perfを使ったPostgreSQLの解析 ～後編

2013年2月9日（土）

NTT DATA

NTT DATA

◆ 後編

1. 前編の復習
2. perfを使ったプロファイリングの流れ
3. PostgreSQLをプロファイリングしてみよう
4. perfでハマった点
5. perfの使いどころ
6. Perfこうなったらいいな
7. まとめ

NTT DATA

◆ 前編

1. はじめに
2. プロファイリングとは？
3. perfとは
4. perfの仕組み
5. perfと他のパフォーマンス解析ツールとの比較
6. perfの導入と使い方
7. perfを使ってみよう
8. まとめ

0.はじめに

あらためまして自己紹介

NTT DATA

氏名

江川 大地

所属

NTTデータ
基盤システム事業本部
システム方式技術ビジネスユニット

Twitter

@daiti0804

やっていたこと

年代	やったこと
-2009	歴史のお勉強(大学時代)
2009-2011	Webシステム開発(Javaなど)
2011-現在	PostgreSQL関係

近況

1ヶ月くらいインドに行ってました。

本日のお題

お題	perfを使ったPostgreSQLの解析（後編）
今日説明すること	perfを使っての解析

参考資料

NTT DATA

- 本日の講演に当たって参考にした資料等
  - perf ソースコード
    - [Linuxカーネルソースのルート]/tools/perf配下
  - perf ドキュメント
    - [Linuxカーネルソースのルート]/tools/perf/Documentation
  - perf wiki
    - https://perf.wiki.kernel.org/index.php/Main_Page
  - 『Linuxカーネル Hacks
    ―パフォーマンス改善、開発効率向上、省電力化のためのテクニック』
    高橋 浩和 (監修), 池田 宗広, 大岩 尚宏, 島本 裕志, 竹部 晶雄, 平松 雅巳(著)

1. 前回の復習

perfとは

NTT DATA

- perf(Performance Counters for Linux)とは？
  - Linuxカーネル上の統合パフォーマンスツール
  - イベント数計測にCPUに内蔵されているレジスタを使用
  - perf tool**s** (ツールの集合体、サブコマンド群)

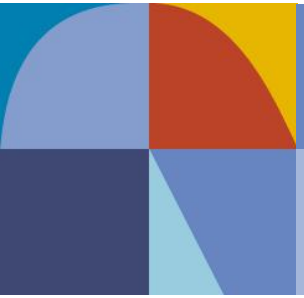
- Performance Countersとは？？
  - ハードウェアのイベントの回数を計測するCPU内蔵のレジスタ

- イベントとは？？
  - キャッシュミス、分岐予測ミス（HWイベント）
  - ページフォルト、コンテキストスイッチ(SWイベント)

プロファイラとしてありたい姿

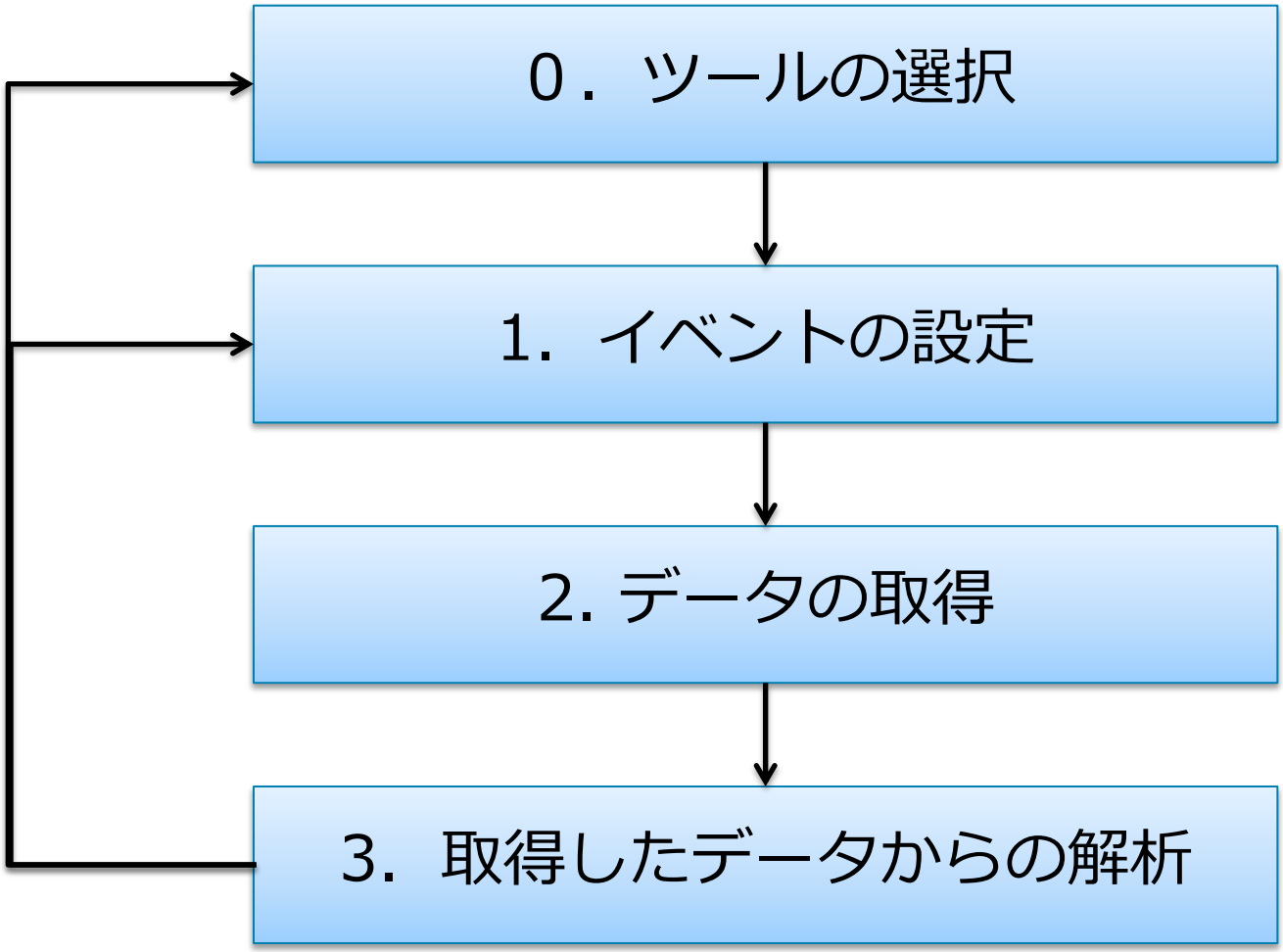
NTT DATA

- 低いオーバヘッド
  - 解析対象のシステムは既に何らかのオーバヘッドが発生していることが多い。そこにさらに負荷がかかるツールを使おうとしても、ツールが動かなかったり、システムがダウンする可能性もある。
- 多くの情報
  - トラブルの原因はいっぱい考えられるから。
- 見やすい表示(多くの情報の中から取捨選択が可能)
  - なにがどう書いてあるのか分からないと、トラブルの原因が表示されていても見落とす可能性もある。



2. perfを使ったプロファイリングの流れ

perfを使ったプロファイリングの流れ



perf の使い方

前編より

- perfの使い方

```
# perf <コマンド> [オプション]
```

- 代表的なコマンド

コマンド	説明
perf annotate	perf recordで作成したperfのデータを読み込み、 関数で呼び出した命令レベルのトレース結果を表示する。
perf probe	新たに動的なトレースポイント(イベント)を定義する。
perf record	イベントの記録を行う。
perf report	perf recordで記録したイベントをプロンプトに表示する。
perf script	perf recordで作成したperfのデータを読み込み、 トレース結果を表示する。
perf stat	引数に指定したコマンドのパフォーマンスカウンタの値を表示する。
perf top	Linuxコマンドの”top”のように動的にシステム監視を行う。

0．ツールの選択

NTT DATA

```
usage: perf [--version] [--help] COMMAND [ARGS]

The most commonly used perf commands are:
  annotate        Read perf.data (created by perf record) and display annotated code
  archive         Create archive with object files with build-ids found in perf.data file
  bench           General framework for benchmark suites
  buildid-cache   Manage build-id cache.
  buildid-list    List the buildids in a perf.data file
  diff            Read two perf.data files and display the differential profile
  evlist          List the event names in a perf.data file
  inject          Filter to augment the events stream with additional information
  kmem            Tool to trace/measure kernel memory(slab) properties
  kvm             Tool to trace/measure kvm guest os
  list            List all symbolic event types
  lock            Analyze lock events
  probe           Define new dynamic tracepoints
  record          Run a command and record its profile into perf.data
  report          Read perf.data (created by perf record) and display the profile
  sched           Tool to trace/measure scheduler properties (latencies)
  script          Read perf.data (created by perf record) and display trace output
  stat            Run a command and gather performance counter statistics
  test            Runs sanity tests.
  timechart       Tool to visualize total system behavior during a workload
  top             System profiling tool.
```

1. イベントの選択

前編より

分類	説明	具体例
Hardware event	プロッセサで計測されるイベント。	cpu-cycles, cache-missesなど
Software event	カーネルのカウンタで計測されるイベント。	cpu-clock, page-faultsなど
Hardware cache event	プロッセサで計測されるイベント。	L1-dcache-load-misses, branch-loadsなど
Tracepoint event	カーネルの処理を記録するためにカーネルに埋め込まれたトレースポイント。	sched:sched_stat_runtime, syscalls:sys_enter_socketなど

2. データの取得

perf recordなどで、データを記録する。

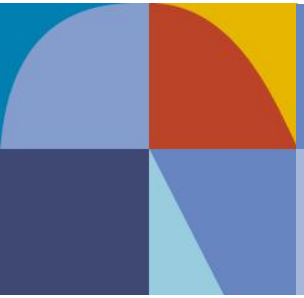
```
# perf record -e cpu-clock stress -c 4 -i 2 -m 2 --timeout 10s
stress: info: [5843] dispatching hogs: 4 cpu, 2 io, 2 vm, 0 hdd
stress: info: [5843] successful run completed in 10s
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.247 MB perf.data (~10775 samples) ]

# ls
perf.data
```

3. 取得したデータからの解析

NTT DATA

perf reportなどで、データを表示、解析する。

```
Samples: 37K of event 'instructions', Event count (approx.): 8662982577
  9.82%  postgres  postgres          [.] AllocSetAlloc
  3.51%  postgres  postgres          [.] base_yyparse
  2.97%  postgres  postgres          [.] SearchCatCache
  2.69%  postgres  postgres          [.] MemoryContextAllocZeroAligned
  2.01%  postgres  postgres          [.] expression_tree_walker
  1.92%  postgres  postgres          [.] MemoryContextAlloc
  1.83%  postgres  libc-2.15.so      [.] __strcmp_sse42
  1.77%  postgres  postgres          [.] hash_search_with_hash_value
  1.63%  postgres  postgres          [.] nocachegetattr
  1.33%  postgres  postgres          [.] MemoryContextAllocZero
  1.26%  postgres  postgres          [.] core_yylex
  1.23%  postgres  libc-2.15.so      [.] __memcpy_ssse3
  1.22%  postgres  postgres          [.] hash_any
  1.03%  postgres  postgres          [.] hash_uint32
  1.00%  postgres  postgres          [.] new_list
  0.95%  postgres  postgres          [.] lappend
             :
```



3. PostgreSQLをプロファイリングしてみよう

今回の講演に当たって使用した環境

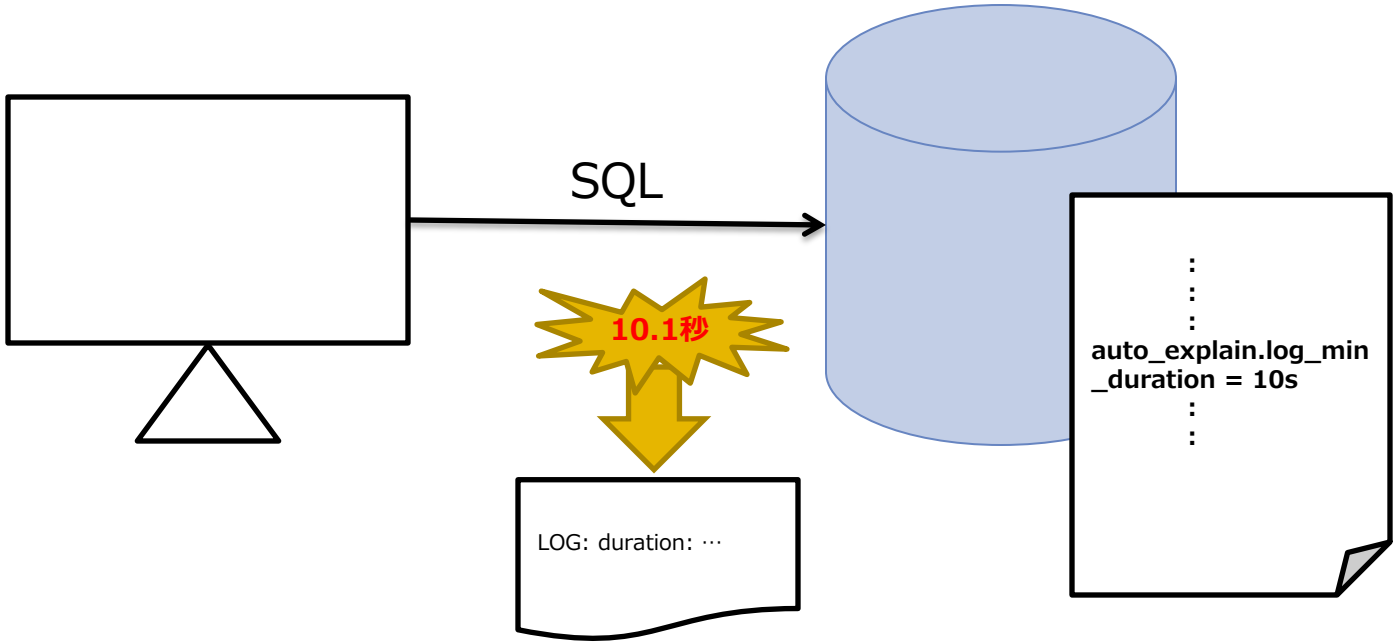
ソフトウェア名	バージョン	備考
Fedra	17	今回は物理マシンにInstall
Linux kernel	3.2.2	
PostgreSQL	9.2.2	

auto_explainを使用したときのオーバヘッドを解析

NTT DATA

auto_explainとは…

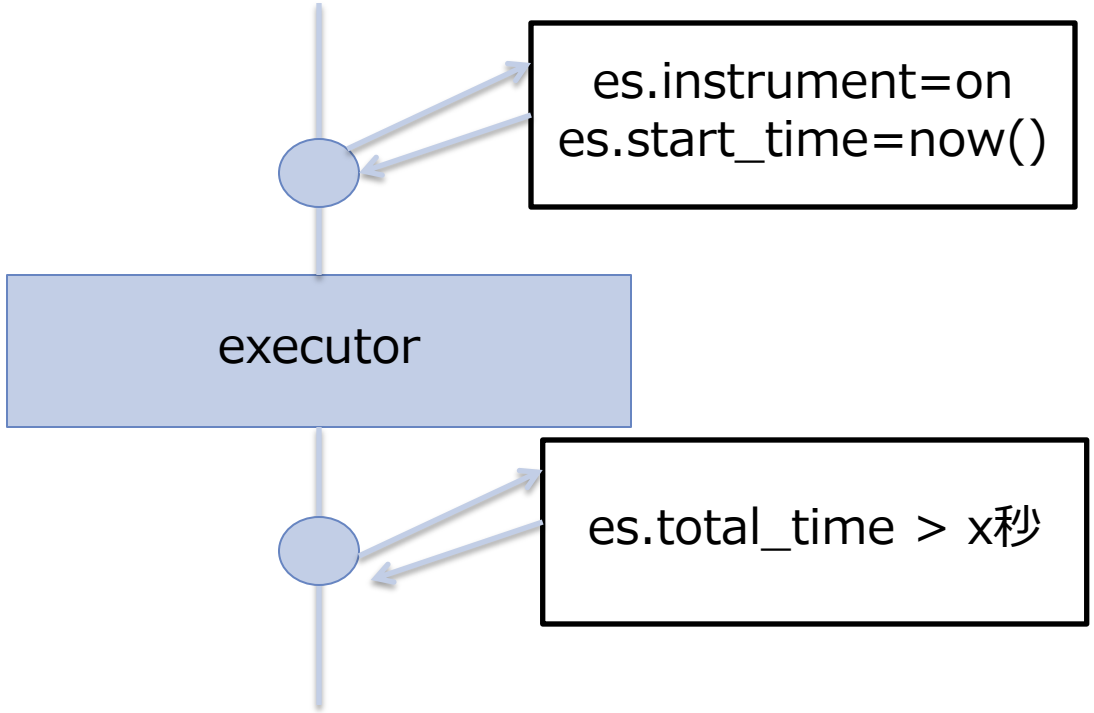
- 設定した時間よりも遅いクエリの実行計画をログに記録する機能。
- アプリケーションにおける最適化されていない問い合わせを追跡するのに特に有用。(by マニュアル)
- ONにしておけば、サポートする側が楽になるかも。



auto_explainを使用したときのオーバヘッドを解析

NTT DATA

- もう少しだけ…



auto_explain…マニュアルの気になる記述

NTT DATA

ロマニュアルより

：

：

：

auto_explain.log_analyze (boolean)

auto_explain.log_analyzeは、実行計画のログが取得されたときに出力されるものとして、単にEXPLAIN出力ではなく、EXPLAIN ANALYZE出力を行います。このパラメータはデフォルトで無効です。スーパユーザのみ、この設定を変更できます。

注意: このパラメータが有効の場合、計画ノードごとの時間的調整は事実上ログされるまで如何に時間が掛かろうと、全ての実行文に対して引き起こります。**極端に性能上のマイナス**の影響が起こり得ます。

：

：

：

（赤字と下線は講演者による）

auto_explainを使用したときのオーバヘッド

NTT DATA

- auto_explainをONにした場合のオーバヘッド
  1. EXPLAIN ANALYZEの情報収集のためのオーバヘッド
  2. サーバログ出力にかかるI/O負荷

auto_explainを使用したときのオーバヘッド

NTT DATA

- auto_explainをONにした場合のオーバヘッド
  1. EXPLAIN ANALYZEの情報収集のためのオーバヘッド
  2. サーバログ出力にかかるI/O負荷

perf は、I/O負荷を測る用途には適していないので、今回は、1のオーバヘッドに特化した解析を行う。

事前準備

NTT DATA

ログ出力のオフ(IO負荷の軽減)

ログ出力にかかる処理が現れないようにするため。

下記の処理を行う。

```
$ cat postgresql.conf
      :
# - Where to Log -
log_destination = 'stderr'              # Valid values are combinations of

# This is used when logging to stderr:
logging_collector = on          # Enable capturing of stderr and csvlog

# These are only used if logging_collector is on:
log_directory = 'pg_log'                # directory where log files are written,
                                        # can be absolute or relative to PGDATA
log_filename = 'postgresql-tmp.log'     # log file name pattern,
      :

$ ln /dev/null /xxxxx/data/pg_logpostgresql-tmp.log
```

※ ただし、この処理を行ってもログを生成する処理は行われる。

1.イベントの選択

NTT DATA

どれにしようか…

List of pre-defined events (to be used in -e):

cpu-cycles OR cycles	[Hardware event]
instructions	[Hardware event]
cache-references	[Hardware event]
cache-misses	[Hardware event]
branch-instructions OR branches	[Hardware event]
branch-misses	[Hardware event]
bus-cycles	[Hardware event]
stalled-cycles-frontend OR idle-cycles-frontend	[Hardware event]
stalled-cycles-backend OR idle-cycles-backend	[Hardware event]
ref-cycles	[Hardware event]
cpu-clock	[Software event]
task-clock	[Software event]
page-faults OR faults	[Software event]
context-switches OR cs	[Software event]
cpu-migrations OR migrations	[Software event]
minor-faults	[Software event]
major-faults	[Software event]
alignment-faults	[Software event]
emulation-faults	[Software event]
L1-dcache-loads	[Hardware cache event]
L1-dcache-load-misses	[Hardware cache event]

:

:

:

1.イベントの選択

- いくつかのイベントの取得結果をまとめてみてみたい

  そんなときには、

perf stat

前編より

perf statは、指定したコマンドのパフォーマンスカウンタの値を表示できる。
perf record/reportと違い、指定したコマンドの総合的な統計情報を得ることが可能である。

■ こんな時に便利
- ― あるコマンドについて、総合的な(複数のイベントにまたがった)性能測定情報が知りたい。
- ― 性能の悪い処理について、何が原因なのか知りたい。
- ― 性能改善(暫定対処、本格対処)を行うにあたっての数値目標を出したい。

perf stat

前編より

perf statは、指定したコマンドのパフォーマンスカウンタの値を表示できる。
perf record/reportと違い、指定したコマンドの総合的な統計情報を得ることが可能である。

■ こんな時に便利
- ー あるコマンドについて、**総合的な(複数のイベントにまたがった)**性能測定情報が知りたい。
- ー 性能の悪い処理について、何が原因なのか知りたい。
- ー 性能改善(暫定対処、本格対処)を行うにあたっての数値目標を出したい。

測定方針～今回の比較対象

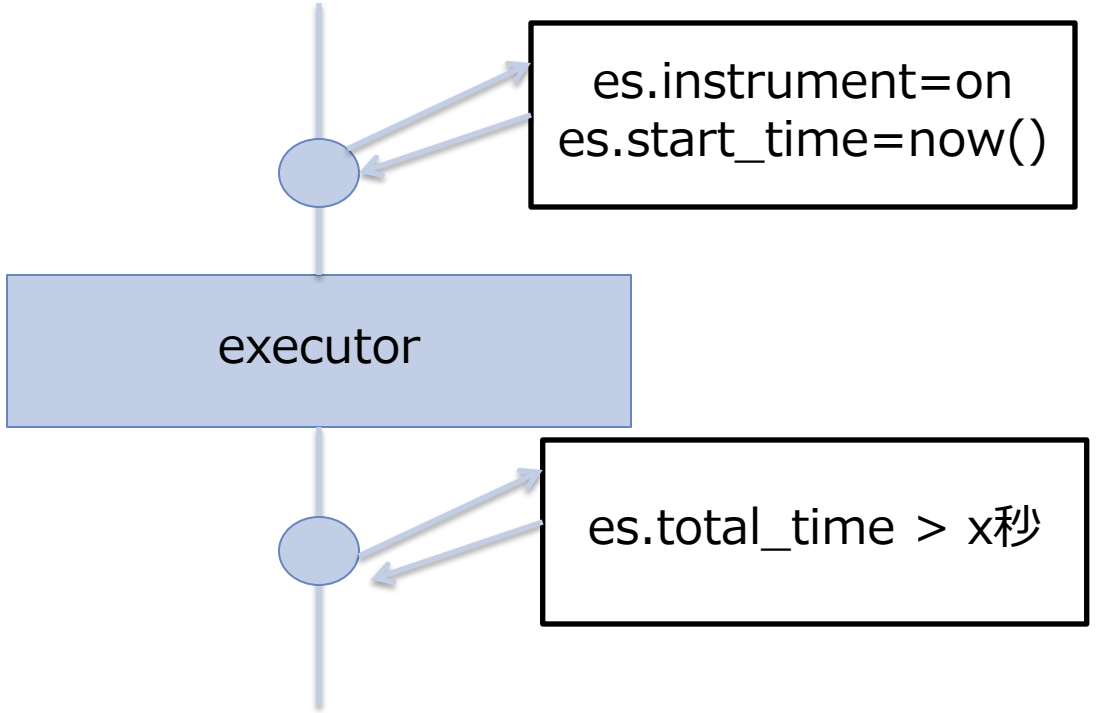
#	比較対象	内容	意図
1	ノーマル	実行計画を取得しない状態。	基準値。
2	EXPLAINモード	EXPLAINコマンドを発行した実行計画を自動的に取得する。	実行計画を取得、保持する処理のオーバヘッドを知りたい。
3	EXPLAIN ANALYZEモード	EXPLAIN ANALYZEコマンドを発行した実行計画を自動的に取得する。	実際のコマンド実行時間などEXPLAIN ANALYZEで取れる情報を取得、保持する処理のオーバヘッドが知りたい。

測定方針～仮説

NTT DATA

EXPLAIN ANALYZEを考える

- EXPLAIN ANALYZE による負荷は、実行計画を取得、保持する処理。

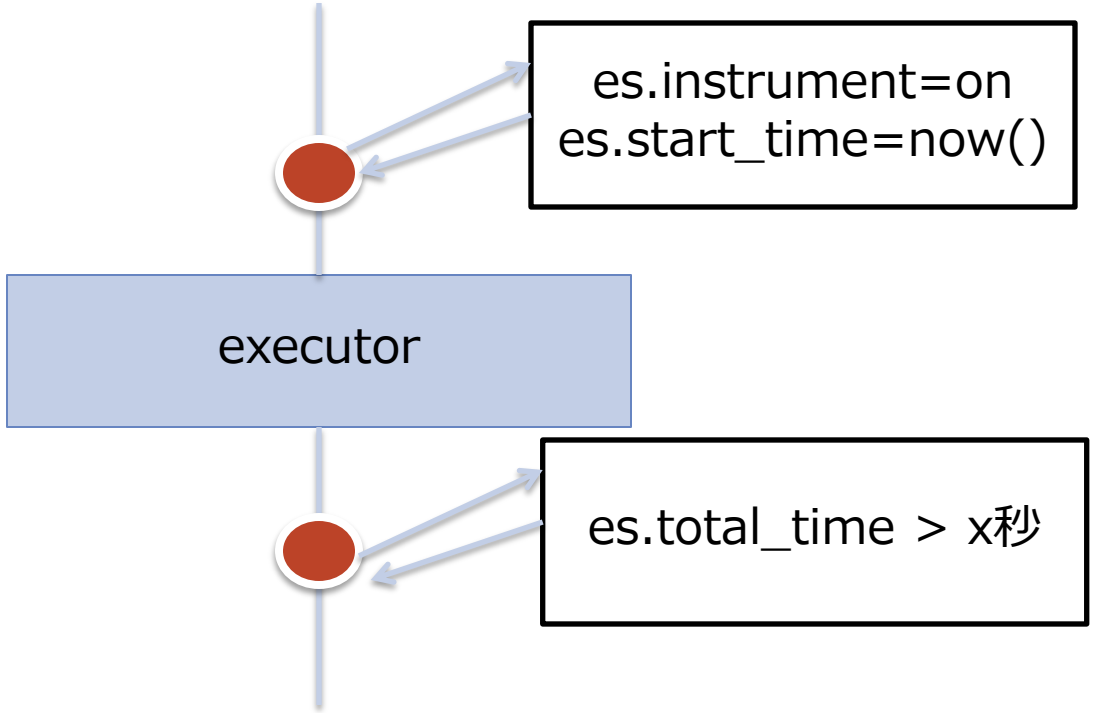


測定方針～仮説

NTT DATA

□EXPLAIN ANALYZEを考える

- EXPLAIN ANALYZE による負荷は、実行計画を取得、保持する処理。



測定方針

NTT DATA

□SQL, EXPLAIN ANALYZEをどう走らせる？

- SQLの重さに関わらず、 EXPLAIN ANALYZEが走るように設定する。
  - 実行計画取得の閾値を 0秒に設定(postgresql.confのパラメタ)。

```
$ cat postgresql.conf
      :
auto_explain.log_min_duration = ‘0’
      :
```

  - EXPLAIN ANALYZEの影響を相対的に大きくするため、SQLは負荷が小さいものを選択。
    pgbench –S オプション で発行。

測定の流れ(1/2)

NTT DATA

どう測定する？

- SQLや実行計画を取得する処理の情報のみを取得したい。

1. pgbenchでSQLを発行。

```
$ pgbench –i test
$ pgbench –S –T 600 test
Sarting vacuum... end.
```

2. psコマンドで、プロセスIDを調べる。

```
# ps aux | grep postgres
                  :
postgres  6868  0.0  0.0 121832   772 ?        Ss   Feb07   0:00 postgres: logger process
postgres  6870  0.0  0.0 153452  1408 ?        Ss   Feb07   0:00 postgres: checkpointer process
postgres  6871  0.0  0.0 153452   912 ?        Ss   Feb07   0:00 postgres: writer process
postgres  6872  0.0  0.0 153452   908 ?        Ss   Feb07   0:00 postgres: wal writer process
postgres  6873  0.0  0.0 121964   968 ?        Ss   Feb07   0:00 postgres: stats collector process
postgres  7480 15.4  0.0  13796  1052 pts/1    S+   00:29   1:25 pgbench -S -T 600 test
postgres  7482 89.4  0.2 154676 19620 ?        Rs   00:29   8:14 postgres: postgres test [local] SELECT
daichi    7512  0.0  0.0 109420   880 pts/10   S+   00:38   0:00 grep --color=auto postgres
```

測定の流れ(2/2)

NTT DATA

3. 調べたプロセスIDをもとに、perf statを実行

```
# perf stat -p 7482 sleep 10

 Performance counter stats for process id '5641':

      8356.290496 task-clock                #    0.836 CPUs utilized
          82,898 context-switches          #    0.010 M/sec
             332 CPU-migrations            #    0.040 K/sec
               0 page-faults               #    0.000 K/sec
  13,800,470,721 cycles                    #    1.652 GHz         [100.00%]
   9,910,834,211 stalled-cycles-frontend   #   71.82% frontend cycles idle    [100.00%]
   8,099,655,897 stalled-cycles-backend    #   58.69% backend cycles idle    [100.00%]
   8,873,259,146 instructions              #    0.64  insns per cycle
                                           #    1.12  stalled cycles per insn [100.00%]
   1,752,642,657 branches                  #  209.739 M/sec         [100.00%]
      29,767,895 branch-misses             #    1.70% of all branches

    10.000876597 seconds time elapsed
```

データ取得結果(perf stat)

NTT DATA

ロperf stat出力結果(3回の計測の平均値)

比較対象	ノーマル	EXPLAINモード	EXPLAIN ANALYZEモード
task-clock	8,525	8,474	8,530
context-switches	85,115	83,528	82,456
CPU-migrations	283	419	898
page-faults	0	0	0
cycles	14,073,651,082	13,996,156,300	14,114,324,153
stalled-cycles-frontend	10,082,417,996	10,099,475,984	10,020,292,828
stalled-cycles-backend	8,224,267,261	8,276,793,827	8,167,568,987
instructions	9,110,231,843	8,908,543,005	9,237,358,941
branches	1,799,358,657	1,759,630,145	1,790,358,429
branch-misses	30,480,893	29,803,620	33,819,770
tps(pgbenchのアウトプット) ※	7420	6328	5553

※ コネクション確立の時間を含んでいない。小数点以下四捨五入

データ取得結果(perf stat)

NTT DATA

□perf stat出力結果(3回の計測の平均値)

比較対象	ノーマル	EXPLAINモード	EXPLAIN ANALYZEモード
task-clock	8,525	8,474	8,530
context-switches	85,115	83,528	82,456
CPU-migrations	283	419	898
page-faults	0	0	0
cycles	14,073,651,082	13,996,156,300	14,114,324,153
stalled-cycles-frontend	10,082,417,996	10,099,475,984	10,020,292,828
stalled-cycles-backend	8,224,267,261	8,276,793,827	8,167,568,987
instructions	9,110,231,843	8,908,543,005	9,237,358,941
branches	1,799,358,657	1,759,630,145	1,790,358,429
branch-misses	30,480,893	29,803,620	33,819,770
tps(pgbenchのアウトプット) ※	7420	6328	5553

※ コネクション確立の時間を含んでいない。小数点以下四捨五入

データ取得結果(perf stat)

NTT DATA

◘perf stat出力結果(3回の計測の平均値)

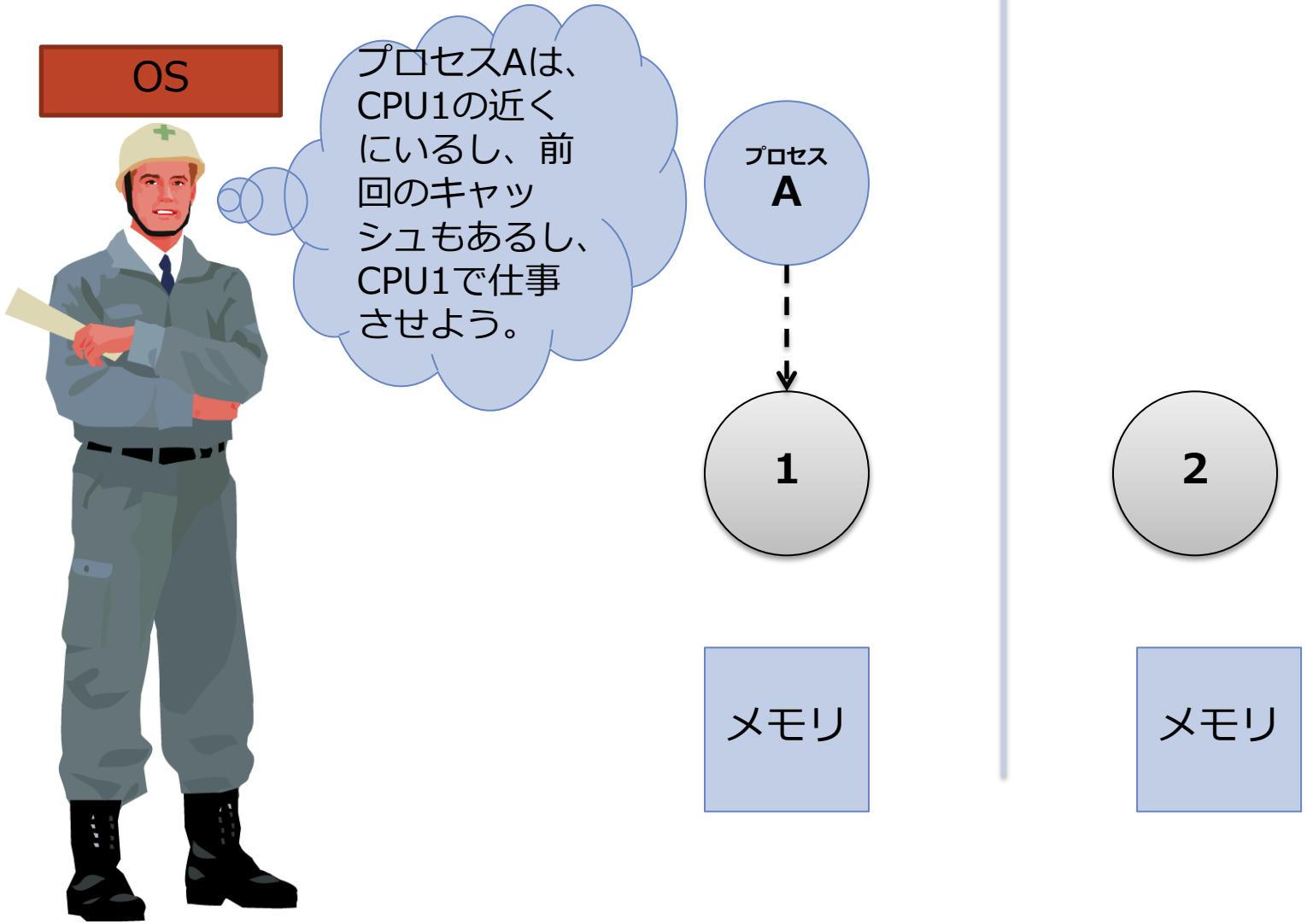
比較対象	ノーマル	EXPLAINモード	EXPLAIN ANALYZEモード
task-clock	8,525	8,474	8,530
context-switches	85,115	83,528	82,456
CPU-migrations	283	419	898
page-faults	0	0	0
cycles	14,073,651,082	13,9	
stalled-cycles-frontend	10,082,417,996	10,099,475,984	10,020,292,828
stalled-cycles-backend	8,224,267,261	8,276,793,827	8,167,568,987
instructions	9,110,231,843	8,908,543,005	9,237,358,941
branches	1,799,358,657	1,759,630,145	1,790,358,429
branch-misses	30,480,893	29,803,620	33,819,770
tps(pgbenchのアウトプット) ※	7420	6328	5553

CPU-migrationに違いが表れている

※ コネクション確立の時間を含んでいない。小数点以下四捨五入

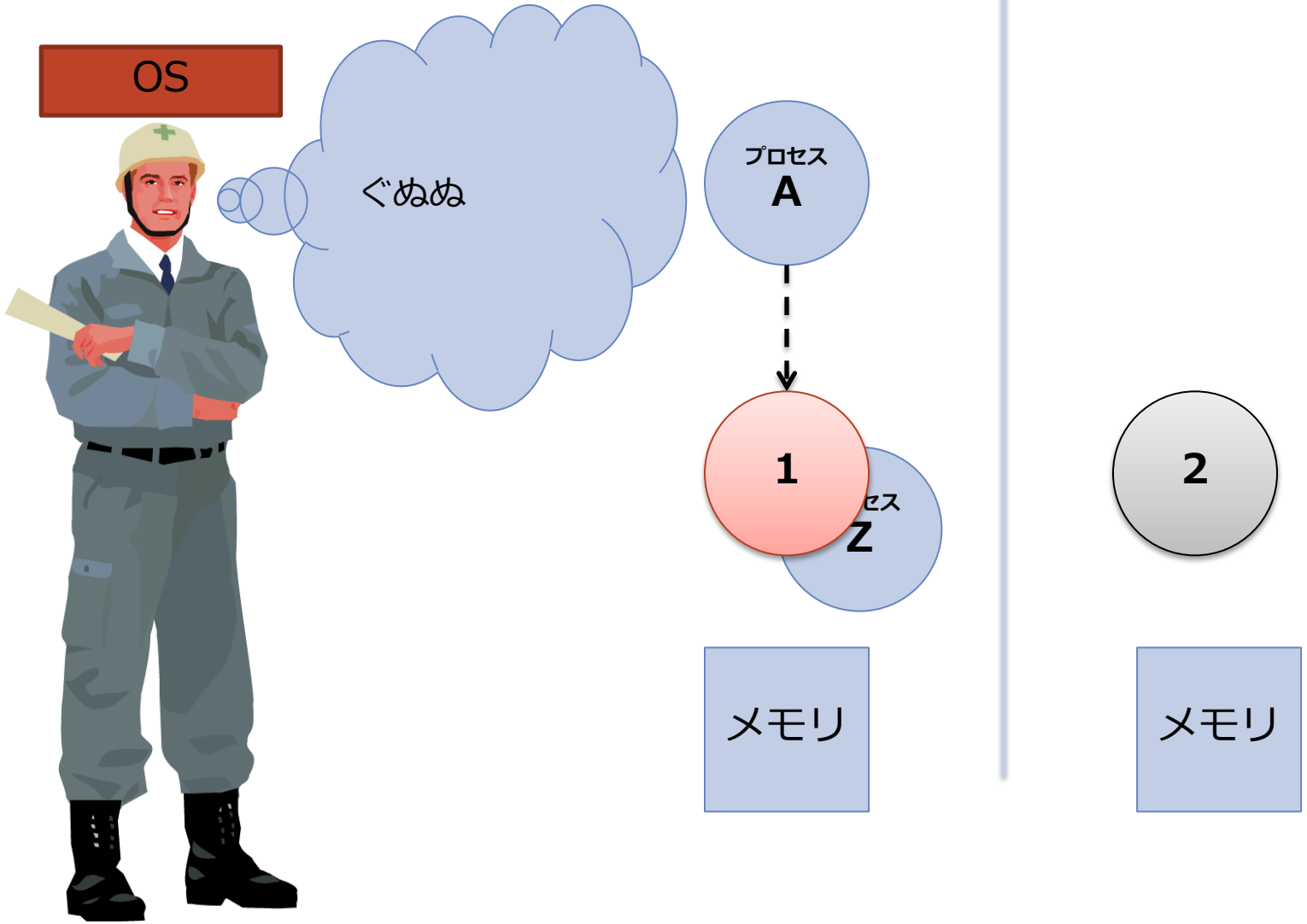
CPU-migrationとは…

プロセスは、OSによってスケジューリングされたCPUで処理を行いたい。



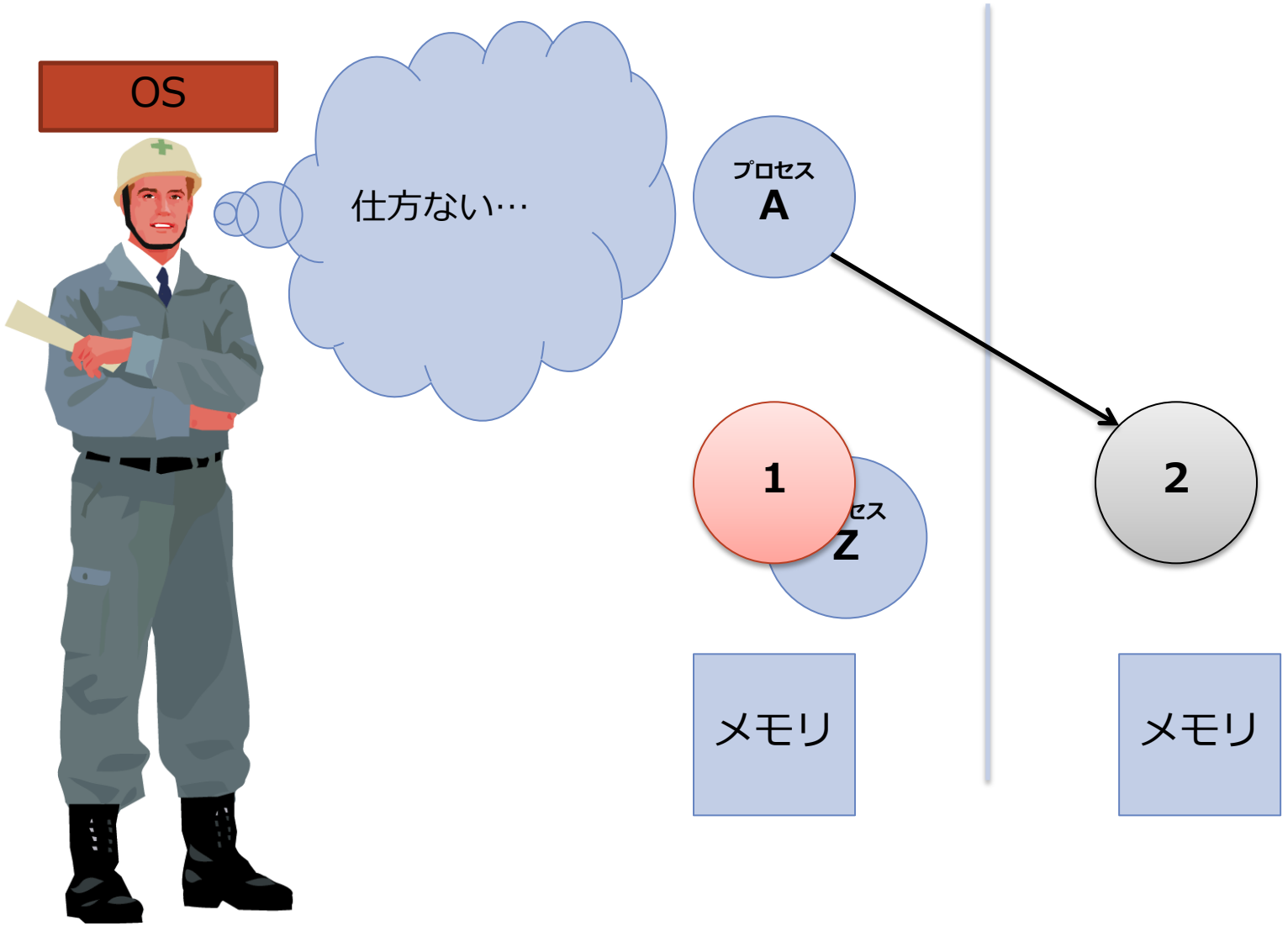
CPU-migrationとは…

CPUが他のプロセスによって、占有されている場合がある。



CPU-migrationとは…

その場合は、他のCPUに移動して処理を行う。



perf recordで動きを追う

NTT DATA

◘perf recordで特定のイベントを起こす処理の動きを追う

perf実行前の準備

```
$ pgbench –i test
$ pgbench –S –T 600 test
Sarting vacuum... end.
```

```
$ ps aux | grep postgres
                  :
postgres  8868  0.0  0.0 121832   772 ?       Ss   Feb07   0:00 postgres: logger process
postgres  8870  0.0  0.0 153452  1408 ?       Ss   Feb07   0:00 postgres: checkpointer process
postgres  8861  0.0  0.0 153452   912 ?       Ss   Feb07   0:00 postgres: writer process
postgres  8862  0.0  0.0 153452   908 ?       Ss   Feb07   0:00 postgres: wal writer process
postgres  8963  0.0  0.0 121964   968 ?       Ss   Feb07   0:00 postgres: stats collector process
postgres  8960 15.4  0.0  13796  1052 pts/1    S+   00:29   1:25 pgbench -S -T 600 test
postgres  8968 89.4  0.2 154676 19620 ?       Rs   00:29   8:14 postgres: postgres test [local] SELECT
daichi    8969  0.0  0.0 109420   880 pts/10   S+   00:38   0:00 grep --color=auto postgres
```

perf recordを実行

```
# perf record -e cpu-migrations -o explain_ana_mig.data -p 8968 sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.058 MB explain_ana_mig.data (~2553 samples) ]
```

perf reportで結果出力

NTT DATA

□perf reportで記録したデータを出力する

```
# perf report -i explain_ana_mig.data
```

ノーマル

```
100.00%  postgres  [kernel.kallsyms]  [k] try_to_wake_up
```

EXPLAIN

```
100.00%  postgres  [kernel.kallsyms]  [k] try_to_wake_up
```

EXPLAIN ANALYZE

```
100.00%  postgres  [kernel.kallsyms]  [k] try_to_wake_up
```

perf reportで結果出力

NTT DATA

□perf reportで記録したデータを出力する

```
# perf report -i explain_ana_mig.data
```

ノーマル

```
100.00%  postgres  [kernel.kallsyms]  [k] try_to_wake_up
```

EXPLAIN

```
100.00%  postgres  [kernel.kallsyms]  [k] try_to_wake_up
```

EXPLAIN ANALYZE

```
100.00%  postgres  [kernel.kallsyms]  [k] try_to_wake_up
```

CPU-migrationを。原因となる命令内容に違いがあるか？

perf recordで動きを追う

NTT DATA

◘perf recordで特定のイベントを起こす処理の動きを追う

instructionsをイベント指定して、perf recordを実行

```
# perf record -e instructions -o explain_ana_instructions.data -p 8968 sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.058 MB explain_ana_mig.data (~2553 samples) ]

# perf report -i explain_ana_instructions.data
```

EXPLAIN ANALYZE

```
Samples: 37K of event 'instructions', Event count (approx.): 8662982577
  8.12%  postgres  postgres          [.] AllocSetAlloc
  3.01%  postgres  postgres          [.] append_with_tabs
  2.79%  postgres  libc-2.15.so      [.] vfprintf
  2.78%  postgres  libc-2.15.so      [.] __printf_fp
  2.74%  postgres  postgres          [.] SearchCatCache
  2.53%  postgres  postgres          [.] base_yyparse
  2.07%  postgres  libc-2.15.so      [.] __strcmp_sse42
  1.95%  postgres  postgres          [.] MemoryContextAllocZeroAligned
  1.44%  postgres  postgres          [.] expression_tree_walker
  1.43%  postgres  postgres          [.] MemoryContextAlloc
  1.32%  postgres  postgres          [.] hash_search_with_hash_value
  1.31%  postgres  libc-2.15.so      [.] __memcpy_ssse3
  1.28%  postgres  postgres          [.] ScanKeywordLookup
  1.28%  postgres  postgres          [.] nocachegetattr
  1.24%  postgres  postgres          [.] hash_any
  1.22%  postgres  libc-2.15.so      [.] _IO_default_xsputn
  1.11%  postgres  postgres          [.] core_yylex
  1.00%  postgres  postgres          [.] MemoryContextAllocZero
                 :
                 :
```

出力結果の比較

NTT DATA

ノーマル

```
Samples: 37K of event 'instructions', Event count (approx.): 8318102033
 10.11% postgres  postgres          [.] AllocSetAlloc
  3.27% postgres  postgres          [.] base_yyparse
  3.09% postgres  postgres          [.] SearchCatCache
  2.72% postgres  postgres          [.] MemoryContextAllocZeroAligned
  1.95% postgres  libc-2.15.so      [.] __strcmp_sse42
  1.91% postgres  postgres          [.] MemoryContextAlloc
  1.80% postgres  postgres          [.] expression_tree_walker
  1.68% postgres  postgres          [.] hash_search_with_hash_value
  1.53% postgres  postgres          [.] nocachegetattr
  1.40% postgres  postgres          [.] hash_any
  1.31% postgres  postgres          [.] core_yylex
  1.31% postgres  libc-2.15.so      [.] __memcpy_ssse3
                  :
```

EXPLAIN

```
Samples: 38K of event 'instructions', Event count (approx.): 8745281382
 10.03% postgres  postgres          [.] AllocSetAlloc
  3.23% postgres  postgres          [.] SearchCatCache
  3.22% postgres  postgres          [.] base_yyparse
  2.62% postgres  postgres          [.] MemoryContextAllocZeroAligned
  1.88% postgres  postgres          [.] expression_tree_walker
  1.86% postgres  postgres          [.] MemoryContextAlloc
  1.80% postgres  postgres          [.] hash_search_with_hash_value
  1.78% postgres  libc-2.15.so      [.] __strcmp_sse42
  1.44% postgres  postgres          [.] nocachegetattr
  1.32% postgres  postgres          [.] hash_any
  1.32% postgres libc-2.15.so       [.] __memcpy_ssse3
  1.28% postgres postgres           [.] core_yylex
                  :
```

EXPLAIN ANALYZE

```
Samples: 37K of event 'instructions', Event count (approx.): 8662982577
 8.12% postgres  postgres           [.] AllocSetAlloc
 3.01% postgres  postgres           [.] append_with_tabs
 2.79% postgres  libc-2.15.so       [.] vfprintf
 2.78% postgres  libc-2.15.so       [.] __printf_fp
 2.74% postgres  postgres           [.] SearchCatCache
 2.53% postgres  postgres           [.] base_yyparse
 2.07% postgres  libc-2.15.so       [.] __strcmp_sse42
 1.95% postgres  postgres           [.] MemoryContextAllocZeroAligned
 1.44% postgres  postgres           [.] expression_tree_walker
 1.43% postgres  postgres           [.] MemoryContextAlloc
 1.32% postgres  postgres           [.] hash_search_with_hash_value
 1.31% postgres  libc-2.15.so       [.] __memcpy_ssse3
 1.28% postgres  postgres           [.] ScanKeywordLookup
 1.28% postgres  postgres           [.] nocachegetattr
                 :
```

出力結果の比較

NTT DATA

```
Samples: 37K of event 'instructions', Event count (approx.): 8318102033
 10.11% postgres  postgres          [.] AllocSetAlloc
  3.27% postgres  postgres          [.] base_yyparse
  3.09% postgres  postgres          [.] SearchCatCache
  2.72% postgres  postgres          [.] MemoryContextAllocZeroAligned
  1.95% postgres  libc-2.15.so      [.] __strcmp_sse42
  1.91% postgres  postgres          [.] MemoryContextAlloc
  1.80% postgres  postgres          [.] expression_tree_walker
  1.68% postgres  postgres          [.] hash_search_with_hash_value
  1.53% postgres  postgres          [.] nocachegetattr
  1.40% postgres  postgres          [.] hash_any
  1.31% postgres  postgres          [.] core_yylex
  1.31% postgres  libc-2.15.so      [.] __memcpy_ssse3
        :
```

ノーマル

```
Samples: 37K of event 'instructions', Event count (approx.): 8662982577
 8.12% postgres  postgres          [.] AllocSetAlloc
 3.01% postgres  postgres          [.] append_with_tabs
 2.79% postgres  libc-2.15.so      [.] vfprintf
 2.78% postgres  libc-2.15.so      [.] __printf_fp
 2.74% postgres  postgres          [.] SearchCatCache
 2.53% postgres  postgres          [.] base_yyparse
 2.07% postgres  libc-2.15.so      [.] __strcmp_sse42
 1.95% postgres  postgres          [.] MemoryContextAllocZeroAligned
 1.44% postgres  postgres          [.] expression_tree_walker
 1.43% postgres  postgres          [.] MemoryContextAlloc
 1.32% postgres  postgres          [.] hash_search_with_hash_value
 1.31% postgres  libc-2.15.so      [.] __memcpy_ssse3
 1.28% postgres  postgres          [.] ScanKeywordLookup
 1.28% postgres  postgres          [.] nocachegetattr
       :
```

EXPLAIN ANALYZE

出力結果の比較

NTT DATA

ノーマル

```
Samples: 37K of event 'instructions', Event count (approx.): 8318102033
 10.11% postgres  postgres          [.] AllocSetAlloc
  3.27% postgres  postgres          [.] base_yyparse
  3.09% postgres  postgres          [.] SearchCatCache
  2.72% postgres  postgres          [.] MemoryContextAllocZeroAligned
  1.95% postgres  libc-2.15.so      [.] __strcmp_sse42
  1.91% postgres  postgres          [.] MemoryContextAlloc
  1.80% postgres  postgres          [.] expression_tree_walker
  1.68% postgres  postgres          [.] hash_search_with_hash_value
  1.53% postgres  postgres          [.] nocachegetattr
  1.40% postgres  postgres          [.] hash_any
  1.31% postgres  postgres          [.] core_yylex
  1.31% postgres  libc-2.15.so      [.] __memcpy_ssse3
        :
```

EXPLAIN ANALYZE

```
Samples: 37K of event 'instructions', Event count (approx.): 8662982577
  8.12% postgres  postgres          [.] AllocSetAlloc
  3.01% postgres  postgres          [.] append_with_tabs
  2.79% postgres  libc-2.15.so      [.] vfprintf
  2.78% postgres  libc-2.15.so      [.] __printf_fp
  2.74% postgres  postgres          [.] SearchCatCache
  2.53% postgres  postgres          [.] base_yyparse
  2.07% postgres  libc-2.15.so      [.] __strcmp_sse42
  1.95% postgres  postgres          [.] MemoryContextAllocZeroAligned
  1.44% postgres  postgres          [.] expression_tree_walker
  1.43% postgres  postgres          [.] MemoryContextAlloc
  1.32% postgres  postgres          [.] hash_search_with_hash_value
  1.31% postgres  libc-2.15.so      [.] __memcpy_ssse3
  1.28% postgres  postgres          [.] ScanKeywordLookup
  1.28% postgres  postgres          [.] nocachegetattr
        :
```

append_with_tabsとは

NTT DATA

PATH: src/backend/utils/error/elog.c

```
3011 /*
3012  *      append_with_tabs
3013  *
3014  *      Append the string to the StringInfo buffer, inserting a tab after any
3015  *      newline.
3016  */
3017 static void
3018 append_with_tabs(StringInfo buf, const char *str)
3019 {
3020   char      ch;
3021
3022   while ((ch = *str++) != '¥0')
3023   {
3024         appendStringInfoCharMacro(buf, ch);
3025         if (ch == '¥n')
3026               appendStringInfoCharMacro(buf, '¥t');
3027   }
3028 }
```

ログ生成時、改行が入る際にタブを挿入する関数。ログの整形などに使用される。

append_with_tabsはなぜ呼ばれているか

NTT DATA

- EXPLIN ANALYZE発行時、ログ生成で使用されている(※)
  - EXPLAIN発行時は、あまり呼ばれていない
  - EXPLAINよりEXPLAIN ANALYZEの方が、改行の量が多いので、負荷が多くかかったのではないかと推測できる。

(※)本解析の事前準備でログ出力をオフにしているが、ログの生成処理は行われている。

append_with_tabsはなぜ呼ばれているか

NTT DATA

- EXPLIN ANALYZE発行時、ログ生成で使用されている(※)
  - EXPLAIN発行時は、あまり呼ばれていない
  - EXPLAINよりEXPLAIN ANALYZEの方が、改行の量が多いので、負荷が多くかかったのではないかと推測できる。

「極端に性能上のマイナス」を与えるのは、ログ生成だけなのか？
→EXPLAIN ANALYZEが、ログ生成を行わない設定にして再測定。

(※)本解析の事前準備でログ出力をオフにしているが、ログの生成処理は行われている。

再調査

□実行計画取得閾値を上げる。

- 実行計画取得の閾値を 10分に設定(postgresql.confのパラメタ)。

```
$ cat postgresql.conf
      :
auto_explain.log_min_duration = ’10min’
      :
```

再計測結果～perf record& reportで確認。

NTT DATA

perf recordで再計測したところ、append_with_tabsは呼ばれなくなった。

```
Samples: 37K of event 'instructions', Event count (approx.): 8662982577
  9.82%  postgres  postgres          [.] AllocSetAlloc
  3.51%  postgres  postgres          [.] base_yyparse
  2.97%  postgres  postgres          [.] SearchCatCache
  2.69%  postgres  postgres          [.] MemoryContextAllocZeroAligned
  2.01%  postgres  postgres          [.] expression_tree_walker
  1.92%  postgres  postgres          [.] MemoryContextAlloc
  1.83%  postgres  libc-2.15.so      [.] __strcmp_sse42
  1.77%  postgres  postgres          [.] hash_search_with_hash_value
  1.63%  postgres  postgres          [.] nocachegetattr
  1.33%  postgres  postgres          [.] MemoryContextAllocZero
  1.26%  postgres  postgres          [.] core_yylex
  1.23%  postgres  libc-2.15.so      [.] __memcpy_ssse3
  1.22%  postgres  postgres          [.] hash_any
  1.03%  postgres  postgres          [.] hash_uint32
  1.00%  postgres  postgres          [.] new_list
  0.95%  postgres  postgres          [.] lappend
```

EXPLAIN ANALYZE

perf statでも確認

```
# perf stat -p 10974 sleep 10

 Performance counter stats for process id '1974':

      7944.809348 task-clock                #    0.794 CPUs utilized          [100.00%]
         84,368 context-switches          #    0.016 M/sec                  [100.00%]
            546 CPU-migrations            #    0.000 M/sec                  [100.00%]
              0 page-faults               #    0.000 K/sec
     12,954,262,298 cycles                    #    1.631 GHz                    [100.00%]
      9,415,587,714 stalled-cycles-frontend   #   72.68% frontend cycles idle   [100.00%]
      7,719,126,537 stalled-cycles-backend    #   59.59% backend  cycles idle   [100.00%]
      7,955,133,955 instructions              #    0.61  insns per cycle
                                              #    1.18  stalled cycles per insn [100.00%]
      1,560,120,420 branches                  #  196.370 M/sec                  [100.00%]
         29,504,875 branch-misses             #    1.89% of all branches

     10.000889779 seconds time elapsed
```

←CPU-migrationの数が、EXPLAINモードと同程度の水準に。

考察

NTT DATA

auto_explain

- 軽いSQLが大量にはかれる場合
  - EXPLAIN ANALYZEが性能上影響を与えるのはログ生成
  - ログ出力の閾値を適切に設定すれば、スロークエリによるログ生成が頻発することはまれである。

EXPLAIN ANALYZEによる性能への影響は軽微

考察

NTT DATA

auto_explain

- 軽いSQLが大量にはかれる場合
  - EXPLAIN ANALYZEが性能上影響を与えるのはログ生成
  - ログ出力の閾値を適切に設定すれば、スロークエリによるログ生成が頻発することはまれである。

EXPLAIN ANALYZEによる性能への影響は軽微

では、重いSQLの場合は？

重いSQLで測定すると

NTT DATA

- 重いSQL
  - 多くのJOINを行うSELECTコマンドを使用

<SQLの一例>

```
SELECT n.nspname as "Schema",
       c.relname as "Name",
       CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'i' THEN 'index' WHEN 'S' THEN
'sequence' WHEN 's' THEN 'special' WHEN 'f' THEN 'foreign table' END as "Type",
      pg_catalog.pg_get_userbyid(c.relowner) as "Owner",
      c2.relname as "Table"
FROM  pg_catalog.pg_class c
      LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
      LEFT JOIN pg_catalog.pg_index i ON i.indexrelid = c.oid
      LEFT JOIN pg_catalog.pg_class c2 ON i.indrelid = c2.oid
WHERE c.relkind IN ('i','')
      AND n.nspname <> 'pg_catalog'
      AND n.nspname <> 'information_schema'
      AND n.nspname !~ '^pg_toast'
      AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;

END;
```

重いSQLで測定すると

NTT DATA

- 実行方法
  - pgbenchにカスタムスクリプトとして使用

```
$ pgbench –f test.sql –T 600 test
Sarting vacuum... end.
```

再計測結果～perf record& reportで確認。

NTT DATA

ノーマル

tps:505

```
Samples: 36K of event 'instructions', Event count (approx.): 18184791491
  12.37%  postgres  postgres          [.] slot_deform_tuple
   8.08%  postgres  postgres          [.] AllocSetAlloc
   4.96%  postgres  postgres          [.] SearchCatCache
   2.83%  postgres  postgres          [.] ExecEvalScalarArrayOp
   1.88%  postgres  postgres          [.] MemoryContextAllocZeroAligned
   1.86%  postgres  libc-2.15.so      [.] __strcmp_sse42
   1.86%  postgres  postgres          [.] slot_getattr
   1.82%  postgres  postgres          [.] base_yyparse
   1.72%  postgres  postgres          [.] hash_uint32
   1.63%  postgres  postgres          [.] expression_tree_walker
   1.45%  postgres  postgres          [.] MemoryContextAlloc
   1.29%  postgres  postgres          [.] lappend
   1.14%  postgres  postgres          [.] heapgettup_pagemode
   1.12%  postgres  postgres          [.] nocachegetattr
   1.02%  postgres  postgres          [.] ExecScan
   0.96%  postgres  postgres          [.] FunctionCall2Coll
   0.94%  postgres  postgres          [.] core_yylex
   0.92%  postgres  postgres          [.] expression_tree_mutator
   0.90%  postgres  postgres          [.] heapgetpage
   0.85%  postgres  postgres          [.] miss
   0.84%  postgres  libc-2.15.so      [.] __memcpy_ssse3
   0.81%  postgres  postgres          [.] check_stack_depth
     :
```

再計測結果～perf record& reportで確認。

NTT DATA

EXPLAIN ANALYZE
閾値'0'

tps:444

```
Samples: 36K of event 'instructions', Event count (approx.):17644280750
  10.11%postgres  postgres          [.] slot_deform_tuple
   7.80%  postgres  postgres         [.] AllocSetAlloc
   4.28%  postgres  postgres         [.] SearchCatCache
   2.36%  postgres  postgres         [.] ExecEvalScalarArrayOp
   2.18%  postgres  postgres         [.] MemoryContextAllocZeroAligned
   1.97%  postgres  postgres         [.] append_with_tabs
   1.96%  postgres  libc-2.15.so     [.] __strcmp_sse42
   1.59%  postgres  libc-2.15.so     [.] _IO_default_xsputn
   1.59%  postgres  postgres         [.] expression_tree_walker
   1.56%  postgres  postgres         [.] base_yyparse
   1.51%  postgres  libc-2.15.so     [.] __printf_fp
   1.48%  postgres  postgres         [.] hash_uint32
   1.47%  postgres  libc-2.15.so     [.] _IO_strn_overflow
   1.46%  postgres  postgres         [.] slot_getattr
   1.28%  postgres  postgres         [.] MemoryContextAlloc
   1.21%  postgres  postgres         [.] heapgettup_pagemode
   1.13%  postgres  [vdso]           [.] 0x00007fffec3bd60c
   1.11%  postgres  postgres         [.] lappend
   1.09%  postgres  libc-2.15.so     [.] vfprintf
   1.06%  postgres  postgres         [.] nocachegetattr
   0.93%  postgres  postgres         [.] heapgetpage
   0.93%  postgres  libc-2.15.so     [.] __memcpy_ssse3
      :
```

再計測結果～perf record& reportで確認。

NTT DATA

EXPLAIN ANALYZE
閾値'10min'

tps:467

```
Samples: 36K of event 'instructions', Event count (approx.): 18100297843
  11.89%postgres  postgres          [.] slot_deform_tuple
   9.37%  postgres  postgres         [.] AllocSetAlloc
   4.28%  postgres  postgres         [.] SearchCatCache
   2.36%  postgres  postgres         [.] ExecEvalScalarArrayOp
   2.18%  postgres  postgres         [.] MemoryContextAllocZeroAligned
   1.96%  postgres  postgres         [.] base_yyparse
   1.83%  postgres  libc-2.15.so     [.] __strcmp_sse42
   1.75%  postgres  postgres         [.] MemoryContextAlloc
   1.65%  postgres  postgres         [.] hash_uint32
   1.64%  postgres  postgres         [.] slot_getattr
   1.63%  postgres  postgres         [.] expression_tree_walker
   1.47%  postgres  postgres         [.] heapgettup_pagemode
   1.44%  postgres  postgres         [.] lappend
   1.13%  postgres  postgres         [.] core_yylex
   1.12%  postgres  [vdso]           [.] 0x00007fffe45ff8eb
   1.01%  postgres  postgres         [.] nocachegetattr
   0.99%  postgres  postgres         [.] new_tail_cell.isra.2
   0.91%  postgres  postgres         [.] heapgetpage
   0.87%  postgres  postgres         [.] ExecScan
   0.86%  postgres  postgres         [.] ExecInitExpr
   0.83%  postgres  postgres         [.] check_stack_depth
   0.83%  postgres  postgres         [.] FunctionCall2Coll
      :
```

考察

NTT DATA

◘auto_explain

- 重いSQLが大量にはかれる場合
  - EXPLAIN ANALYZEが性能上影響を与えるのはログ生成
  - ログ出力の閾値を適切に設定すれば、スロークエリによるログ生成が頻発することはまれである。

auto_explainをonにするのは、アリ。

perfのオーバヘッドってなかったの？

NTT DATA

- pgbenchとperfで試す。
  - -測定条件：pgbench -S –T 10でトライ。

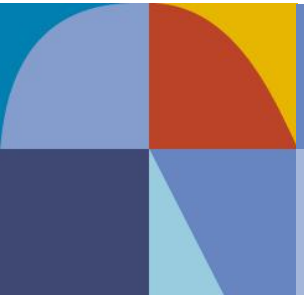
比較対象	ノーマル	EXPLAIN ANALYZEモード
pgbench	7047.1	4987.66
pgbench + perf	7021.12	4883.11

（余談）9.2におけるpostgresql.confの変化

NTT DATA

- silent_modeがない
  - -”silent_mode = on” と同じことをやりたい場合は、
    ログの出力先ファイルに/dev/nullへのシンボリックリンクを張る。

- custom_variable_classesがない
  - -shared_preload_libraries と auto_explain.log_min_duration さえ設定すれば、auto_explainが使用できる。



4. perfでハマった点

debuginfo

- PostgreSQLのオブジェクトのシンボル名を表示させたい場合
  - ソースコードからPostgreSQLをインストール
    ./configure時にデバッグを有効に。
  - rpmからだと、間違ったシンボルが表示される…

5. perfの使いどころ

perfの使いどころ

NTT DATA

- システム運用中、開発中を問わない。
  - -オーバヘッドが低いので、運用の裏で解析をすることが可能かも

- トラブルを起こしていることや、そのプロセスは分かっているが、犯人までは分かっていない場合。
  - -犯人が分かっていて、動きをおいたい場合は他のツールで可能

6. perfこうなってほしいな

今後の期待

- ドキュメント
- perf diff

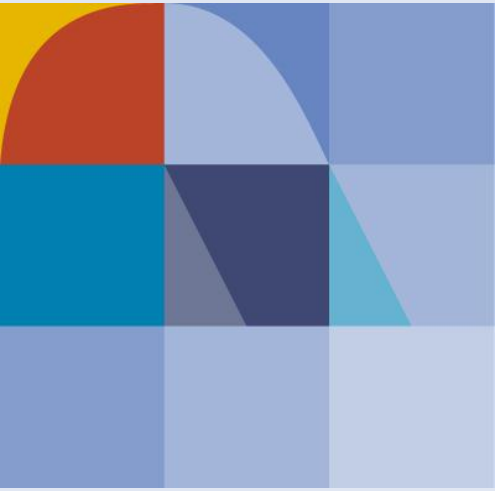
7. まとめ

まとめ

NTT DATA

◆ perfのいいとこ
- ❑システムで行われている処理を俯瞰できる。
- ❑プロセスごとの処理を俯瞰できる。
- ❑客観的な情報から、絞り込みがはかれる。
  →解析ビギナーでも容疑者を特定しやすい
- ❑オーバヘッドが低い
  →システム運用中でも使用しやすい
- ❑ツールの種類が豊富

◆ perfのもうちょっと頑張ってほしいとこ、注意点
- ❑ドキュメント
- ❑debuginfo



NTT DATA

変える力を、ともに生み出す。