の楽なところ

- 研究テーマを考える必要がない
  - 上司のわがままを聞いてあげればよい
    - アプリケーションの進化が新しいテーマをもたらしてくれる
  - 部下の身勝手に付き合ってあげればよい
    - ハードウェアの進化が新しいテーマをもたらしてくれる
- 「もうやることない」と思っても
  向こうから研究テーマが
  やってくる

研究テーマの移り変わり

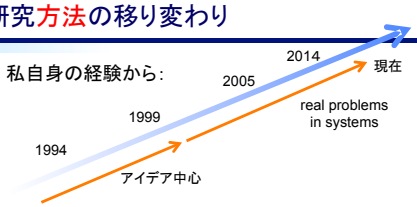
- 私自身の経験から：



- 1994～2000年ぐらいまで：
  - LANやPCクラスタを前提とした closed な分散システムの研究
- 2000～2010年ぐらいまで：
  - Peer-to-Peer などの Internet-scale の分散システムの研究
- 2005年～現在まで：
  - 仮想化をベースとしたクラウドのための基盤技術の研究
- 2013年～現在まで：
  - GPU や non-volatile memory など新しめのデバイスのための基盤

研究方法の移り変わり

- 私自身の経験から：



  - 1994～2000年ぐらいまで：
    - “空想” でモノを作っていればよかった
    - きっとこういう機能があると便利だろうな
    - しょぼい実装にしょぼい実験で許された
      - スクラッチからプロトタイプを作る
  - 2000年ぐらい～現在まで：
    - “現実” のシステムの “現実” の問題を解け
    - 問題が存在することを定量的に示すことからスタート
    - 提案方式を “現実” のシステムに “組み込む” 必要あり
    - しょぼい実験は許されない

システムソフトウェア研究の辛いところ

- “空想”の問題を解いても評価されない
- “正しさ” を証明する方法がない
- “良さ” を客観的に比較する方法がない

- 研究に対する評価が厳しい
  - 査読は減点法になってしまう
  - 文句をつけられる箇所があったら reject というノリ

どうやって辛さを乗り切るのか？（その1）

- “空想”の問題を解いても評価されない

- 実際のシステムで起きている問題を見つけろ！

- いろいろな視点から性能評価をやってみる
  - まずは人為的なベンチマークで極限状況を調べる
  - 直感に反する結果が出ていないかどうか調べる・考える
  - 直感に反する結果を見つけたら・・・
    - 何が起きているのか詳細に調べていく
  - 原因がわかったら・・・
    - 同じ現象が起きる実アプリケーションを探す

Containers or Hypervisors, Which is Better for Database Consolidation?

in collaboration with
Asraa Abdulrazak Ali Mardan

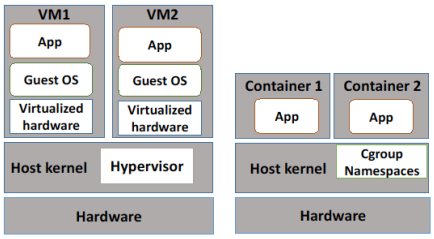
Introduction

- Virtualization becomes an essential building block in today datacenters
  - Meeting growing demands of resources by providing resources sharing and consolidation
  - Brings significant cost savings

- Major virtualization technologies
  - Hypervisor-based solution like KVM , Vmware ..etc
  - OS-level Virtualization or containers such as LXC , Docker..etc

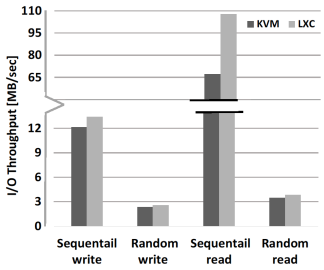
Containers vs Hypervisors

- Trade off between performance and isolation
  - Container provides near native performance with no virtualization overhead
  - Hypervisor provides strong isolation



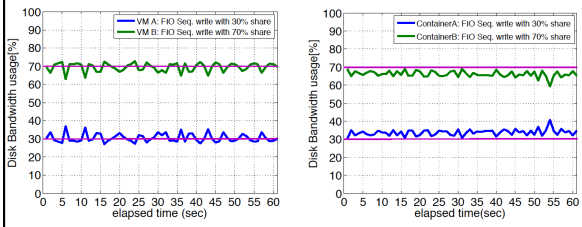
Disk I/O Performance

- 期待通り，コンテナのほうが性能がよい
  - LXC というコンテナ，KVM というハイパーバイザを利用



Disk I/O Performance Isolation

- Containers provide good performance isolation
  - Two VMs/containers are lunched
  - The one VM/container given 70% share, the other 30%



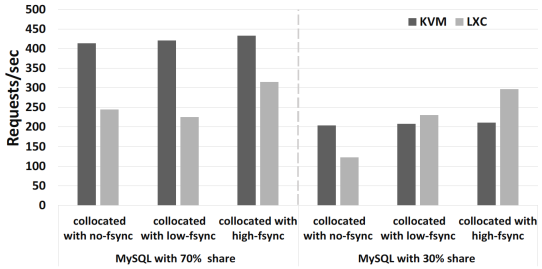
(a) Virtual Machines (b) Containers

Research Question

- Question:
  - Which is better for database consolidation?
    - Containers or Hypervisors?
- Answer:
  - Looks containers are better because of:
    - Better disk I/O performance
    - Comparable disk I/O isolation
- となるんだったら面白くない
  - 誰もが想像できる結果
  - 深く調べるまでもない

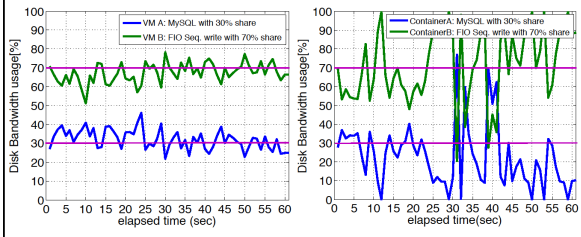
Answer: 予想とは逆の結果

- Hypervisors are better for DB consolidation
  - Provide better throughput



Answer: 予想とは逆の結果

- Hypervisors are better for DB consolidation
  - Provide better throughput & isolation



(a) Virtual Machines (b) Containers

どうやってこんな現象を見つけたのか？

- 研究者の期待に反する結果が得られたのが面白い
  - コンテナと VM の構造から想像すれば，コンテナのほうが（少なくとも）性能はいいはず
- 最初の動機は単純
  - 最近，コンテナって流行ってるよね
  - ハイパーバイザより性能がいいっていうけど，どれくらいいいんだろうね？
  - とりあえず，測ってみるか・・・
- システムソフトウェア研究の楽なところ
  - テーマは向こうからやってくる

実際には・・・

- 数百通りのパラメータの組み合わせで実験
  - （動きの解析しやすい）簡単なベンチマークを使う
  - とにかく brute-force で頑張る
  - 結果は出ると無邪気に信じる
    - 学生は半ベソでした・・・
- fsync の頻度が高いとコンテナの性能が低下する
  - fsync: ファイルの更新をディスクに反映する処理
  - コンテナではジャーナリングの競合が起きるのが原因
  - この辺りの分析は experience と expertise の勝利
- fsync を多用する実アプリケーションを探す
  - データベースなんかどうだろう？
  - We made it!!!

どうやって辛さを乗り切るのか？（その2）

- “空想”の問題を解いても評価されない
- 実際のシステムで起きている問題を見つけろ！
- 報告されている問題を全部調べる
  - 先入観は排除せよ
    - フィルタリングしたり絞り込んだりしてはいけない
  - 先入観があると“想定範囲内”の問題しかみつからない
    - 驚きを伴ういい仕事になりにくい

Linux のバグから学ぶ

～バグのフィールド調査からコード検査器の実現まで～

慶應義塾大学 河野健二

本日の話題

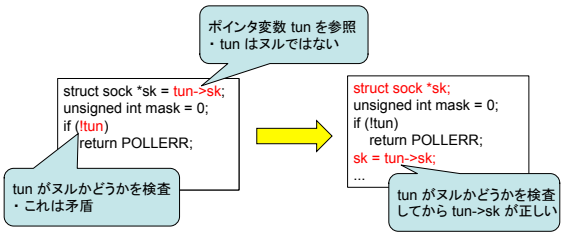
- Linux ってバグがないわけではない・・・
  - でも Linux って社会のインフラになってます
- Linux のバグを駆逐しよう
  - OS 固有の知識を活かしたコード検査
    - OS に典型的なバグ・パターンを静的に検出する
      - ヌルポインタかどうかの検査忘れ
      - unlock や free などの呼び出し忘れ
      - 暗黙の約束事を忘れている
        - » ブロックしてはいけない関数の中でブロックする関数を呼び出す
- Linux に固有のバグって？
  - これまでは Linux 開発者の経験と勘が頼り
    - Linux には “○○” というバグが多い
    - その “○○” っていうバグをとるコード検査器を作りました

本日の話題

- コード検査器の実現は古典芸能の職人芸
  - ワシの経験と勘が頼りじゃ
- 経験と勘にたよらないコード検査器
  - 膨大なバグ修正記録を活用
    - ソフトウェア工学におけるビッグデータ
  - 37 万件を超える Linux のコード修正記録を分析
    - 自然言語処理によるパワープレイ
    - Amazon EC2. 100 インスタンスで 1 ヶ月.
  - Linux に典型的なバグ・パターンの抽出
  - 割込み関連のバグ・パターンについてコード検査器を実現
  - 最新の Linux においても複数のバグを発見

Example of Linux Bugs (1)

- 初歩的なバグでさえ，まだまだ残っている
  - 例：“ヌルポインタ参照”のバグ
    - tun/tap: Fix crashes if open() /dev/net/tun and then poll() it.
    - Author: Mariusz Kozlowski <m.kozlowski@tuxland.pl>

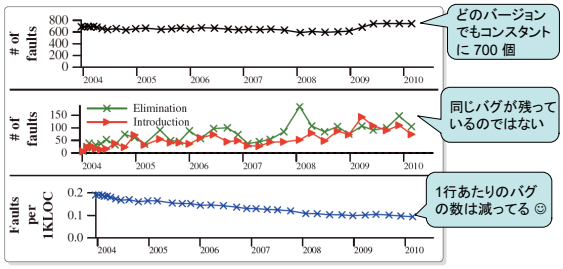


Linux における “簡単な” バグ

- 簡単な静的解析で見つかるバグを調査
  [Palix et al. 2011]
  - Null（ヌル検査忘れ）:
    - Null を返すかもしれない関数の返値のチェック忘れ
  - Inull（Inconsistent null check）:
    - ポインタ参照をした後にヌル・チェック
      - さきほどのバグの例
  - Block（Calling blocking func in non-blocking context）
    - ブロックしていはいけないコンテキストでブロックする関数を呼ぶ
      - 例:スピンロックを保持したまま，ロックを獲得する
      - 例:ファイルシステムから・・・・
  - など 12 種類のバグを検査

Linux でも“簡単な”バグがたくさんある

- どのバージョンでもほぼ 700 個のバグがある
  - Linux 2.6.0 ～ 2.6.33 までの調査 [Palix et al. 2011]



Example of Linux Bug (2)

- “簡単” ではないバグの例
  - 静的なコード検査では見つけるのが難しいもの
    - 関数にまたがった解析が必要なケース
    - 割込みなどの非同期的な振る舞いが絡むケース
    - などなど
- 割込みのタイミングに依存するバグ
  - デバイスの取り外し処理
    1. デバイスの割込みを解除する
       - 割込みを受け付けないようにする
    2. デバイス管理のためのデータ構造を開放する
  - 複雑なコードでは・・・
    - 1. と 2. の処理の順番がひっくり返ってしまうことがある

Example of Linux Bug (2)

- OS らしいバグ：割込みのタイミングに依存するバグ

```
void usb_remove_hcd(struct usb_hcd *hcd)
{                          Interrupt occur
    ……
    remove_debug_files (ohci);
    ohci_mem_cleanup (ohci);
    if (ohci->hcca) {
      …..
      ohci->hcca = NULL;
      ohci->hcca_dma = 0;
    }
    hcd->state = HC_STATE_HALT;
    if (hcd->irq >= 0)
        free_irq(hcd->irq, hcd)
    usb_deregister_bus(&hcd->self);
    hcd_buffer_destroy(hcd);
}
```

drivers/usb/core/hcd.c

```
static irqreturn_t ohci_irq (struct usb_hcd *hcd, struct pt_regs *ptregs)
{
    …
    …                    Nullポインタ参照
    if ((ohci->hcca->done_head != 0)
            && ! (hc32_to_cpup (ohci, &ohci->hcca->done_head)
                    & 0x01)) {
    …
}
```

Interrupt occur　　drivers/usb/host/ohci-hcd.c

Linux 2.6.18
id: 71795c1df30b034414c921b4930ed88de34ca348
で報告されている

簡単な静的解析では見つからない

Example of Linux Bugs (3)

- Many Linux device drivers assume device perfection [Kadav et al. 2009]
- Example: Infinite polling
  - Driver waiting for device to enter particular state
  - If device not working correctly, the loop never ends. Hang

```
static int amd8111e_read_phy(……)
{
 ...
  reg_val = readl(mmio + PHY_ACCESS);
  while (reg_val & PHY_CMD_ACTIVE)
        reg_val = readl(mmio + PHY_ACCESS)
  .
}
```

AMD 8111e network driver(amd8111e.c)

Example of Linux Bugs (3)

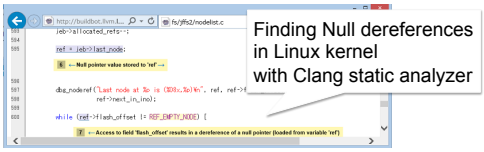
- Solution: add the code for timeout
  - If timeout occurs, recover code is invoked

```
static int amd8111e_read_phy(……)
{
 ...
  timeout = 0;
  reg_val = readl(mmio + PHY_ACCESS);
  while (reg_val & PHY_CMD_ACTIVE) {
        reg_val = readl(mmio + PHY_ACCESS)
      if (timeout++ >= 200)
            __shadow_recover();
  }
  .
}
```

AMD 8111e network driver(amd8111e.c)

Eliminating bugs

- Bugs in software have to be eliminated
  - To avoid security issues and low availability
  - Developers do a lot of debugging efforts
    - Code review, testing, maintenance etc.
- Static code checkers help developers find *typical* bugs



Why "typical" bugs?

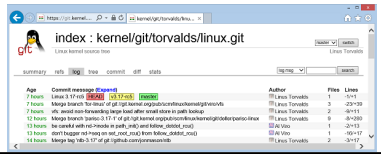
- Focusing on typical bugs is reasonable
  - People make the same mistakes as others have done
- Examples of typical bugs:
  - Pair API misuses e.g., alloc/free, lock/unlock
    - [Saha et al. '13], [Palix et al. '11]
  - Unhandled device failures e.g., infinite polling
    - [Kadav et al. '09]

Who writes what checkers?

- Knowing typical bugs is difficult
  - Bugs are human mistakes
    - Hard to predict typical developers' behaviors
  - Many bugs are domain-specific
    - >50% of bugs in Linux file systems are violations of file system semantics [Lu et al. FAST'13]
    - Hard to understand semantics in large-scale software

Learning from bug repositories

- Challenge: Recognizing many & similar patterns
  - Hard to understand & summarize many documents
- Bugs are often documented in English
  - E.g., >370,000 patches in Linux kernel
  - Developers can extract typical bugs

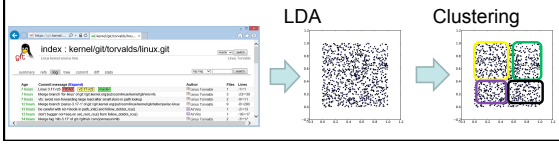


Goals

- Use machine learning to extract typical patch documents in Linux kernel
  - Many & similar patches are extracted
- Extract bug patterns from the extracted typical patch documents
- Develop checkers for the extracted bug patterns
- Apply the checkers to the latest Linux kernel

Extracting many & similar patches

- Natural language processing calculates the similarity of patches
  - Latent Dirichlet allocation (LDA) [Blei et al. '03]
- Clustering groups similar patches
  - Recursively divides clusters by 2-means
  - Enables us to extract large groups (5,000 - 10,000) of similar patches



LDA in short

- LDA infers latent topics in documents
  - A document is regarded as probability sets of topics
    - The similarity of two documents is the distance between the probability sets for them
  - Keywords characterizing patches can be obtained

In function devkmsg_read/writev/llseek/poll/open()..., the function raw_spin_lock/unlock is used, there is potential deadlock case happening. CPU1: thread1 doing the cat /dev/kmsg: raw_spin_lock(&logbuf_lock); while (user->seq == log_next_seq) { when thread1 run here, at this time one interrupt is coming on CPU1 and running based on this thread,if the interrupt handle called the printk which need the log buf_lock spin also, it will cause deadlock.

LDA: $p(D|\alpha,\beta) = \prod_{d=1}^{M} \int p(\theta_d|\alpha)(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn}|\theta_d)p(w_{dn}|z_{dn},\beta))d\theta_d$, $p(\theta,z|w,\alpha,\beta) = \frac{p(w|\alpha,\beta)}{p(\theta,z,w|\alpha,\beta)}$, $p(w|\alpha,\beta) = \int p(\theta|\alpha)(\prod_{n=1}^{N} \sum_{z_n} p(z_n|\theta)p(w_n|z_n,\beta)d\theta$

topic A: 0.113, topic B: 0.055, topic C: 0.04 topic D: 0.038, topic E: 0.038, topic F: .....
Keywords: lock, unlock, spin, lock, protect,

Analyzing Linux patch documents

- 370,403 patch documents are analyzed
  - Linux 2.6-rc2 ~ 3.12-rc5 (April 2005 – October 2013)
  - Merge commits are excluded
  - Analyzer has been implemented on Hadoop MapReduce
  - LDA from Apache Mahout
- Result: 66 clusters
  - Linux had topics for general software bugs, OSs, devices, CPU platforms, etc.

Result 1/3: common bugs

- Clusters for bugs in general software
  - Null dereferences, memory leaks, lock/unlock
  - Typical software bugs in the Linux kernel
- Example document: memory leak
  - Topic keywords: memory, leak, cpu,hotplug
  - Misuse of kmalloc() and kfree()

commit 003615301, 2nd paragraph

USB: io_ti: fix port-data memory leak
Fix port-data memory leak by moving port data allocation and deallocation to port_probe and port_remove.

Result 2/3: hardware

- Clusters for CPU platforms and devices
  - ARM, X86, GPU, USB, NIC, etc.
  - Major hardware-speicifc issues
- Example document: ARM
  - Topic keywords: arm, mach, h, asm
  - Problems deriving from ARM features

commit 9cff337, 3rd paragraph

So far as I am aware this problem is ARM specific, because only ARM supports software change of the CPU (memory system) byte sex, however the partition table parsing is in generic MTD code.

Result 3/3: common OS features

- Clusters for common OS features
  - Interrupt handling, buffer cache, DMA, etc
  - Typical implementation issues in OSs
- Example document: interrupt handling
  - Topic keywords: irq, interrupt, msi
  - Problems around masking interrupts

commit ea6dedd, 2nd paragraph

The current OMAP GPIO IRQ framework doesn't use the do_edge_IRQ, do_level_IRQ handlers, but instead calls do_simple_IRQ. This doesn't handle disabled interrupts properly, so drivers will still get interrupts after calling disable_irq. ....

Extracting bug patterns from a cluster

- The cluster for interrupts are expected to typical bugs in OSs
  - The cluster remains too large (5,334 patches)
- Topic keywords help us extract interesting sub-clusters
  - A sub-cluster with keyword "free" is expected to have bugs around free and interrupts
- The sub-cluster with keyword "free" contains 364 patches
  - 160 are identified as bugs

Result: the misuse of free_irq() is common

	Description	Num.
1:	free_irq() with inconsistent device ID	41
2:	missing free_irq() on initialization error path	25
3:	free_irq() with invalid irq	25
4:	missing free_irq() on module unloading	13
5:	double free_irq()	9
6:	freeing other src before free_irq()	7
7:	freeing pages with interrupt disabled	7
8:	missing free_irq() before suspend	6
9:	freeing shared irq with interrupt enabled	5
	Other (most contain free and irq)	22
	Total	160

Developing checkers

- A domain-specific checker for free_irq() misuse
  - Checks the consistency of two arguments
    - Interrupt number and device ID
  - Checks a typical life cycle of PCI device drivers
    - Probe, suspend, resume, remove, shutdown, etc.
  - Runs on the Clang static analyzer

Checking Linux 3.15

- 2 bugs are found across 593 PCI drivers

```
1204    /* We must finish initialization here */
1205
1206    if (!socket->cb_irq || request_irq(socket->cb_irq, yenta_interrupt, IRQF_SHARED, "yenta", socket)) {
            (24) ← Taking false branch →
1250    } else {
1251        socket->socket.features |= SS_CAP_CARDBUS;
1252    }
1262    /* Register it with the pcmcia layer.. */
1263    ret = pcmcia_register_socket(&socket->socket);
1264    if (ret == 0) {
            (25) ← Assuming 'ret' is not equal to 0 →
            (26) ← Taking false branch →
1272    }
1273
1274 unmap:
1275        iounmap(socket->base);
1276 release:
1277        pci_release_regions(dev);
1278 disable:
1279        pci_disable_device(dev);
1280 free:
1281        kfree(socket);
1282 out:
1283        return ret;
1284 }
```

1, request_irq() succeeded

2, pcmcia_register_socket() fails

3, Freeing src although interrupts may be delivered

4, A device probe fails without free_irq()

どうやって辛さを乗り切るのか？（その3）

- “正しさ” を証明する方法がない
  - 例えば，アルゴリズムの正しさを証明できない
  - 実装の正しさはもっと証明できない
- “良さ” を客観的に比較する方法がない
  - Linux と Windows. どちらがより良い OS か比較できない
- パワープレイで（査読者を）黙らせるしかない
  - “正しい”ことを示すかわりに・・・
    たくさんのベンチマークを動かしてみせる
  - “良さ” を示すかわりに・・・
    たくさんのシステムと比較実験をしてみせる

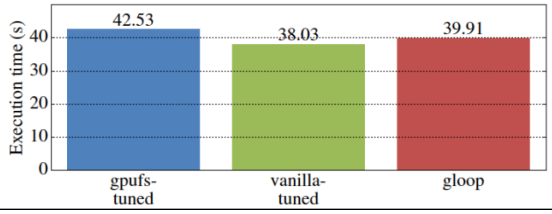
Towards Multi-tenant GPGPU: Event-driven Programming Model for System-wide Scheduling on Shared GPUs

in collaboration with
Yusuke Suzuki*
Hiroshi Yamada**, Shinpei Kato***

* Keio University
** Tokyo University of Agriculture and Technology
*** University of Tokyo

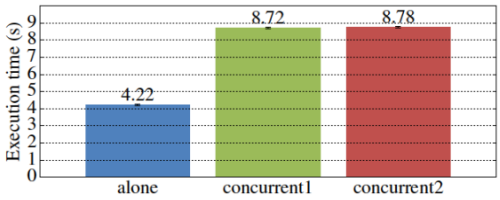
Case Study: grep_text

- Port GPUfs grep_text application
  - To show GLoop can represent GPU eaters with reasonable performance
- GLoop’s grep_text performance is comparable to one of GPUfs’ and CUDA vanilla one



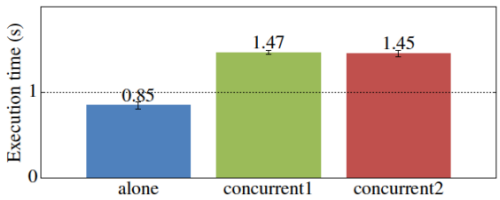
Consolidation: throttle

- To show GLoop can interleave kernel executions among GPU eaters
- Run 2 instances of throttle GPGPU app
  - Throttle performs event loop callbacks 10000 times
- Execution times of concurrent ones are almost doubled compared to one of alone one (2.07x and 2.08x)



Consolidation: copy1GB

- Run 2 instances of copy1GB
  - Copy1GB performs 1GB file copy using GLoop I/O ops
  - File contents are cached in OS buffer cache
    - Copy1GB performs frequent Host-Device memory transfers
- Interestingly, concurrent ones are faster than doubled time of alone one
  - 1.73x and 1.71x



実験，たいしたことないじゃん

- それはプレゼンだから
- 実際の論文では・・・

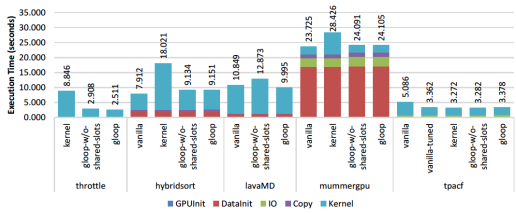


Figure 6: Execution times and breakdown of applications.

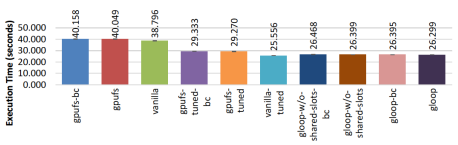


Figure 7: Execution times of grep applications.

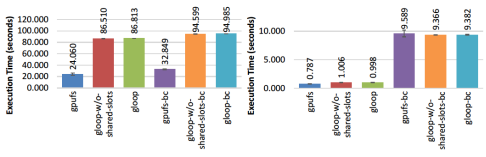


Figure 8: Execution times of img variations.

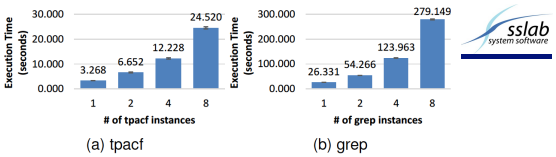


Figure 9: Performance across multiple application instances.

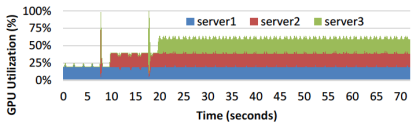


Figure 14: Stacked GPU usage of consolidated GPU matrix multiplication servers (over 200ms).

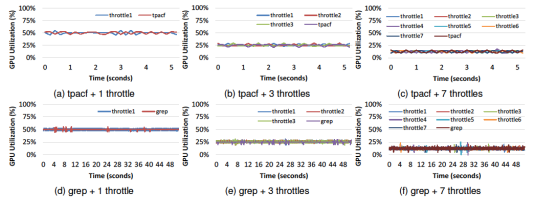


Figure 10: GPU usage of GPU tasks and concurrently running throttles (over 200 ms).

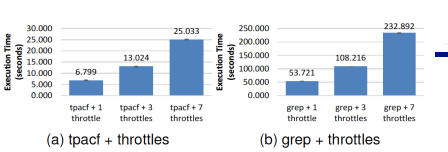


Figure 11: Execution times of GPU tasks with 1, 3, and 7 throttles.

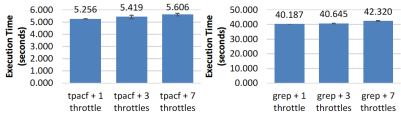


Figure 13: Execution times of GPU tasks with 1, 3, and 7 throttles. 66% of GPU computing resource is assigned to GPU tasks.

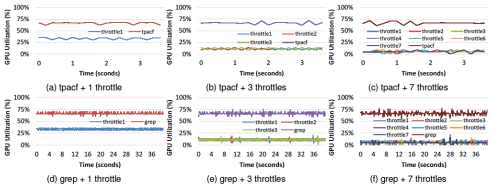


Figure 12: GPU usage of GPU tasks and concurrently running throttles (over 200 ms). 66% of GPU computing resource is assigned to GPU tasks.

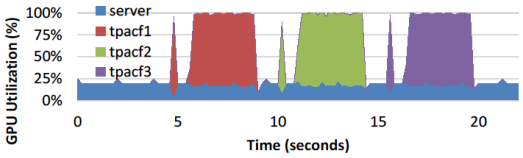


Figure 15: Stacked GPU usage of GPU matrix multiplication server and tpacf application (over 200ms).

Take-Away Message & Conclusion

- システムソフトウェア研究の変遷
  - 研究テーマは自然に決まる
    - システムソフトウェアは・・・
      - 上司（アプリ）にゴマするイエスマン
      - 部下（ハード）のワガママは全部聞く

      無能な中間管理職でよい
  - 研究のやり方はどんどんシビアになっている
    - 実システムで起きている現実の問題に目を向ける
      - 問題発見のための努力を惜しまない
    - 提案方式は，既存システムに組み込んでナンボ
    - 性能評価は徹底的に！
      - できる実験は全部やる