2017.1.30 SRE Tech Talks #2

SREグループができてこの半年間やってきたこと

XFLAG STUDIO

ゲーム開発部 SREグループ　清水 勲 @isaoshimizu

About me

- 清水 勲 @isaoshimizu
- 2011.8-2014.3 SNS mixiの運用
- XFLAG スタジオ
    - 2014.4-2016.6 サーバーエンジニア
    - 2016.7- SRE
    - 主に国内向けモンスターストライクを支えるお仕事
    - 他にもモンストスタジアム、ブラナイDASHなど
- 好きなもの
    - Linux、MySQL、nginx、Memcachedなどのミドルウェア全般、Golang、クラフトビール 🍺
- 2016.3.1 dots. Conference Spring 2016「モンストを支えるインフラの今とこれから」を発表
    - https://speakerdeck.com/isaoshimizu/monsutowozhi-eruinhurafalsejin-tokorekara

XFLAG スタジオにおけるSRE

XFLAG スタジオにおけるSRE

- 2016年7月にSREグループが創設
- SREになったからといって突然大きく変わったことはなく、従来から継続してやっていることも多い
- SREのチームにおけるミッションがわかりやすくなった
    - 社内＆社外からもわかりやすい組織体制
    - ゲームに関わる機能開発からの分離
    - 負荷対策、効率化、自動化などに注力
- メンバーの得意不得意を相互に補完しながら作業をこなす
- 当番制で不測の事態に備える
- 基本業務
    - インフラ・サーバー運用、可用性・パフォーマンス向上、ログ収集、開発環境整備、セキュリティ対策、ツール開発
    - 本番/ステージング環境へのデプロイ、ゲームデータのインポート
        - ステージング以前の開発環境は開発者自身がSlackのボット経由でデプロイ・インポート可能
    - グラフを見る、ログを読む、コードを理解する、GitHub Issueで問題を共有する、Pull Requestで解決する

当番制

- 当番は計11名（SREのメンバーは現在7名）
- 2名体制で毎週ローテーション
- GWや年末年始などの連休の時は1日単位で交代
- 基本的に定時外と休日は当番が一次対応する
- Nagiosで障害を検知、PagerDutyと連携して当番へ通知
- 障害に限らず、エンジニア以外から緊急の連絡が受けられるように、Slackからボット経由で当番へ通知できる仕組みもある
- 場合によってはエスカレーションして総動員で対応する
- 今の当番制ができたのは5年以上前らしい

インフラ構成

オンプレとクラウドのハイブリッド構成

- モンスターストライク（日本版）の話です
- インフラは複数DCでオンプレミス（物理サーバ）が主体
  - グローバル版や他案件ではAWSが多い
- クラウドも積極的に利用
  - おもに本番Appサーバ用途として計数百台規模のインスタンス（時期によって変動）
  - AWS c4.4xlarge (16core, 30GB)
  - GMOアプリクラウド N-4096 (40core, 96GB)
  - それ以外ではRoute 53, CloudFront, Turnサーバ, 開発環境, Webサイト系など
- 現在は本番以外も含めて全体で1,300台前後のサーバを運用

最近取り組んだことの例

年末年始対策の一例

年末年始対策の一例

- Appサーバの増強
    - 総コア数換算（論理コア）で10,000コアを15,000コアに一時的に増強
    - ワーカー数の調整
- データベース
    - 負荷の高いDBをシャーディング
    - クエリ発行数を減らす
- スケールアップ
    - ioDrive, ioMemoryへの入れ替え
    - 古いマシン、インスタンスのリプレース
- チューニング（後述）
    - MySQL(InnoDB)の設定の最適化
    - OSのアップデート
- うるう秒対策
    - FM電波式のNTPサーバ（アプライアンス）の利用（外部サービスに依存しない）
    - 2017年1月1日9:00の6時間前から徐々に遅らせて1秒間を調整する設定（LIは挿入させない）

MySQLチューニング

MySQL

- MariaDB 5.5系で統一
- Chefでプロビジョニング
- ディスクはSSD or ioDrive/ioMemoryの2パターン（SASは無い）
- 当然ながらInnoDBのよくあるような設定は一通りやってある（SNS時代から改善を続けてきたmy.cnf）
- 今回、主にチューニングしたの設定は2つ（次ページへ）

MySQL(InnoDB) チューニング①

- innodb_io_capacity
  - SSD, ioDrive/ioMemoryでそれぞれ固定値を設定してきたが、
    用途によって変える余地があった（グラフ観察により）
  - Dirty pages rate, Checkpoint Age, Disk IO, CPU Usageあたりの
    バランスを見ながら最適値を決める
  - 未使用のSlaveで試してからMasterに適用
  - ioDriveで10,000から20,000に変えた事例も
    - 結果としてレプリケーション遅延を緩和する効果や、spin lockの回数が
      減る効果が得られた

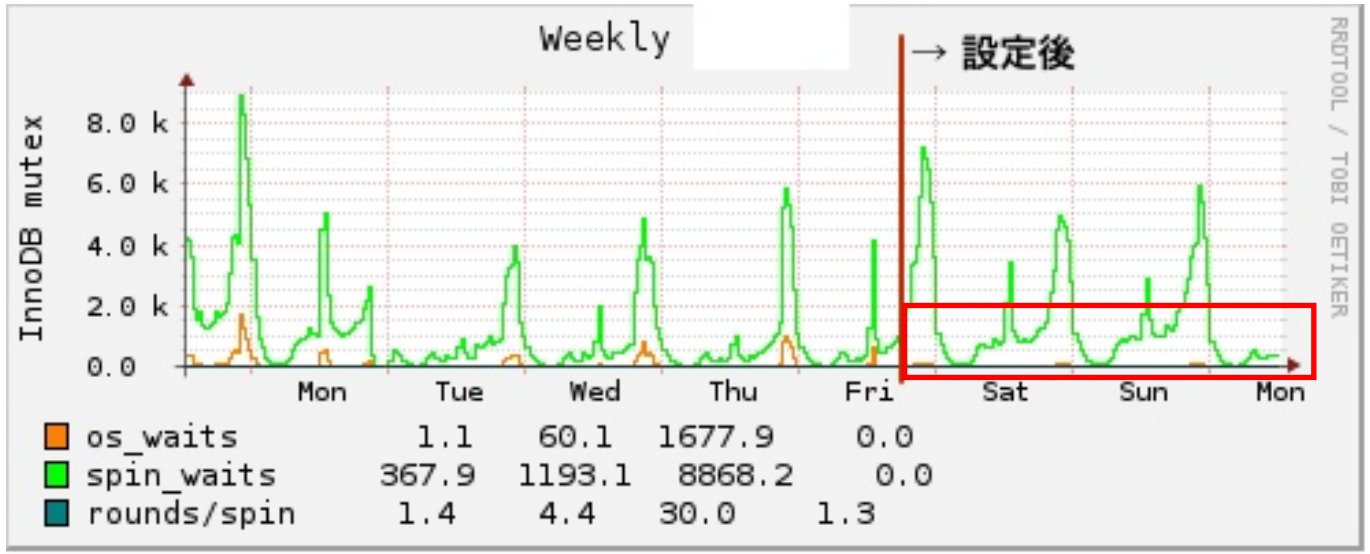
MySQL(InnoDB) チューニング②

- innodb_spin_wait_delay
  - いままでデフォルト値 6 を使ってきた（要は未設定）
  - ここからMySQLにおける同期処理の話
  - スレッドの同期処理にspin lockを優先して使っている（ビジーウェイトとも言う）
  - 通常のmutexはロックを取得できない場合シグナルを待ち続けるが、spin lockはループによってロックを取得しつづける
  - スレッドはロックが使用可能かどうかチェックをするためにループしつづける
  - ただし、ロックの解放を待っている間、ループしている分CPUを使い続けるので、マルチコアの環境かつ、短時間でロックが解放されるケースに有効な手法といわれている
  - ロック待ちのループの回数がinnodb_sync_spin_loopsを超えた場合は、OSのmutex(pthread_cond_wait)に引き継がれる
  - 一般的にOSが提供するmutexを使う方がコストが高いとされている

MySQL(InnoDB) チューニング③

- MySQLのspin lockでは、innodb_sync_spin_loops 分ループが回るまでロックの取得を試行するが 0 から innodb_spin_wait_delay (default 6) の範囲でランダムに待ち時間を設けている
- MariaDBのコード例

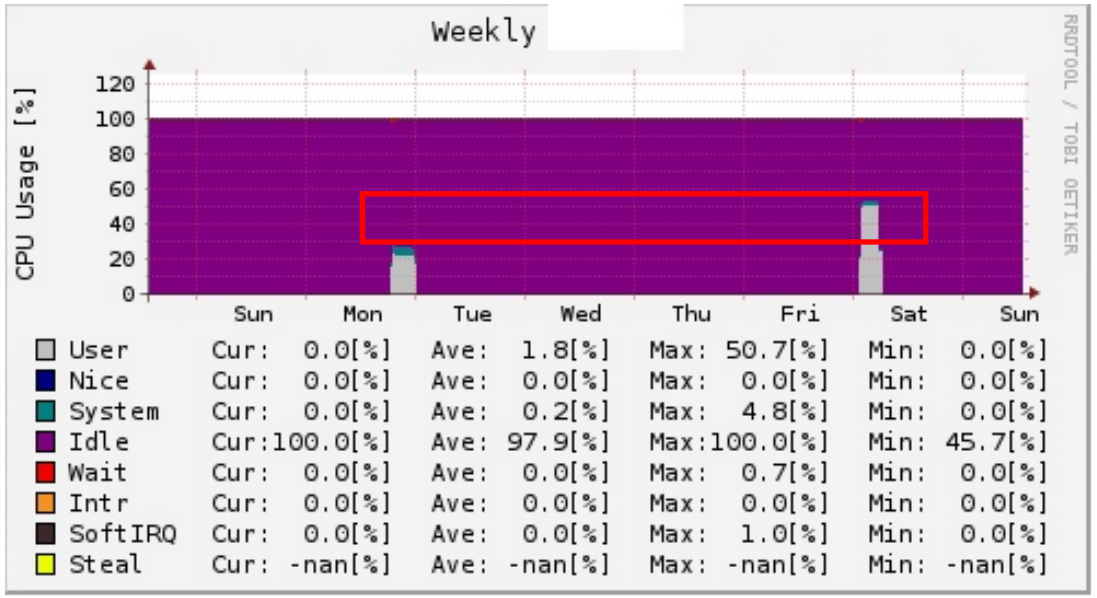
  https://github.com/MariaDB/server/blob/mariadb-5.5.54/storage/innobase/sync/sync0sync.c#L527
- innodb_spin_wait_delay=50を設定したときの実際のグラフの例

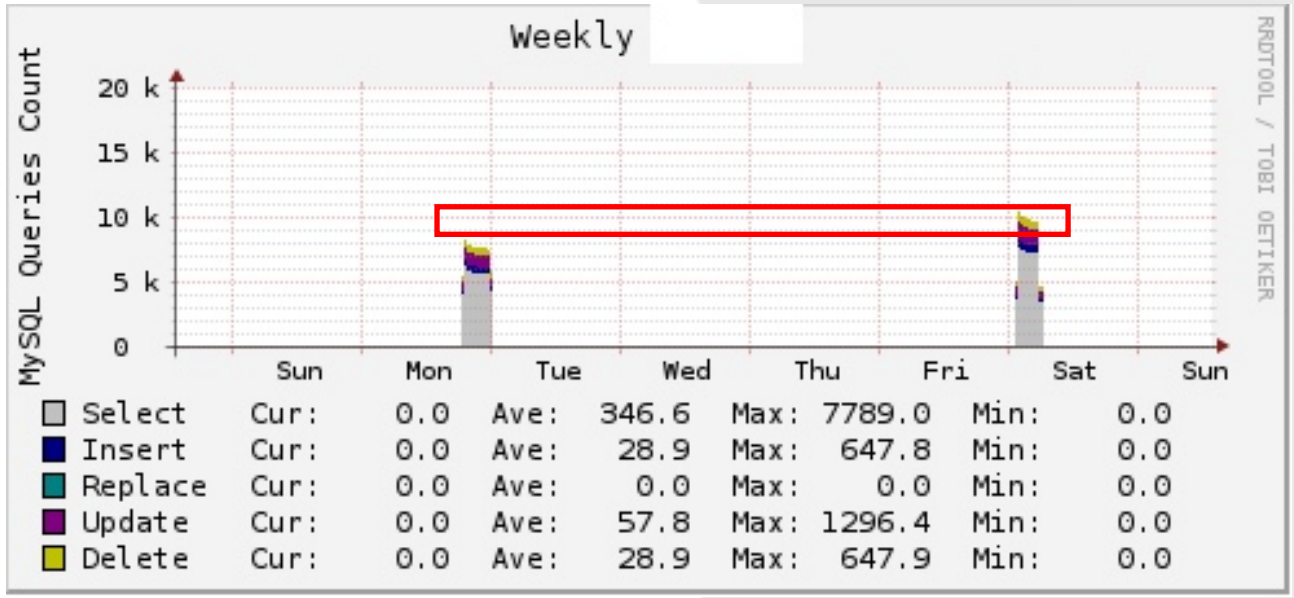


設定後はos_waitsが減っている

MySQL(InnoDB) チューニング sysbench結果①

- innodb_spin_wait_delayのチューニングはsysbenchの結果にも好影響が出た



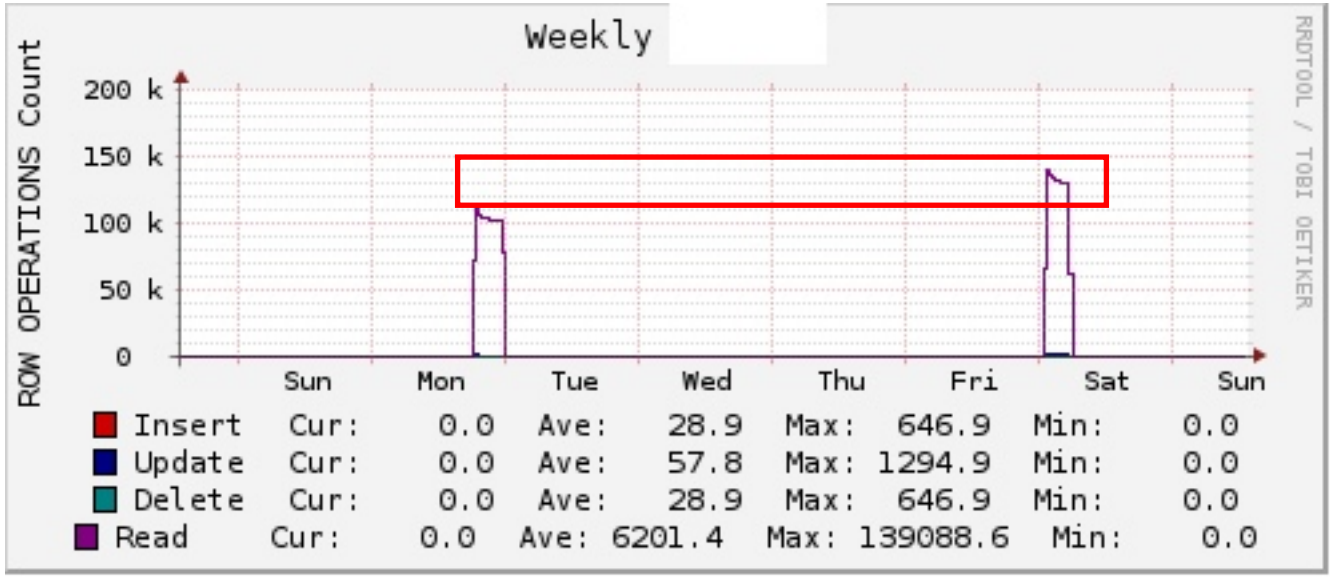


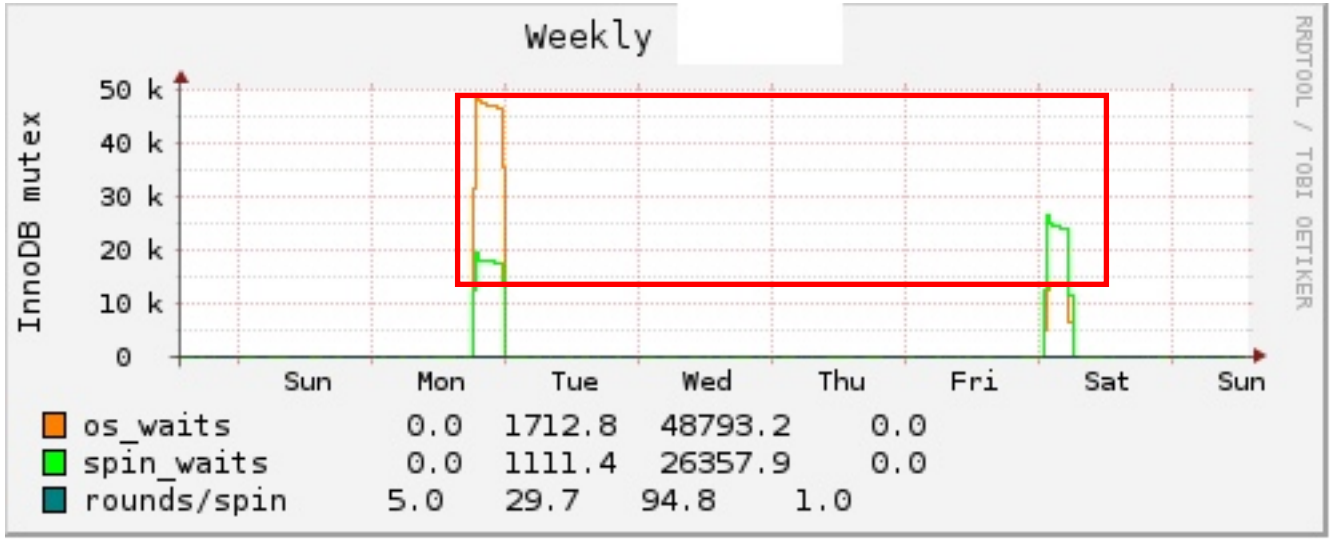
左（月曜日のグラフ）が設定前（デフォルト値）、右（土曜日のグラフ）が設定後

※CPU UserのUsageが増えたのはspin loop時の待ち時間が増えたため（実運用でこれほどCPUを食うことはほぼ無い）

※実施マシンのスペック: Xeon X5650 2.67GHz x 2(24core), Memory 64GB, SSD 480GBx2(LVM)

MySQL(InnoDB) チューニング sysbench結果②





左（月曜日のグラフ）が設定前（デフォルト値）、
右（土曜日のグラフ）が設定後

ベンチ結果の数値(約22%改善)

total time: -> 20401.1135s -> 15875.2834s
read/write requests: 7847.98 -> 10082.02/s
transactions: 490.21/s -> 629.96/s

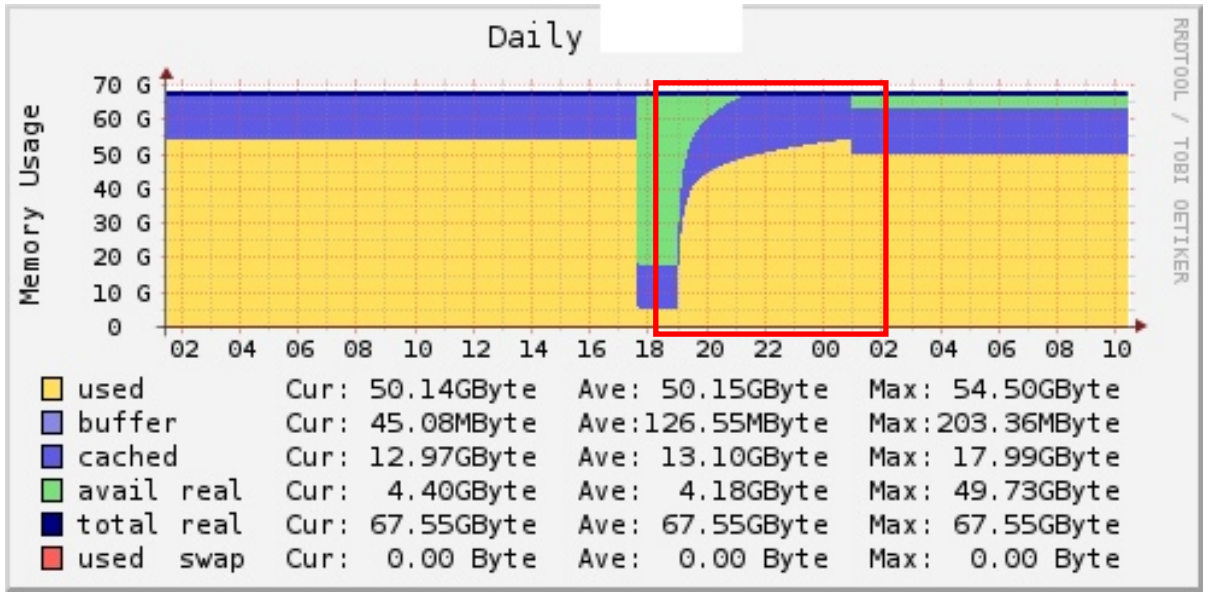
OSアップデート

OSのアップデート

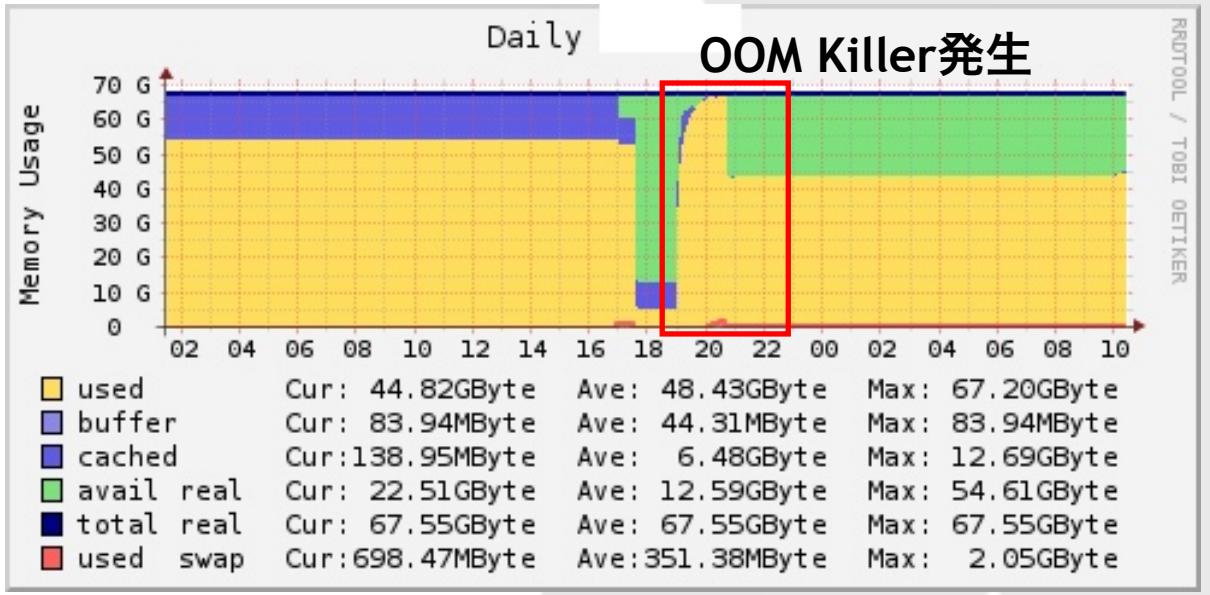
- 2013年にサービス開始した当初はUbuntu Server 12.04 LTS
- 現在はUbuntu Server 14.04 LTSを利用
- 2017年4月にEOLを迎える12.04から14.04へ徐々に入れ替えている
- 16.04はsystemdへの対応や、Kernelの差分が大きいことから作業コストがやや高そうだったので一旦見送り（早い段階でアップデートしたい）
- 14.04を使い始めるときに起きたこと
  - Transparent Huge Page問題
  - パフォーマンスの変化

Transparent Huge Page問題①

- Kernelバージョンの差で起きるメモリ問題
- 以下のグラフはKernel 3.8と3.13の環境でsysbenchを実行したときのメモリ使用量



Kernel 3.8



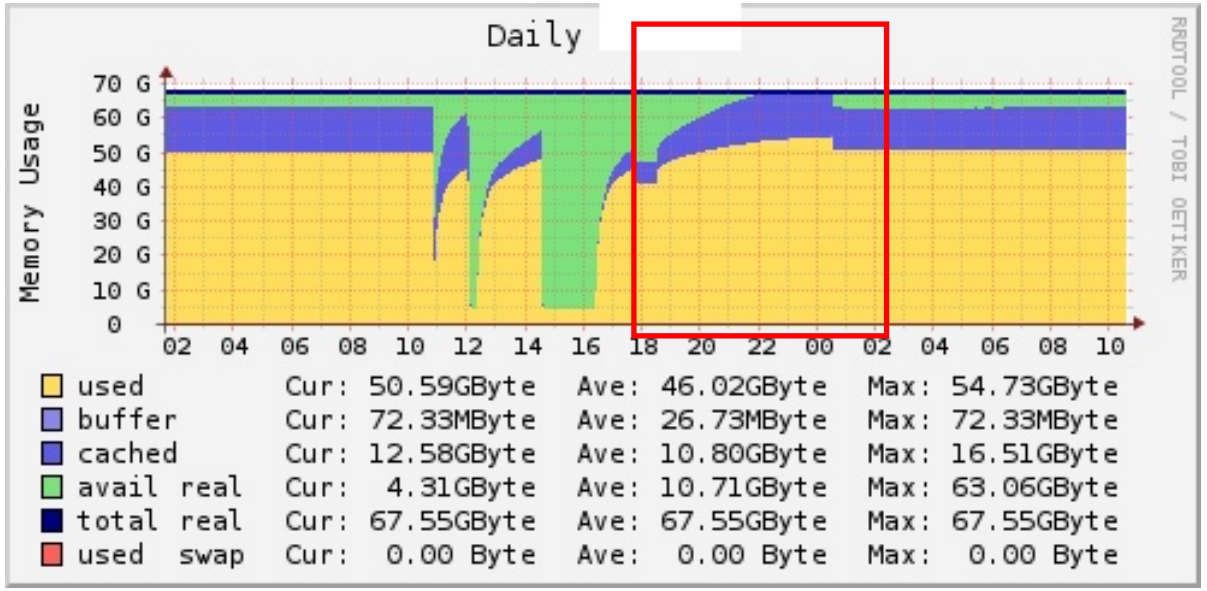
Kernel 3.13

Transparent Huge Page問題②

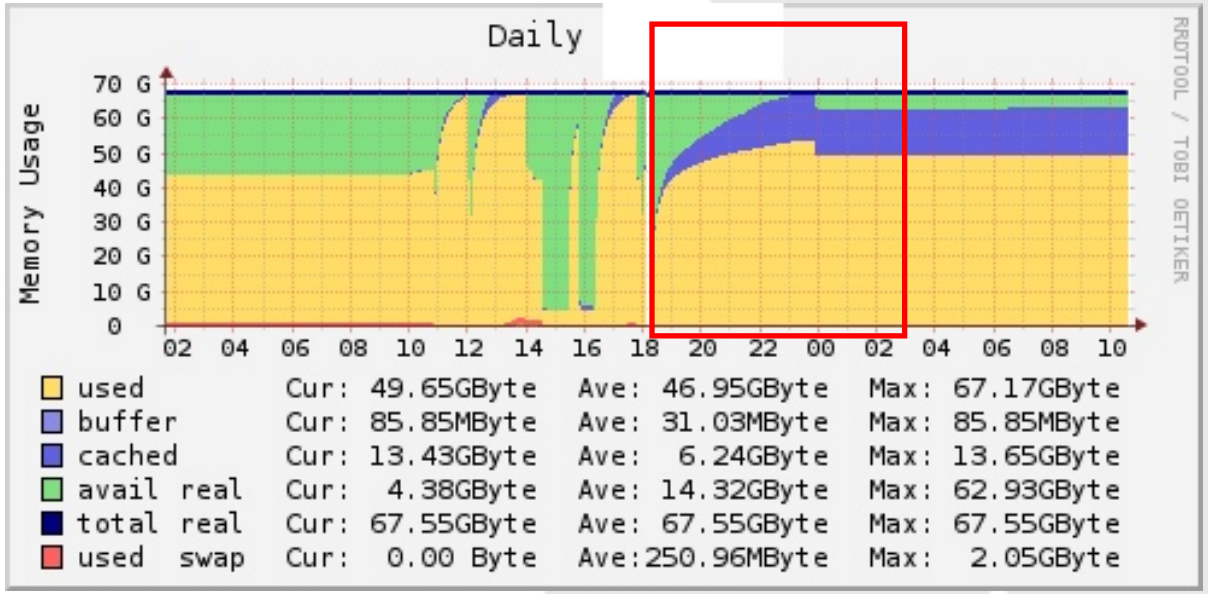
- Kernel 3.13以降でMySQLなどのメモリを大量に使うケースにおいて、メモリ使用量が膨らむのは Transparent Huge Pages(THP)が影響していることが判明
- THPはLinux 2.6.38でマージされたもの（2011年1月）
  - http://kernelnewbies.org/Linux_2_6_38#head-f28790278bf537b4c4869456ad7b84425298708b
- /sys/kernel/mm/transparent_hugepage/enabled の値でTHPの有効・無効が制御できる
- Ubuntuの場合、Kernel 3.8では madvise、Kernel 3.13以降では always がデフォルト値として設定されるため、3.13以降ではTHPが常に有効となる
  - madvise では明示的にTHPを使うようにしない限り使うことはない
  - never を指定するとTHPは使わない
- RHELのPerformance Tuning Guide https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-memory-transhuge.html
  - “THP is not recommended for database workloads.”
- THPが有効の場合、/proc/meminfoのAnonHugePagesの値が増える
- 実はMongoDBではおなじみの設定
  https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/
- Brendan Gregg先生のスライドにはだいぶ助けられた
  http://www.slideshare.net/brendangregg/linux-systems-performance-2016

Transparent Huge Page問題③

- Kernel 3.13で、/sys/kernel/mm/transparent_hugepage/enabled の値を neverにして sysbenchを実施



Kernel 3.8 (madvise)



Kernel 3.13 (never)

Transparent Huge Page問題④

- 検証でおこなったsysbenchについて
  - sysbench --test=oltp.lua --oltp-table-size=300000000 --mysql-table-engine=innodb --num-threads=10000 --max-requests=10000000 --max-time=0 --oltp-test-mode=complex --mysql-host=自マシンのIP ー-mysql-user=benchuser --mysql-password=benchpass run
  - oltp.luaはSUMとDISTINCTのクエリを削除（実運用では使わない関数のため）
- かかった時間
  - Kernel 3.8: 362分
  - Kernel 3.13: 361分
  - Kernel 4.2: 313分
- 4.2だけやたらと速かった
- スレッド単体のベンチマーク（ sysbench --test=threads --num-threads=$NUM_THREADS --max-time=0 run を5回実施した平均値）
  - Kernel 3.8: 5.2s
  - Kernel 3.13: 6.0s
  - Kernel 3.16: 3.5s
  - Kernel 4.2: 4.0s
- Kernel 3.16以降スレッドまわりが改善された（？）

補足: NUMAとTHPに関わるメモリ問題①

- 最近のマシンはCPUごとにメモリが区分けされて、CPUに近いメモリに対して高速にアクセスできるようになった（NUMA=Non-Uniform Memory Access）。
- NUMAにおけるデフォルトの挙動としては、プロセスが利用するCPUから近いメモリノードが優先的に使われていく。大量にメモリを使うケースなどで、近いノードを使い尽くしてしまうと他方の遠いメモリノードを使いはじめるが、swapが発生してしまう問題がある（swap insanityと呼ばれる)。swapの発生によってパフォーマンスは劣化してしまう。
- 明示的にinterleaveポリシーを使うことでCPUに近いメモリノードを使うのではなく、ラウンドロビンにメモリノードを選択することが可能。これは、numactlコマンドの--interleveオプションを指定して実行することで同一プロセスでinterleaveポリシーが実現できる。メモリを大量に使うケースではswapを防ぐことができるため、これが推奨される。

補足: NUMAとTHPに関わるメモリ問題②

- THPはページサイズを4KBから2MBに大きくすることで、PTE(Page Table Entry)のキャッシュであるTLB(Translation Lookaside Buffer)利用の効率化が図られるが、サイズが大きくなる分メモリ確保に時間かかかるようになり、2MB以下のメモリ確保時に無駄が生まれるというデメリットがある。
- NUMAにおいて、ページサイズが4KBのときに2MB確保した場合は、CPUに近いノードに分散してメモリが確保されるが、ページサイズが2MBで2MB分確保すると、特定のノードにしか確保されなくなり、遠方のCPUからアクセスされる際にオーバーヘッドが生まれることがある(http://www.fabiengaud.net/resources/gaud14large-slides.pdf)。
- スレッドやプロセスを参照しているメモリの近くに移行したり、参照しているデータを近くのメモリに移行するのを自動的にやってくれるNUMA Auto Balacingという機能があるが、移行することによってPage Faultが増加するケースがある。
- zone_reclaim_modeが有効になっていると、CPUに近いメモリを積極的に使うようになるが、そのためにキャッシュを捨てることも辞さない。memoryのnodeの距離を見て、遠いと自動的に1とするやっかいなものなので、memoryを沢山使うようなworkloadでは固定的に0にするのが常套手段(zone_reclaim_mode = 0)。
- 以上より、NUMA、特にメモリを使うデータベースにおいては、THP、NUMA Auto Balancing、zone_reclaim_modeを無効化しないと不利益なことが多い。

補足: NUMAとTHPに関わるメモリ問題③

- 実際の設定値
  - echo never > /sys/kernel/mm/transparent_hugepage/enabled
  - echo 0 > /proc/sys/kernel/numa_balancing
  - zone_reclaim_mode = 0

THPとNUMA Auto BalancingはGRUBの設定で無効化できる

- /etc/default/grub に transparent_hugepage=never numa_balancing=disable を追記してupdate-grubする
- OSの初期設定時にこれらを設定している

補足: NUMAとTHPに関わるメモリ問題④

Amazon Linux AMI 2016.09.1.20170119 x86_64 HVM (Kernel 4.4.41-36.55)

$ cat /sys/kernel/mm/transparent_hugepage/enabled

always [madvise] never

Red Hat Enterprise Linux 7.3 (HVM) (Kernel 3.10.0-514)

$ cat /sys/kernel/mm/transparent_hugepage/enabled

[always] madvise never

CentOS Linux 7 (Kernel 3.10.0-327.10.1)

$ cat /sys/kernel/mm/transparent_hugepage/enabled

[always] madvise never

Ubuntu Server 16.04 LTS (HVM) (Kernel 4.4.0.53)

$ cat /sys/kernel/mm/transparent_hugepage/enabled

[always] madvise never

まとめ

まとめ

- この半年くらいに起きたこと、対策したことの一部を紹介しました
- MySQLの挙動はできる限り可視化してチューニング
- NUMAやTHPまわりの問題は要注意（無効化が無難）

- SREのお仕事の幅はとても広く、やれることはまだまだたくさんあります
- 大規模ならではのやりがいもたくさんあります
- 一緒にシステムを改善していきませんか？
  - https://xflag.com/recruit/

ちょっとだけ宣伝

- 【DMM GAMES主催！】「複雑・大規模webサービスを支える技術勉強会」
- 2017年2月7日(火) 19:30 〜 22:00、場所はここドッツで！
- https://eventdots.jp/event/610781 （抽選制）
- 『大きくなったシステムを改善していくということ』というタイトルで、
  弊社 SREグループ リーダーの伊藤が登壇します。ぜひご登録ください。

参考文献

- Linux Systems Performance 2016 http://www.slideshare.net/brendangregg/linux-systems-performance-2016
- Performance Analysis and Tuning - Part 1 http://www.slideshare.net/tommylee98229/shak-larryjederperfandtuningsummit14part1final
- Optimizing Linux Memory Management for Low-latency / High-throughput Databases https://engineering.linkedin.com/performance/optimizing-linux-memory-management-low-latency-high-throughput-databases
- zone_reclaim_modeのデフォルト値が変わります http://mkosaki.blog46.fc2.com/blog-entry-936.html
- MySQL Performance: InnoDB IO Capacity & Flushing http://dimitrik.free.fr/blog/archives/2010/07/mysql-performance-innodb-io-capacity-flushing.html
- MySQLやSSDとかの話 後編 http://www.slideshare.net/takanorisejima/mysqlssd-56045479
- InnoDB の mutex の話（入門編） http://labs.gree.jp/blog/2015/12/15043/
- show innodb status http://www.slideshare.net/myfinder/show-innodb-status-6345058
- Dead Lock Analysis of spin_lock() in Linux Kernel (english) http://www.slideshare.net/SneekerYeh/dead-lock-analysis-of-spin-lock-in-linux-kernel-english
- 仮想メモリーを支えるもうひとつのキャッシュ TLB http://ascii.jp/elem/000/000/567/567889/

Thank you!