4 章性能監視ツール付録

pg_statsinfo の利用例

1. 目的

pg_statsinfo は導入に関して、手厚いマニュアルが用意されています。しかし、その利用例など実際の運用に関する情報はあまりないようです。

そこで、実際の運用をイメージしやすくするために、pg_statsinfo の簡易レポートと pg_stats_reporter の例を提示しながら簡単な利用例を紹介します。

2. メモリ不足時のレポート

2.1. pg_statsinfo 簡易レポートの例

データベース単位のキャッシュヒット率は [Database Statistics] の[Cache Hit Ratio] 行で確認できます。

DB の使い方にもよりますが、キャッシュヒット率が 90%を下回っていたら、メモリ不足として shared_buffer の追加を検討するとよいでしょう。

ただし、この値は OS のキャッシュ利用などは考慮していません。こちらを利用することで性能を担保できているケースもある点に注意してください。

```
----------------------------------------
/* Database Statistics */
----------------------------------------
Database Name              : postgres
Database Size              : 6 MiB
Database Size Increase     : 0 MiB
Commit/s                   : 0.239
Rollback/s                 : 0.000
Cache Hit Ratio            : 100.0 %
Block Read/s (disk+cache)  : 16.855
Block Read/s (disk)        : 0.000
Rows Read/s                : 80.135
Temporary Files            : 0
Temporary Bytes            : 0 MiB
Deadlocks                  : 0
Block Read Time            : 0.000 ms
Block Write Time           : 0.000 ms
```

図 2.1: 簡易レポートでのデータベースキャッシュヒット率

テーブル単位のキャッシュヒット率は[Notable Tables] -> [Heavily Accessed Tables] 表の[Cache Hit Ratio(%)]列 で確認できます。ただし、この表はレポート期間内にシーケンシャルスキャンが 1 度以上発生したテーブルのうち、[Read Rows/Scan(シーケンシャルスキャン 1 回あたりの平均読み込み行数)] が上位 20 位のものしか表示されません。また、このキャッシュヒット率はテーブル、インデックス、TOAST(テーブルとインデックス)、全てを合算した数値です。

```
----------------------------------------
/* Notable Tables */
----------------------------------------

. . .

/** Heavily Accessed Tables **/
-----------------------------------
```

Database	Schema	Table	Seq Scans	Read Rows	Read Rows/Scan	Cache Hit Ratio(%)
tpcc_large	public	order_line	3	28818354	9606118.000	91.4
tpcc_large	public	stock	5	16000000	3200000.000	76.0
tpcc_large	public	customer	6	5760000	960000.000	63.2
tpcc_large	public	orders	3	2880000	960000.000	94.4
tpcc_large	public	history	2	1920000	960000.000	93.0
tpcc_large	public	new_orders	2	576000	288000.000	94.2
tpcc_large	public	item	3	300000	100000.000	92.9
statsinfo	statsrepo	database	3	3230	1076.667	99.9
statsinfo	statsrepo	memory	3	1073	357.667	92.0
statsinfo	statsrepo	xlog	6	2146	357.667	88.0
statsinfo	statsrepo	archive	6	2146	357.667	88.3
tpcc_large	public	district	5	1600	320.000	87.6
statsinfo	statsrepo	loadavg	17	4109	241.706	92.0
statsinfo	statsrepo	snapshot	105	24323	231.648	100.0
statsinfo	statsrepo	autovacuum	12	488	40.667	58.1
tpcc_large	public	warehouse	4	128	32.000	70.2
tpcc1	public	district	42000	420000	10.000	100.0
statsinfo	statsrepo	alert_message	3	4	1.333	66.7
statsinfo	statsrepo	instance	4	4	1.000	99.7

図 2.2: 簡易レポートでのテーブルキャッシュヒット率

以下のようなケースでは、直接スナップショットのデータを select する必要があります。

- Index スキャンしかされていないテーブルに関するキャッシュヒット率が知りたい
- TOAST など、特定の部分のみのキャッシュヒット率を知りたい
- [Read Rows/Scan] 上位 20 位に入っていないテーブルのキャッシュヒット率を知りたい
  ※この上位 20 位は、全データベース、全スキーマでの順位な点に注意

```
statsinfo=# SELECT
statsinfo-#         e.database,
statsinfo-#         e.schema,
statsinfo-#         e.table,
statsinfo-#         statsrepo.div(
statsinfo(#               statsrepo.sub(e.heap_blks_hit, b.heap_blks_hit),
statsinfo(#               statsrepo.sub(e.heap_blks_hit, b.heap_blks_hit) +
statsinfo(#               statsrepo.sub(e.heap_blks_read, b.heap_blks_read)
statsinfo(#               ) "キャッシュヒット(seq)",
statsinfo-#         statsrepo.div(
statsinfo(#               statsrepo.sub(e.idx_blks_hit, b.idx_blks_hit),
statsinfo(#               statsrepo.sub(e.idx_blks_hit, b.idx_blks_hit) +
statsinfo(#               statsrepo.sub(e.idx_blks_read, b.idx_blks_read)
statsinfo(#               ) "キャッシュヒット(index)",
statsinfo-#         statsrepo.div(
statsinfo(#               statsrepo.sub(e.toast_blks_hit, b.toast_blks_hit),
statsinfo(#               statsrepo.sub(e.toast_blks_hit, b.toast_blks_hit) +
statsinfo(#               statsrepo.sub(e.toast_blks_read, b.toast_blks_read)
statsinfo(#               ) "TOAST キャッシュヒット(index)",
statsinfo-#         statsrepo.div(
statsinfo(#               statsrepo.sub(e.tidx_blks_hit, b.tidx_blks_hit),
statsinfo(#               statsrepo.sub(e.tidx_blks_hit, b.tidx_blks_hit) +
statsinfo(#               statsrepo.sub(e.tidx_blks_read, b.tidx_blks_read)
statsinfo(#               ) "TOAST キャッシュヒット(index)"
statsinfo-# FROM
statsinfo-#         statsrepo.tables e LEFT JOIN statsrepo.table b
statsinfo-#                 ON e.tbl = b.tbl AND e.nsp = b.nsp AND e.dbid = b.dbid
statsinfo-# WHERE
statsinfo-#         b.snapid = 900 AND
statsinfo-#         e.snapid = 910 AND
statsinfo-#         e.table = 'customer' AND
statsinfo-#         e.database = 'tpcc1' AND
statsinfo-#         e.schema = 'public'
statsinfo-# ;

 database | schema |  table   | キャッシュヒット(seq) | キャッシュヒット(index) | TOAST キャッシュヒット(index) | TOAST キャッシュヒット(index)
----------+--------+----------+----------------------+------------------------+------------------------------+------------------------------
 tpcc1    | public | customer |                1.000 |                  1.000 |                        0.000 |                        0.000
(1 行)
```

図 2.3: テーブルキャッシュヒット率の詳細調査

インデックスのキャッシュヒット率は[Schema Information] -> [Indexes] 表の [Disk Reads]列と[Cache Reads]列より算出できます。ただし下記のように、この表では直接キャッシュヒット率を表示しません。値を知りたい場合は自分で以下の計算をする必要があります。

- [Cache Reads] / ([Disk Reads] + [Cache Reads])

```
----------------------------------------
/* Schema Information */
----------------------------------------

. . .

/** Indexes **/
----------------------------------------
```

Database	Schema	Index	Table	Size	Size Incr	Index Scans	Rows/Scan	Disk Reads	Cache Reads	Index Key
tpcc1	public	customer_ix1	customer	1 MiB	0 MiB	0	0.000	1	41999	c_w_id, c_d_id, c_last
tpcc1	public	customer_pk	customer	0 MiB	0 MiB	42000	1.000	2375	123964	c_w_id, c_d_id, c_id
tpcc1	public	district_pk	district	0 MiB	0 MiB	0	0.000	1	41999	d_w_id, d_id
tpcc1	public	item_pk	item	2 MiB	0 MiB	0	0.000	0	0	i_id
tpcc1	public	new_orders_pk	new_orders	0 MiB	0 MiB	0	0.000	0	0	no_w_id, no_d_id, no_o_id
tpcc1	public	order_line_pk	order_line	9 MiB	0 MiB	84000	4.988	4561	362554	ol_w_id, ol_d_id, ol_o_id, ol_number
tpcc1	public	orders_ix1	orders	0 MiB	0 MiB	42000	1.000	2373	123966	o_w_id, o_d_id, o_c_id
tpcc1	public	orders_pk	orders	0 MiB	0 MiB	42000	1.000	2332	123995	o_w_id, o_d_id, o_id
tpcc1	public	stock_pk	stock	2 MiB	0 MiB	7962775	1.000	368	16013516	s_w_id, s_i_id
tpcc1	public	warehouse_pk	warehouse	0 MiB	0 MiB	0	0.000	0	0	w_id
tpcc_large	public	customer_ix1	customer	37 MiB	37 MiB	0	0.000	4762	0	c_w_id, c_d_id, c_last
tpcc_large	public	customer_pk	customer	28 MiB	28 MiB	0	0.000	3700	0	c_w_id, c_d_id, c_id
tpcc_large	public	district_pk	district	0 MiB	0 MiB	0	0.000	2	0	d_w_id, d_id
tpcc_large	public	item_pk	item	2 MiB	2 MiB	2	1.000	279	1	i_id
tpcc_large	public	new_orders_pk	new_orders	8 MiB	8 MiB	1	288000.000	2218	0	no_w_id, no_d_id, no_o_id
tpcc_large	public	order_line_pk	order_line	288 MiB	288 MiB	1	9606118.000	71969	1829	ol_w_id, ol_d_id, ol_o_id, ol_number
tpcc_large	public	orders_ix1	orders	28 MiB	28 MiB	1	960000.000	7381	0	o_w_id, o_d_id, o_c_id
tpcc_large	public	orders_pk	orders	28 MiB	28 MiB	2	960000.000	10832	230	o_w_id, o_d_id, o_id
tpcc_large	public	stock_pk	stock	68 MiB	68 MiB	0	0.000	8777	0	s_w_id, s_i_id
tpcc_large	public	warehouse_pk	warehouse	0 MiB	0 MiB	0	0.000	2	0	w_id

図 2.4: 簡易レポートでのインデックスキャッシュヒット率

2.2. pg_stats_reporter の例

データベース単位のキャッシュヒット率は [Statistics] -> [Databases Statistics] 表の[Hit %]列で確認できます。(下図を参照)DB の使い方にもよりますが、キャッシュヒット率が 90%を下回っていたら、メモリ不足として shared_buffer の追加を検討しましょう。ただし、この値は OS のキャッシュ利用などは考慮していません。こちらを利用することで十分な性能を確保できているケースもある点に注意してください。

Statistics

Databases Statistics

Database	MiB	+MiB	Commit/s	Rollback/s	Hit%	Gets/s	Reads/s	Rows/s
tpcc_large	3242	3242	0.333	0.001	69.2	1195.087	368.270	4579.568
statsinfo	134	3	0.106	0.000	99.2	48.283	0.399	134.816
tpcc1	120	0	13.368	0.000	100.0	7954.341	1.912	9697.997
postgres	6	0	0.240	0.000	99.0	16.901	0.177	80.191

1/1 5

図 2.5: pg_stats_reporter でのデータベースキャッシュヒット率

テーブル単位のキャッシュヒット率は[Activities] -> [Notable Tables] -> [Heavily Updated Tables]表の[Hit ratio (%)]列 で確認できます。(下図を参照)

Heavily Accessed Tables

Database	Schema	Table	Seq scan	Seq tup read	Tup per seq	Hit ratio (%)
tpcc1	public	district	30000	300000	10.000	100.0
statsinfo	statsrepo	snapshot	71	16335	230.070	100.0
statsinfo	statsrepo	loadavg	12	2807	233.917	89.6
statsinfo	statsrepo	autovacuum	8	322	40.250	50.0
tpcc_large	public	customer	6	5760000	960000.000	63.2
tpcc_large	public	stock	5	16000000	3200000.000	76.0
tpcc_large	public	district	5	1600	320.000	87.6
statsinfo	statsrepo	xlog	4	1420	355.000	82.9
statsinfo	statsrepo	archive	4	1420	355.000	83.3
tpcc_large	public	warehouse	4	128	32.000	70.2
tpcc_large	public	order_line	3	28818354	9606118.000	91.4
tpcc_large	public	orders	3	2880000	960000.000	94.4
tpcc_large	public	item	3	300000	100000.000	92.9
statsinfo	statsrepo	instance	3	3	1.000	99.7
tpcc_large	public	history	2	1920000	960000.000	93.0
tpcc_large	public	new_orders	2	576000	288000.000	94.2
statsinfo	statsrepo	database	2	2132	1066.000	99.9
statsinfo	statsrepo	memory	2	710	355.000	89.6
statsinfo	statsrepo	alert_message	2	2	1.000	60.0

1/1 30

図 2.6: pg_stats_reporter でのテーブルキャッシュヒット率

ただし、この表はレポート期間内にシーケンシャルスキャンが 1 度以上発生したテーブルしか表示しません。また、このキャッシュヒット率はテーブル、インデックス、TOAST(テーブルとインデックス)、全てを合算した数値です。もしも以下のようなケースでは、直接スナップショットのデータを select する必要があります。(select 文の例は簡易レポートの項目を参照)

- Index スキャンしかされていないテーブルに関するキャッシュヒット率が知りたい
- TOAST など、特定の部分のみのキャッシュヒット率を知りたい

インデックスのキャッシュヒット率は[Miscellaneous] -> [Tables and Indexes] -> [Indexes] 表の [Reads]列と[Hits]列より算出できます。(下図を参照)

Indexes

Database	Schema	Index	Table	MiB	+MiB	Scans	Rows/scan	Reads	Hits	Keys
tpcc1	public	customer_ix1	customer	1	0	0	0.000	1	29999	c_w_id, c_d_id, c_last
tpcc1	public	customer_pk	customer	0	0	30000	1.000	2375	87874	c_w_id, c_d_id, c_id
tpcc1	public	district_pk	district	0	0	0	0.000	1	29999	d_w_id, d_id
tpcc1	public	item_pk	item	2	0	0	0.000	0	0	i_id
tpcc1	public	new_orders_pk	new_orders	0	0	0	0.000	0	0	no_w_id, no_d_id, no_o_id
tpcc1	public	order_line_pk	order_line	9	0	60000	4.989	4561	257619	ol_w_id, ol_d_id, ol_o_id, ol_number
tpcc1	public	orders_ix1	orders	0	0	30000	1.000	2373	87876	o_w_id, o_d_id, o_c_id
tpcc1	public	orders_pk	orders	0	0	30000	1.000	2332	87902	o_w_id, o_d_id, o_id
tpcc1	public	stock_pk	stock	2	0	5685951	1.000	368	11434534	s_w_id, s_i_id
tpcc1	public	warehouse_pk	warehouse	0	0	0	0.000	0	0	w_id

1/2 10

図 2.7: pg_stats_reporter でのインデックスキャッシュヒット率

ただし上記のように、この表では直接キャッシュヒット率を表示しません。値を知りたい場合は、自分で以下の計算をする必要があります。

- [Hits] / ([Reads] + [Hits])

2.3. 時系列情報の取得

pg_statsinfo 簡易レポート、pg_stats_reporter 共に、キャッシュヒット率を出力することができますが、あくまでレポート出力区間の平均値 1 つのみです。そのため、何れもレポート 1 つでは、日時によるヒット率の遷移などを知ることができません。

キャッシュヒット率の時系列での遷移を知るためには、複数区間のレポートを出力するか、直接スナップショットのデータを select する必要があります。例えば、特定テーブルにおける、各スナップショット間のキャッシュヒット率の遷移を知るには、以下の select 文の実行で可能です。

```
statsinfo=# select
statsinfo-#         s.time,
statsinfo-#         statsrepo.div(
statsinfo(#                         statsrepo.sub(
statsinfo(#                                     t.heap_blks_hit,
statsinfo(#                                     lag(t.heap_blks_hit, 1) over (order by t.snapid asc)
statsinfo(#                                     ),
statsinfo(#                         statsrepo.sub(
statsinfo(#                                     t.heap_blks_read,
statsinfo(#                                     lag(t.heap_blks_read, 1) over (order by t.snapid asc)
statsinfo(#                                     ) +
statsinfo(#                         statsrepo.sub(
statsinfo(#                                     t.heap_blks_hit,
statsinfo(#                                     lag(t.heap_blks_hit, 1) over (order by t.snapid asc)
statsinfo(#                                     )
statsinfo(#                         ) "キャッシュヒット率(seq)"
statsinfo-# from
statsinfo-#         statsrepo.tables t LEFT JOIN statsrepo.snapshot s
statsinfo-#                 on t.snapid = s.snapid
statsinfo-# where
statsinfo-#         t."table" = 'customer' and
statsinfo-#         t.database = 'tpcc1' and
statsinfo-#         t.schema = 'public'  and
statsinfo-#         s.time between '2015-09-14 00:00:00'::timestamp and '2015-09-14 19:00:00'::timestamp
statsinfo-# ;

            time              | キャッシュヒット率(seq)
------------------------------+------------------------
 2015-09-14 00:00:00.011973+09 |                  0.999
 2015-09-14 00:30:00.016698+09 |                  1.000
 2015-09-14 01:00:00.145519+09 |                  1.000
 2015-09-14 01:30:00.150055+09 |                  1.000
 2015-09-14 02:00:00.122768+09 |                  1.000
 2015-09-14 02:30:00.175287+09 |                  1.000
 2015-09-14 03:00:00.197908+09 |                  1.000
 2015-09-14 03:30:00.059208+09 |                  1.000
```

図 2.8: キャッシュヒット率の時系列での遷移情報の確認

3. チェックポイント過多時のレポート

3.1. pg_statsinfo 簡易レポートの例

チェックポイントに関する情報は、[Checkpoint Activity] で確認することができます。

```
--------------------------------------
/* Checkpoint Activity */
--------------------------------------
Total Checkpoints        : 143
Checkpoints By Time      : 76
Checkpoints By XLOG      : 65
Written Buffers Average  : 2843.119
Written Buffers Maximum  : 12411.000
Write Duration Average   : 9.198 sec
Write Duration Maximum   : 149.243 sec
```

図 3.1: 簡易レポートでのチェックポイント状況の確認

レポート取得期間内で ([Checkpoints By Time] + [Checkpoints By XLOG]) 回 checkpoint が発生していることが分かります。取得したレポート期間と比べてこの回数が多い場合は、対応するパラメータのチューニングをする必要があります。

- [Checkpoints By Time] はタイムアウトによるチェックポイント発生回数なので、checkpoint_timeout パラメータの値を増やすことで、チェックポイント発生回数を抑えることができる
- [Checkpoints By XLOG] は WAL 書き込みによるチェックポイント発生回数なので、checkpoint_segments パラメータの値を増やすことで、チェックポイント発生回数を抑えることができる

3.2. pg_stats_reporter の例

チェックポイントに関する情報は、[Maintenances] -> [Checkpoints] で確認することができます。

Maintenances

Checkpoints

Number of checkpoints	119
Caused by timeout	52
Caused by xlogs	65
Average written buffers	3410.697
Maximum written buffers	12411.000
Average checkpoint duration	10.484
Maximum checkpoint duration	149.243

図 3.2: pg_stats_reporter でのチェックポイント状況の確認

レポート取得期間内で ([Caused by timeout] + [Caused by xlogs]) 回 checkpoint が発生していることが分かります。取得したレポート期間と比べてこの回数が多い場合は、対応するパラメータのチューニングをする必要があります。

- [Caused by timeout] はタイムアウトによるチェックポイント発生回数なので、checkpoint_timeout パラメータの値を増やすことで、チェックポイント発生回数を抑えることができる
- [Caused by xlogs] は WAL 書き込みによるチェックポイント発生回数なので、checkpoint_segments パラメータの値を増やすことで、チェックポイント発生回数を抑えることができる

また、pg_stats_reporter は他の時系列グラフ上でチェックポイントの実行状況を確認できます。

図 3.3: pg_stats_reporter でのデータベースサイズとチェックポイント状況の重ね合わせ

一例として、データベースの容量(プロット)とチェックポイントの発生日時(赤で塗られた範囲)を合わせたグラフを載せました。9:00 以降にデータベースの容量が急激に増加し、それに合わせて高頻度の checkpoint が発生していることを確認できます。このようなグラフを用いることで、チェックポイントによる影響を受けそうな時刻を知ることができます。

4. Autovacuum 関係のレポート

4.1. pg_statsinfo 簡易レポートの例

[Autovacuum Activity] -> [Vacuum Basic Statistics (Average)] の項目で、Autovacuum に関する大まかな動向を把握することができます。

```
---------------------------------------
/* Autovacuum Activity */
---------------------------------------

/** Vacuum Basic Statistics (Average) **/
---------------------------------
Table                              Count  Removed Rows   Remain Rows   Remain Dead   Index Scans      Duration  Duration(Max)  Cancels
-------------------------------------------------------------------------------------------------------------------------------------
tpcc_large.pg_toast.pg_toast_2619      4        75.500        83.750         8.250         1.000     0.115 s       0.260 s        0
tpcc_large.pg_catalog.pg_statistic     1        67.000       482.000         0.000         1.000     0.090 s       0.090 s        0
```

図 4.1: 簡易レポートでの Autovacuum 実行状況の確認

レポートで指定した期間において Autovacuum が実行されたテーブルに関して、その結果回収された行や残った行、実施時間などを知ることができます。

ただし逆を言えば、Autovacuum が実施されていないテーブルはレポートに出てこないために、存在を見落としやすいです。尤もレポートには出てこなくとも、スナップショット上の情報としては全テーブルの vacuum 等の履歴は記録されています。そのため適宜スナップショットの情報を select することで、ある程度レポートの内容を補完することもできます。特に気になっているテーブルがある場合は、直接問い合わせても良いでしょう。

```
statsinfo=# select
statsinfo-#         s.time,
statsinfo-#         t.table,
statsinfo-#         t.last_autovacuum,
statsinfo-#         t.last_autoanalyze
statsinfo-# from
statsinfo-#         statsrepo.tables t LEFT JOIN statsrepo.snapshot s
statsinfo-#                 on t.snapid = s.snapid
statsinfo-# where
statsinfo-#         t."table" = 'customer' and
statsinfo-#         t.database = 'tpcc1' and
statsinfo-#         t.schema = 'public'  and
statsinfo-#         s.time between '2015-09-14 00:00:00'::timestamp and '2015-09-14 19:00:00'::timestamp
statsinfo-# ;
             time              |  table   |   last_autovacuum   |  last_autoanalyze
-------------------------------+----------+---------------------+---------------------
 2015-09-14 00:00:00.011973+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 00:30:00.016698+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 01:00:00.145519+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 01:30:00.150055+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 02:00:00.122768+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 02:30:00.175287+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 03:00:00.197908+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 03:30:00.059208+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 04:00:00.081345+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 04:30:00.176348+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 05:00:00.057491+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 05:30:00.067982+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 06:00:00.142911+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 06:30:00.191404+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 07:00:00.040937+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 07:30:00.190169+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 08:00:00.134319+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 08:30:00.193447+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 09:00:00.060906+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 09:30:00.194468+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 10:00:00.044055+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 10:30:00.076956+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 11:00:00.076419+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 11:30:00.087946+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 12:00:00.161177+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 12:30:00.090309+09 | customer | 2015-09-14 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 13:00:00.107821+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 13:30:00.085055+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 14:00:00.129338+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 14:30:00.008191+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 15:00:00.102508+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 15:30:00.147937+09 | customer | 2015-09-01 10:00:00 | 2015-09-01 10:00:00
 2015-09-14 16:00:00.003438+09 | customer | 2015-09-14 15:47:31 | 2015-09-14 15:47:31
 2015-09-14 16:30:00.034887+09 | customer | 2015-09-14 15:47:31 | 2015-09-14 15:47:31
 2015-09-14 17:00:00.135354+09 | customer | 2015-09-14 15:47:31 | 2015-09-14 15:47:31
 2015-09-14 17:30:00.033453+09 | customer | 2015-09-14 15:47:31 | 2015-09-14 15:47:31
 2015-09-14 18:00:00.006959+09 | customer | 2015-09-14 15:47:31 | 2015-09-14 15:47:31
 2015-09-14 18:30:00.070489+09 | customer | 2015-09-14 15:47:31 | 2015-09-14 15:47:31
(38 行)
```

図 4.2: 特定テーブルの Autovacuum 実行状況の確認

4.2. pg_stats_reporter の例

[Autovacuums] -> [Overview] の項目で、Autovacuum に関する大まかな動向を把握することができます。

Autovacuums

Overview

Database	Schema	Table	Count	Avg index scans	Avg removed rows	Avg remain rows	Avg remain dead	Avg duration (sec)	Max duration (sec)	Cancels
tpcc_large	pg_toast	pg_toast_2619	4	1.000	75.500	83.750	8.250	0.115	0.260	0
tpcc_large	pg_catalog	pg_statistic	1	1.000	67.000	482.000	0.000	0.090	0.090	0

1/1 10

図 4.3: pg_stats_reporter での Autovacuum 実行状況の確認

pg_stats_reporter も pg_statsinfo 簡易レポートと同様の情報を確認することができます。autovacuum されないテーブルは、レポートに出現しない点も、pg_statsinfo 簡易レポートと同様です。

5. ディスク使用量 関係のレポート

5.1. pg_statsinfo 簡易レポートの例

テーブルスペースに紐付くディスクの使用量は「Disk Usage」→「Disk Usage per Tablespace」から確認することができます。

```
-------------------------------------
/* Disk Usage */
-------------------------------------

/** Disk Usage per Tablespace **/
---------------------------------
Tablespace        Location                          Device              Used         Avail       Remain
-------------------------------------------------------------------------------------------------------
pg_default        /var/lib/pgsql/9.4/data           253:0            4668 MiB     12850 MiB      73.4 %
pg_global         /var/lib/pgsql/9.4/data           253:0            4668 MiB     12850 MiB      73.4 %
```

図 5.1: 簡易レポートでのディスク使用状況の確認

ただし、この情報は作成するレポート期間における、最新スナップショット時刻 1 点のデータです。もしも時系列のデータを確認したい場合は、複数期間のレポートを取得するか、直接 SQL 文でデータを取得する必要があります。

```
statsinfo=# select
statsinfo-#         s.time,
statsinfo-#         t.avail,
statsinfo-#         t.total,
statsinfo-#         100 * t.avail / t.total as "Remain(%)"
statsinfo-# from
statsinfo-#         statsrepo.tablespace t left join statsrepo.snapshot s
statsinfo-#                 on t.snapid = s.snapid
statsinfo-# where
statsinfo-#         t.name = 'pg_default' and
statsinfo-#         s.time between '2015-09-14 00:00:00'::timestamp and '2015-09-15 00:00:00'::timestamp
statsinfo-# order by s.time;

             time              |    avail    |    total    | Remain(%)
-------------------------------+-------------+-------------+----------
 2015-09-14 00:00:00.011973+09 | 16872951808 | 18369396736 |       91
 2015-09-14 00:30:00.016698+09 | 16887189504 | 18369396736 |       91
 2015-09-14 01:00:00.145519+09 | 16886767616 | 18369396736 |       91
 2015-09-14 01:30:00.150055+09 | 16886337536 | 18369396736 |       91
 2015-09-14 02:00:00.122768+09 | 16885907456 | 18369396736 |       91
 2015-09-14 02:30:00.175287+09 | 16885473280 | 18369396736 |       91
 2015-09-14 03:00:00.197908+09 | 16885055488 | 18369396736 |       91
 2015-09-14 03:30:00.059208+09 | 16884641792 | 18369396736 |       91
 2015-09-14 04:00:00.081345+09 | 16884219904 | 18369396736 |       91
 2015-09-14 04:30:00.176348+09 | 16883818496 | 18369396736 |       91
 2015-09-14 05:00:00.057491+09 | 16883400704 | 18369396736 |       91
 2015-09-14 05:30:00.067982+09 | 16882999296 | 18369396736 |       91
 2015-09-14 06:00:00.142911+09 | 16882573312 | 18369396736 |       91
 2015-09-14 06:30:00.191404+09 | 16882171904 | 18369396736 |       91
 2015-09-14 07:00:00.040937+09 | 16881754112 | 18369396736 |       91
 2015-09-14 07:30:00.190169+09 | 16881348608 | 18369396736 |       91
 2015-09-14 08:00:00.134319+09 | 16880926720 | 18369396736 |       91
 2015-09-14 08:30:00.193447+09 | 16880513024 | 18369396736 |       91
```

図 5.2: ディスク使用量の時系列での遷移情報の確認

簡易レポート はデータベース単位の容量もレポートしてくれます。(「Database Statistics」→「Database Size」)こちらは、時系列でのデータを出力してくれます。

```
--------------------------------------
/* Database Statistics */
--------------------------------------

. . .

/** Database Size **/
---------------------------------
DateTime          Database                    Size
-------------------------------------------------
2015-09-14 08:00  postgres                6.651 MiB
2015-09-14 08:00  statsinfo             131.331 MiB
2015-09-14 08:00  tpcc1                 120.198 MiB
2015-09-14 08:30  postgres                6.651 MiB
2015-09-14 08:30  statsinfo             131.612 MiB
2015-09-14 08:30  tpcc1                 120.198 MiB
2015-09-14 09:00  postgres                6.651 MiB
2015-09-14 09:00  statsinfo             131.932 MiB
2015-09-14 09:00  tpcc1                 120.198 MiB
2015-09-14 09:00  tpcc_large            810.268 MiB
2015-09-14 09:30  postgres                6.651 MiB
2015-09-14 09:30  statsinfo             132.284 MiB
2015-09-14 09:30  tpcc1                 120.198 MiB
2015-09-14 09:30  tpcc_large           3242.354 MiB
2015-09-14 10:00  postgres                6.651 MiB
2015-09-14 10:00  statsinfo             132.589 MiB
2015-09-14 10:00  tpcc1                 120.198 MiB
2015-09-14 10:00  tpcc_large           3242.354 MiB
2015-09-14 10:30  postgres                6.651 MiB
2015-09-14 10:30  statsinfo             132.917 MiB
2015-09-14 10:30  tpcc1                 120.198 MiB
2015-09-14 10:30  tpcc_large           3242.354 MiB
2015-09-14 11:00  postgres                6.651 MiB
2015-09-14 11:00  statsinfo             133.237 MiB
2015-09-14 11:00  tpcc1                 120.198 MiB
2015-09-14 11:00  tpcc_large           3242.354 MiB
2015-09-14 11:30  postgres                6.651 MiB
2015-09-14 11:30  statsinfo             133.549 MiB
2015-09-14 11:30  tpcc1                 120.198 MiB
2015-09-14 11:30  tpcc_large           3242.354 MiB
2015-09-14 12:00  postgres                6.651 MiB
2015-09-14 12:00  statsinfo             133.839 MiB
2015-09-14 12:00  tpcc1                 120.198 MiB
```

図 5.3: 簡易レポートでのデータベース単位のデータ量の確認

簡易レポート は更にテーブル単位の容量もレポートしてくれます。テーブル単位の容量という観点では、よく似た 2 つの表がレポートされます。

1 つめは「Schema Information」→「Tables」にある表です。データの増加量なども載っており、容量周りのレポートとしてこちらをメインで使うとよいでしょう。

```
----------------------------------------
/* Schema Information */
----------------------------------------

/** Tables **/
-----------------------------------
```

Database	Schema	Table	Columns	Rows	Size	Size Incr	Table Scans	Index Scans
tpcc1	public	customer	21	30000	17 MiB	0 MiB	0	42000
tpcc1	public	district	11	10	0 MiB	0 MiB	42000	0
tpcc1	public	history	8	30000	2 MiB	0 MiB	0	0
tpcc1	public	item	5	100000	10 MiB	0 MiB	0	0
tpcc1	public	new_orders	3	9000	0 MiB	0 MiB	0	0
tpcc1	public	order_line	10	299671	28 MiB	0 MiB	0	84000
tpcc1	public	orders	8	30000	1 MiB	0 MiB	0	84000
tpcc1	public	stock	17	100000	33 MiB	0 MiB	0	7962775
tpcc1	public	warehouse	9	1	0 MiB	0 MiB	0	0
tpcc_large	public	customer	21	960052	565 MiB	565 MiB	6	0
tpcc_large	public	district	11	320	0 MiB	0 MiB	5	0
tpcc_large	public	history	8	960000	81 MiB	81 MiB	2	0
tpcc_large	public	item	5	100000	10 MiB	10 MiB	3	2
tpcc_large	public	new_orders	3	288000	12 MiB	12 MiB	2	1
tpcc_large	public	order_line	10	9606021	926 MiB	926 MiB	3	1
tpcc_large	public	orders	8	960000	62 MiB	62 MiB	3	3
tpcc_large	public	stock	17	3199939	1082 MiB	1082 MiB	5	0
tpcc_large	public	warehouse	9	32	0 MiB	0 MiB	4	0

図 5.4: 簡易レポートの'Schema Information'項目でのテーブルデータ量の確認

2つめは「Disk Usage」→「Disk Usage per Table」にある表です。各種 Read 量も併せて確認することができます。しかし、ここで出力されるテーブルは、レポート期間において総 read 量がトップ 10 位のテーブルのみなので、容量という観点では、補助的な役割で使うとよいでしょう。

```
----------------------------------------
/* Disk Usage */
----------------------------------------

・・・

/** Disk Usage per Table **/
-----------------------------------
```

Database	Schema	Table	Size	Table Reads	Index Reads	Toast Reads
tpcc_large	public	stock	1082 MiB	1140819	8777	0
tpcc_large	public	order_line	926 MiB	878602	71969	0
tpcc_large	public	customer	565 MiB	701818	8462	0
tpcc_large	public	orders	62 MiB	96066	18213	0
tpcc_large	public	history	81 MiB	74832	0	0
tpcc_large	public	new_orders	12 MiB	16166	2218	0
tpcc1	public	order_line	28 MiB	7565	4561	0
tpcc_large	public	item	10 MiB	7799	279	0
tpcc1	public	customer	17 MiB	5675	2376	0
tpcc1	public	orders	1 MiB	2836	4705	0

図 5.5: 簡易レポートの'Disk Usage'項目でのテーブルデータ量の確認

レポートこそ出力されないものの、テーブル単位の容量の情報もスナップショットとして取られているため、直接 select すれば時系列のデータを取得できます。特に注意したいテーブルがある場合は、データの定期取得を検討しても良いかもしれません。

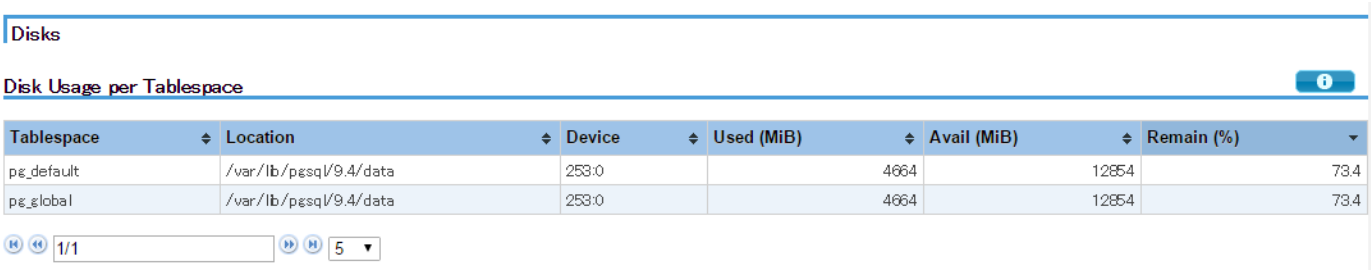
```
statsinfo=# select
statsinfo-#         s.time,
statsinfo-#         t.name,
statsinfo-#         t.size
statsinfo-# from
statsinfo-#         statsrepo.table t  left join statsrepo.snapshot s
statsinfo-#                 on t.snapid = s.snapid
statsinfo-# where
statsinfo-#         t.dbid = 19151   and
statsinfo-#         t.nsp  = 2200    and
statsinfo-#         t.name = 'stock' and
statsinfo-#         s.time between '2015-09-14 00:00:00'::timestamp and '2015-09-15 00:00:00'::timestamp
statsinfo-# order by s.time;

             time              | name  |    size
-------------------------------+-------+------------
 2015-09-14 09:00:00.060906+09 | stock |  283820032
 2015-09-14 09:30:00.194468+09 | stock | 1135247360
 2015-09-14 10:00:00.044055+09 | stock | 1135247360
 2015-09-14 10:30:00.076956+09 | stock | 1135247360
 2015-09-14 11:00:00.076419+09 | stock | 1135247360
 2015-09-14 11:30:00.087946+09 | stock | 1135247360
 2015-09-14 12:00:00.161177+09 | stock | 1135247360
 2015-09-14 12:30:00.090309+09 | stock | 1135247360
 2015-09-14 13:00:00.107821+09 | stock | 1135247360
 2015-09-14 13:30:00.085055+09 | stock | 1135247360
 2015-09-14 14:00:00.129338+09 | stock | 1135247360
 2015-09-14 14:30:00.008191+09 | stock | 1135247360
 2015-09-14 15:00:00.102508+09 | stock | 1135247360
 2015-09-14 15:30:00.147937+09 | stock | 1135247360
 2015-09-14 16:00:00.003438+09 | stock | 1135247360
 2015-09-14 16:30:00.034887+09 | stock | 1135247360
 2015-09-14 17:00:00.135354+09 | stock | 1135247360
 2015-09-14 17:30:00.033453+09 | stock | 1135247360
 2015-09-14 18:00:00.006959+09 | stock | 1135247360
 2015-09-14 18:30:00.070489+09 | stock | 1135247360
 2015-09-14 19:00:00.186174+09 | stock | 1135247360
 2015-09-14 19:30:00.142485+09 | stock | 1135247360
 2015-09-14 20:00:00.003807+09 | stock | 1135247360
 2015-09-14 20:30:00.090117+09 | stock | 1135247360
 2015-09-14 21:00:00.183505+09 | stock | 1135247360
 2015-09-14 21:30:00.101526+09 | stock | 1135247360
 2015-09-14 22:00:00.133769+09 | stock | 1135247360
 2015-09-14 22:30:00.138867+09 | stock | 1135247360
 2015-09-14 23:00:00.15604+09  | stock | 1135247360
 2015-09-14 23:30:00.165511+09 | stock | 1135247360
```

図 5.6: テーブル容量の時系列での遷移情報の確認

5.2. pg_stats_reporter の例

テーブルスペースに紐付くディスクの使用量は「OS Resources」→「Disks」→「Disk Usage per Tablespace」から確認することができます。



Disks

Disk Usage per Tablespace

Tablespace	Location	Device	Used (MiB)	Avail (MiB)	Remain (%)
pg_default	/var/lib/pgsql/9.4/data	253:0	4664	12854	73.4
pg_global	/var/lib/pgsql/9.4/data	253:0	4664	12854	73.4

1/1 5

図 5.7: pg_stats_reporter でのディスク使用状況の確認

pg_statsinfo 簡易レポートと同様、pg_stats_reporter が提供するのも時刻 1 点のデータです。そのため、もし時系列データを確認したい場合は、簡易レポートの項目と同様、直接 SQL 文でデータを取得する必要があります。

pg_stats_reporter はデータベース単位の容量もレポートしてくれます。(「Statistics」→「Databases Statistics」→「Database Size」)こちらは、時系列でのプロットを表示してくれます。そのため、pg_stats_reporter を用いた運用では、データ量の遷移をデータベース単位で確認するとやりやすいでしょう。

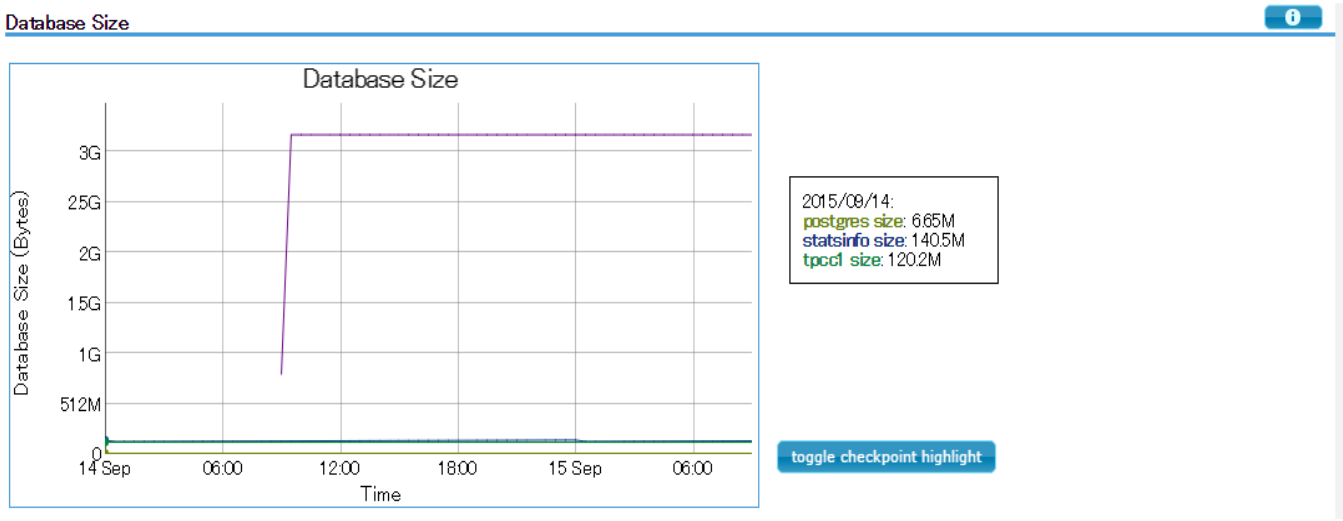


図 5.8: pg_stats_reporter でのデータベース単位のデータ量の確認

pg_stats_reporter は更にテーブル単位の容量もレポートしてくれます。テーブル単位の容量という観点では、よく似た 2 つの表がレポートされるので、好みで使い分けると良いでしょう。

1 つめは「Miscellaneous」→「Tables and Indexes」→「Tables」にある表です。データの増加量なども載っており、容量周りのレポートとしてはこちらの方がオーソドックスで使いやすいと思います。

2 つめは「OS Resources」→「Disks」→「Disk Usage per Table」にある表です。こちらには円グラフも添えられており、全体の使用量の比率も併せて確認できるのが利点です。

Miscellaneous

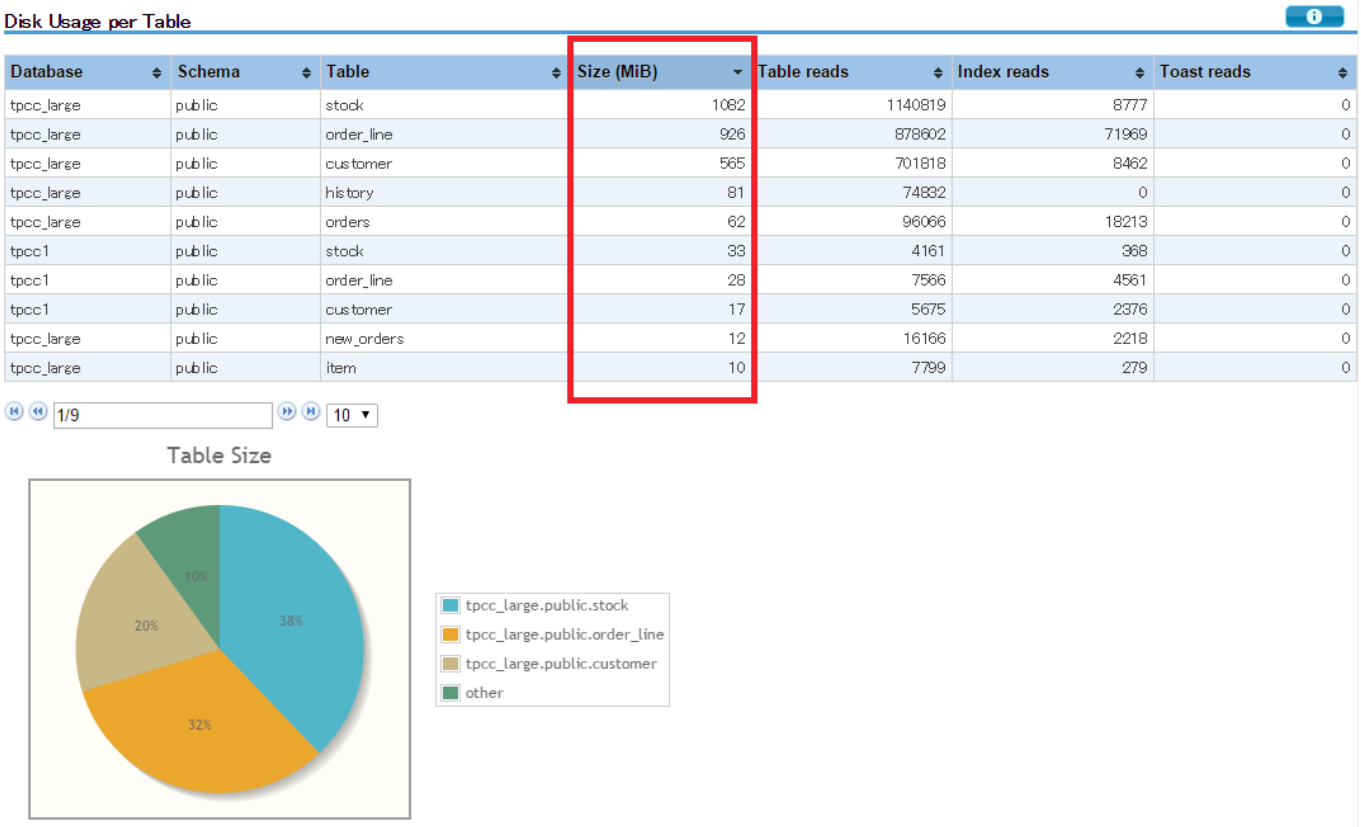
Tables and Indexes

Tables

Database	Schema	Table	Columns	Rows	MiB	+MiB	Table scans	Index scans
tpcc1	public	customer	21	30000	17	0	0	198000
tpcc1	public	district	11	10	0	0	198000	0
tpcc1	public	history	8	30000	2	0	0	0
tpcc1	public	item	5	100000	10	0	0	0
tpcc1	public	new_orders	3	9000	0	0	0	0
tpcc1	public	order_line	10	299671	28	0	0	396000
tpcc1	public	orders	8	30000	1	0	0	396000
tpcc1	public	stock	17	100000	33	0	0	37544759
tpcc1	public	warehouse	9	1	0	0	0	0
tpcc_large	public	customer	21	960052	565	565	6	0
tpcc_large	public	district	11	320	0	0	5	0
tpcc_large	public	history	8	960000	81	81	2	0
tpcc_large	public	item	5	100000	10	10	3	2
tpcc_large	public	new_orders	3	288000	12	12	2	1
tpcc_large	public	order_line	10	9606021	926	926	3	1
tpcc_large	public	orders	8	960000	62	62	3	3
tpcc_large	public	stock	17	3199939	1082	1082	5	0
tpcc_large	public	warehouse	9	32	0	0	4	0

1/1 30

図 5.9: pg_stats_reporter の'Miscellaneous'項目でのテーブル単位のデータ量の確認



Disk Usage per Table

Database	Schema	Table	Size (MiB)	Table reads	Index reads	Toast reads
tpcc_large	public	stock	1082	1140819	8777	0
tpcc_large	public	order_line	926	878602	71969	0
tpcc_large	public	customer	565	701818	8462	0
tpcc_large	public	history	81	74832	0	0
tpcc_large	public	orders	62	96066	18213	0
tpcc1	public	stock	33	4161	368	0
tpcc1	public	order_line	28	7566	4561	0
tpcc1	public	customer	17	5675	2376	0
tpcc_large	public	new_orders	12	16166	2218	0
tpcc_large	public	item	10	7799	279	0

1/9 10

Table Size

図 5.10: pg_stats_reporter の'OS Resource'項目でのテーブル単位のデータ量の確認

レポートこそ出力されないものの、テーブル単位の容量の情報もスナップショットとして取られているため、直接 select すれば時系列のデータを取得できます。特に注意したいテーブルがある場合は、簡易レポートの項目にあるような SQL 文でデータの定期取得を検討しても良いでしょう。

6. インデックス使用 関係のレポート

6.1. pg_statsinfo 簡易レポートの例

pg_statsinfo 簡易レポートのうち、テーブルとインデックスのデータ 2 つがインデックス使用状況の判断に活用できます。まずテーブルの情報として、「Schema Information」→「Tables」表の「Table Scans」列と「Index Scans」列が参考になります。

この表のうち、「Table Scans」はシーケンシャルスキャン回数、「Index Scans」はインデックススキャン回数です。
インデックススキャンを念頭に入れているテーブルなのに、シーケンシャルスキャン数が多い場合は、SQL 文の見直しや新しいインデックスの作成を検討すべきでしょう。

```
---------------------------------------
/* Schema Information */
---------------------------------------

/** Tables **/
-----------------------------------
```

Database	Schema	Table	Columns	Rows	Size	Size Incr	Table Scans	Index Scans
tpcc1	public	customer	21	30000	17 MiB	0 MiB	0	42000
tpcc1	public	district	11	10	0 MiB	0 MiB	42000	0
tpcc1	public	history	8	30000	2 MiB	0 MiB	0	0
tpcc1	public	item	5	100000	10 MiB	0 MiB	0	0
tpcc1	public	new_orders	3	9000	0 MiB	0 MiB	0	0
tpcc1	public	order_line	10	299671	28 MiB	0 MiB	0	84000
tpcc1	public	orders	8	30000	1 MiB	0 MiB	0	84000
tpcc1	public	stock	17	100000	33 MiB	0 MiB	0	7962775
tpcc1	public	warehouse	9	1	0 MiB	0 MiB	0	0
tpcc_large	public	customer	21	960052	565 MiB	565 MiB	6	0
tpcc_large	public	district	11	320	0 MiB	0 MiB	5	0
tpcc_large	public	history	8	960000	81 MiB	81 MiB	2	0
tpcc_large	public	item	5	100000	10 MiB	10 MiB	3	2
tpcc_large	public	new_orders	3	288000	12 MiB	12 MiB	2	1
tpcc_large	public	order_line	10	9606021	926 MiB	926 MiB	3	1
tpcc_large	public	orders	8	960000	62 MiB	62 MiB	3	3
tpcc_large	public	stock	17	3199939	1082 MiB	1082 MiB	5	0
tpcc_large	public	warehouse	9	32	0 MiB	0 MiB	4	0

図 6.1: 簡易レポートの'Tables'表によるインデックス使用状況の確認

他にテーブルの情報は、「Disk Usage」→「Disk Usage per Table」にある表にも存在しています。ただし、こちらの表の数値は回数単位ではなく容量単位なので、両スキャンの比較は少し難しいかもしれません。また、載っているテーブルも総 read 量が Top10 のもののみなため、インデックス使用状況の調査時にはあくまで参考程度の位置付けでしょう。

```
---------------------------------------
/* Disk Usage */
---------------------------------------

. . .

/** Disk Usage per Table **/
-----------------------------------
```

Database	Schema	Table	Size	Table Reads	Index Reads	Toast Reads
tpcc_large	public	stock	1082 MiB	1140819	8777	0
tpcc_large	public	order_line	926 MiB	878602	71969	0
tpcc_large	public	customer	565 MiB	701818	8462	0
tpcc_large	public	orders	62 MiB	96066	18213	0
tpcc_large	public	history	81 MiB	74832	0	0
tpcc_large	public	new_orders	12 MiB	16166	2218	0
tpcc1	public	order_line	28 MiB	7565	4561	0
tpcc_large	public	item	10 MiB	7799	279	0
tpcc1	public	customer	17 MiB	5675	2376	0
tpcc1	public	orders	1 MiB	2836	4705	0

図 6.2: 簡易レポートの'Disk Usage'項目によるインデックス使用状況の確認

インデックスの情報は、「Schema Information」→「Indexes」表が参考になります。

```
----------------------------------------
/* Schema Information */
----------------------------------------

. . .

/** Indexes **/
----------------------------------
```

Database	Schema	Index	Table	Size	Size Incr	Index Scans	Rows/Scan	Disk Reads	Cache Reads	Index Key
tpcc1	public	customer_ix1	customer	1 MiB	0 MiB	0	0.000	1	41999	c_w_id, c_d_i
tpcc1	public	customer_pk	customer	0 MiB	0 MiB	42000	1.000	2375	123964	c_w_id, c_d_i
tpcc1	public	district_pk	district	0 MiB	0 MiB	0	0.000	1	41999	d_w_id, d_id
tpcc1	public	item_pk	item	2 MiB	0 MiB	0	0.000	0	0	i_id
tpcc1	public	new_orders_pk	new_orders	0 MiB	0 MiB	0	0.000	0	0	no_w_id, no_d
tpcc1	public	order_line_pk	order_line	9 MiB	0 MiB	84000	4.988	4561	362554	ol_w_id, ol_d
tpcc1	public	orders_ix1	orders	0 MiB	0 MiB	42000	1.000	2373	123966	o_w_id, o_d_i
tpcc1	public	orders_pk	orders	0 MiB	0 MiB	42000	1.000	2332	123995	o_w_id, o_d_i
tpcc1	public	stock_pk	stock	2 MiB	0 MiB	7962775	1.000	368	16013516	s_w_id, s_i_i
tpcc1	public	warehouse_pk	warehouse	0 MiB	0 MiB	0	0.000	0	0	w_id
tpcc_large	public	customer_ix1	customer	37 MiB	37 MiB	0	0.000	4762	0	c_w_id, c_d_i
tpcc_large	public	customer_pk	customer	28 MiB	28 MiB	0	0.000	3700	0	c_w_id, c_d_i
tpcc_large	public	district_pk	district	0 MiB	0 MiB	0	0.000	2	0	d_w_id, d_id
tpcc_large	public	item_pk	item	2 MiB	2 MiB	2	1.000	279	1	i_id
tpcc_large	public	new_orders_pk	new_orders	8 MiB	8 MiB	1	288000.000	2218	0	no_w_id, no_
tpcc_large	public	order_line_pk	order_line	288 MiB	288 MiB	1	9606118.000	71969	1829	ol_w_id, ol
tpcc_large	public	orders_ix1	orders	28 MiB	28 MiB	1	960000.000	7381	0	o_w_id, o_d_
tpcc_large	public	orders_pk	orders	28 MiB	28 MiB	2	960000.000	10832	230	o_w_id, o_d_
tpcc_large	public	stock_pk	stock	68 MiB	68 MiB	0	0.000	8777	0	s_w_id, s_i_i
tpcc_large	public	warehouse_pk	warehouse	0 MiB	0 MiB	0	0.000	2	0	w_id

図 6.3: 簡易レポートの'Indexes'表によるインデックス使用状況の確認

特に、インデックス使用数(Index Scans)が 0 のものには注意が必要です。長い間使われていないインデックスについては、このインデックスを利用する予定だったクエリの見直しや、このインデックス自体の削除を検討したほうが良いでしょう。

6.2. pg_stats_reporter の例

pg_stats_reporter が出力するレポートのうち、テーブルとインデックスのデータ 2 つがインデックス使用状況の判断に使用できます。まずテーブルの情報として、「Miscellaneous」→「Tables and Indexes」→「Tables」表の「Table scans」列と「Index scans」列が参考になります。

「Table scans」はシーケンシャルスキャン回数、「Index scans」はインデックススキャン回数です。インデックススキャンを念頭に入れているテーブルなのに、シーケンシャルスキャン数が多い場合は、SQL 文の見直しや新しいインデックスの作成を検討すべきでしょう。

Miscellaneous

Tables and Indexes

Tables

Database	Schema	Table	Columns	Rows	MiB	+MiB	Table scans	Index scans
tpcc1	public	customer	21	30000	17	0	0	198000
tpcc1	public	district	11	10	0	0	198000	0
tpcc1	public	history	8	30000	2	0	0	0
tpcc1	public	item	5	100000	10	0	0	0
tpcc1	public	new_orders	3	9000	0	0	0	0
tpcc1	public	order_line	10	299671	28	0	0	396000
tpcc1	public	orders	8	30000	1	0	0	396000
tpcc1	public	stock	17	100000	33	0	0	37544759
tpcc1	public	warehouse	9	1	0	0	0	0
tpcc_large	public	customer	21	960052	565	565	6	0
tpcc_large	public	district	11	320	0	0	5	0
tpcc_large	public	history	8	960000	81	81	2	0
tpcc_large	public	item	5	100000	10	10	3	2
tpcc_large	public	new_orders	3	288000	12	12	2	1
tpcc_large	public	order_line	10	9606021	926	926	3	1
tpcc_large	public	orders	8	960000	62	62	3	3
tpcc_large	public	stock	17	3199939	1082	1082	5	0
tpcc_large	public	warehouse	9	32	0	0	4	0

1/1 30

図 6.4: pg_stats_reporter の'Miscellaneous'項目によるインデックス使用状況の確認

他にテーブルの情報は、「OS Resources」→「Disks」→「Disk Usage per Table」にある表にも存在しています。ただし、こちらの表の数値は回数単位ではなく容量単位なので、両スキャンの比較は少し難しいかもしれません。

Disk Usage per Table

Database	Schema	Table	Size (MiB)	Table reads	Index reads	Toast reads
tpcc_large	public	stock	1082	1140819	8777	0
tpcc_large	public	order_line	926	878602	71969	0
tpcc_large	public	customer	565	701818	8462	0
tpcc_large	public	history	81	74832	0	0
tpcc_large	public	orders	62	96066	18213	0
tpcc1	public	stock	33	4161	368	0
tpcc1	public	order_line	28	7566	4561	0
tpcc1	public	customer	17	5675	2376	0
tpcc_large	public	new_orders	12	16166	2218	0
tpcc_large	public	item	10	7799	279	0

1/9 10

図 6.5: pg_stats_reporter の'OS Resource'項目によるインデックス使用状況の確認

インデックスの自身の情報は、「Miscellaneous」→「Tables and Indexes」→「Indexes」表を参照するとよいでしょう。

Indexes

Database	Schema	Index	Table	MiB	+MiB	Scans	Rows/scan	Reads	Hits	Keys
tpcc1	public	customer_ix1	customer	1	0	0	0.000	1	197999	c_w_id, c_d_id, c_last
tpcc1	public	customer_pk	customer	0	0	198000	1.000	2375	593164	c_w_id, c_d_id, c_id
tpcc1	public	district_pk	district	0	0	0	0.000	1	197999	d_w_id, d_id
tpcc1	public	item_pk	item	2	0	0	0.000	0	0	i_id
tpcc1	public	new_orders_pk	new_orders	0	0	0	0.000	0	0	no_w_id, no_d_id, no_o_id
tpcc1	public	order_line_pk	order_line	9	0	396000	4.992	4561	1726292	ol_w_id, ol_d_id, ol_o_id, ol_number
tpcc1	public	orders_ix1	orders	0	0	198000	1.000	2373	593166	o_w_id, o_d_id, o_c_id
tpcc1	public	orders_pk	orders	0	0	198000	1.000	2332	593188	o_w_id, o_d_id, o_id
tpcc1	public	stock_pk	stock	2	0	37544759	1.000	368	75504589	s_w_id, s_i_id
tpcc1	public	warehouse_pk	warehouse	0	0	0	0.000	0	0	w_id
tpcc_large	public	customer_ix1	customer	37	37	0	0.000	4762	0	c_w_id, c_d_id, c_last
tpcc_large	public	customer_pk	customer	28	28	0	0.000	3700	0	c_w_id, c_d_id, c_id
tpcc_large	public	district_pk	district	0	0	0	0.000	2	0	d_w_id, d_id
tpcc_large	public	item_pk	item	2	2	2	1.000	279	1	i_id
tpcc_large	public	new_orders_pk	new_orders	8	8	1	288000.000	2218	0	no_w_id, no_d_id, no_o_id
tpcc_large	public	order_line_pk	order_line	288	288	1	9606118.000	71969	1829	ol_w_id, ol_d_id, ol_o_id, ol_number
tpcc_large	public	orders_ix1	orders	28	28	1	960000.000	7381	0	o_w_id, o_d_id, o_c_id
tpcc_large	public	orders_pk	orders	28	28	2	960000.000	10832	230	o_w_id, o_d_id, o_id
tpcc_large	public	stock_pk	stock	68	68	0	0.000	8777	0	s_w_id, s_i_id
tpcc_large	public	warehouse_pk	warehouse	0	0	0	0.000	2	0	w_id

1/1 20

図 6.6: pg_stats_reporter の'Indexes'表によるインデックス使用状況の確認

特に、インデックス使用数(Scans)が 0 のものには注意が必要です。長い間使われていないインデックスについては、このインデックスを利用する予定だったクエリの見直しや、このインデックス自体の削除を検討したほうが良いでしょう。

pg_monz の利用例

1. 目的

pg_monz は導入に関して分かりやすいマニュアルが用意されています。しかし、その利用例など実際の運用に関する情報はあまりないようです。

そこで、実際の運用をイメージしやすくするために、pg_monz のグラフを提示しながら簡単な利用例を紹介します。なお、pg_monz は性能情報に関する監視だけでなく、死活監視など監視全般の機能を有しています。しかし、ここでは性能情報に関するトピックに絞ります。

2. メモリ不足時のレポート

データベース単位のキャッシュヒット率は「監視データ」→「最新データ」→「pg.stat_database」→「[DB 名] Cache hit ratio (%)」で確認することができます。

デフォルトではキャッシュヒット率 90%を閾値にトリガー設定されており、それを破線で確認することができます。DB の使い方にもよりますが、キャッシュヒット率がこれを下回っていたら、メモリ不足として shared_buffer の追加を検討すると良いでしょう。

ただし、この値は OS のキャッシュ利用などは考慮していません。こちらを利用することで性能を担保できているケースもある点に注意してください。

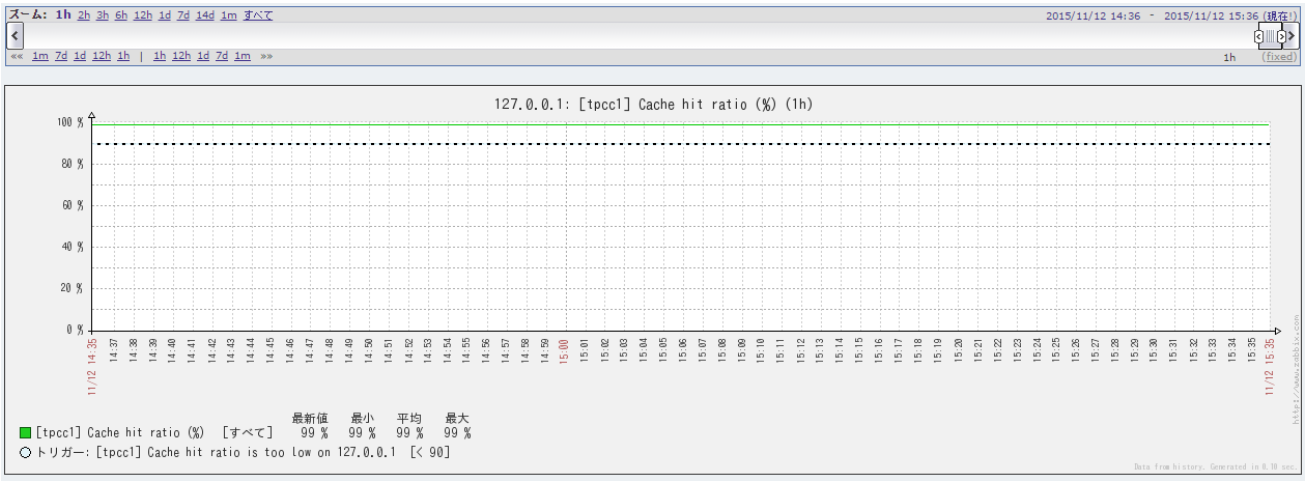


図 2.1:「最新データ」機能でのデータベースキャッシュヒット率の確認

pg_monz はこのグラフとは別に、データベースのキャッシュヒット率のカスタムグラフを作る設定も入っています。これは「監視データ」→「グラフ」→「[DB 名] Cache hit ratio」で確認することができます。しかし、検証対象のバージョンではこれは「最新データ」で閲覧できるグラフと全く同じものです。今後のバージョンで、新しい機能が追加されるかもしれませんが、今のところ両者を使い分ける必要はなさそうです。

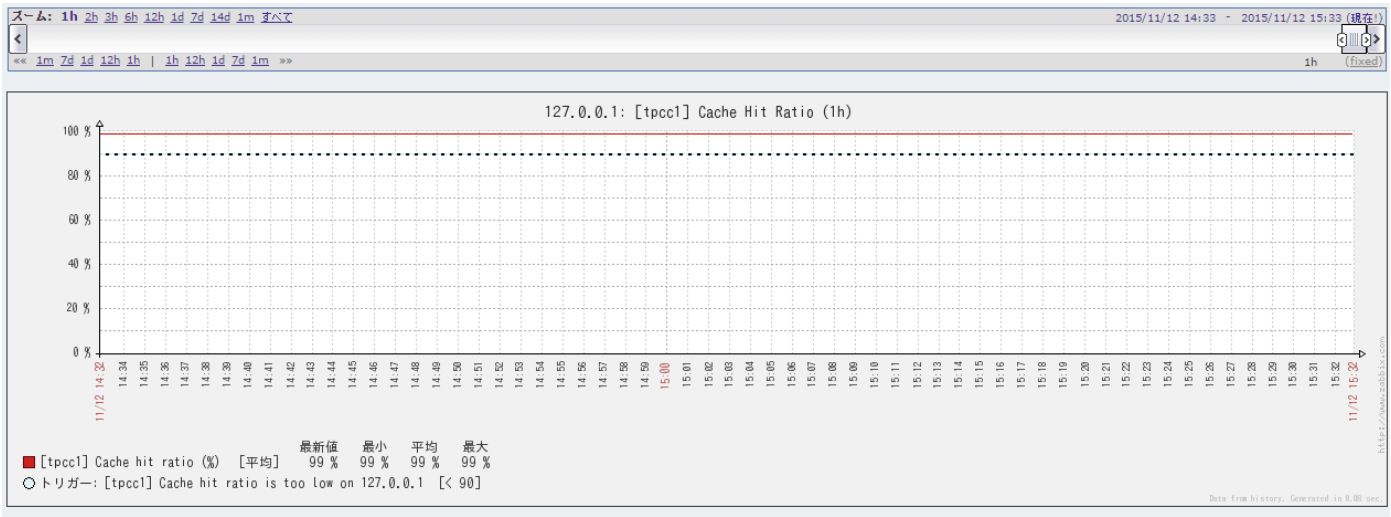


図 2.2:「カスタムグラフ」機能でのデータベースキャッシュヒット率の確認

テーブル単位のキャッシュヒット率は「監視データ」→「最新データ」→「「pg.stat_table」→ 「[DB 名] (スキーマ名.テーブル名) heap cache hit ratio %」で確認できます。このデータは TOAST を含まない純粋なテーブル部分に関するキャッシュヒット率の値です。もしも監視データベースに過剰に TOAST が存在している表が存在しており、TOAST 部分のキャッシュヒット率を確認したい場合は、Zabbix にその監視情報を追加しましょう。

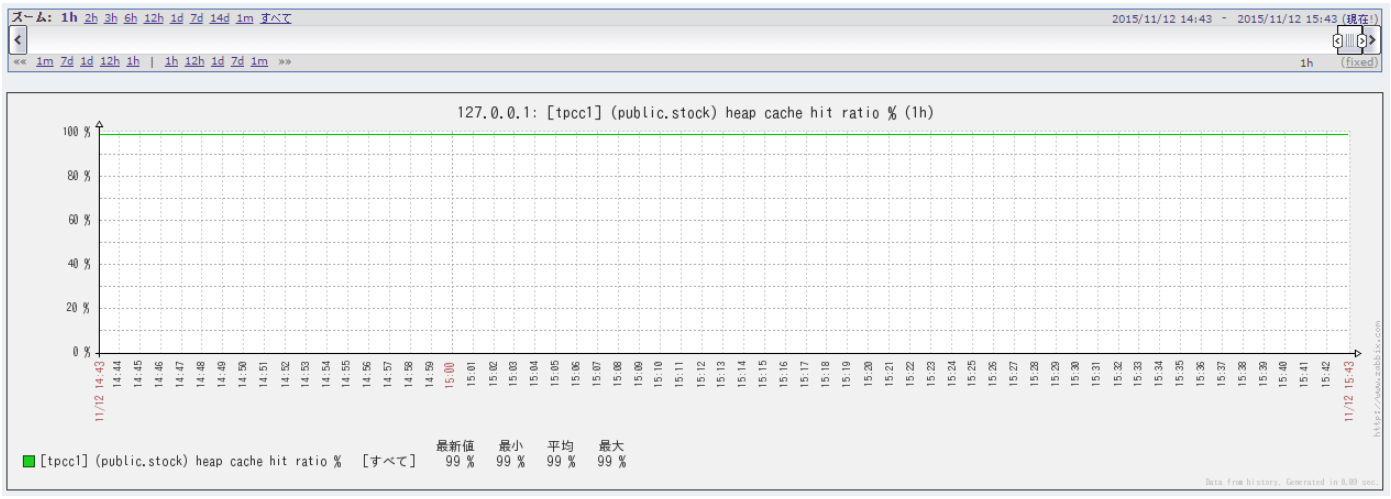


図 2.3:「最新データ」機能でのテーブルのキャッシュヒット率

pg_monz はインデックス単位の情報を取得していないので、インデックスのキャッシュヒット率を直接確認することはできません。ただ、テーブルに紐付く全インデックス合計のキャッシュヒット率(pg_statio_user_tables ビューの情報)は取得しているので、こちらである程度代用することが可能です。

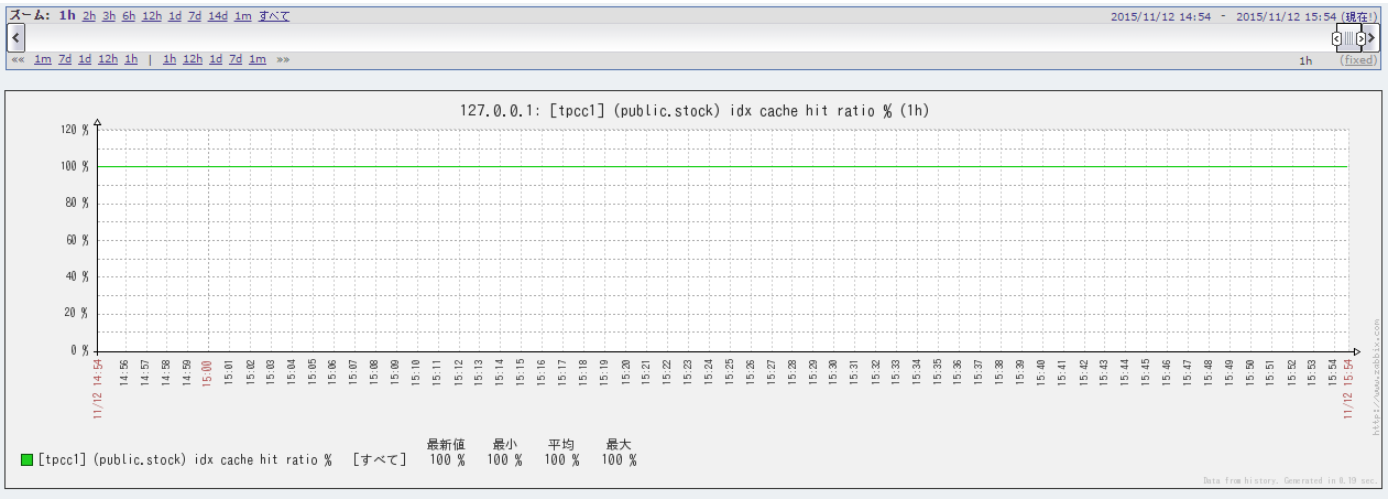


図 2.4:「最新データ」機能でのテーブルに紐付く、全インデックス合計のキャッシュヒット率

なお、ここで紹介したようなテーブル単位の情報を取得するようにすると、取得するアイテム数も多くなりがちです。アイテム数が多すぎると、「閲覧したいグラフを探すのが難しい」「アイテム一覧表示した時にブラウザへの負荷が大きくなってしまう」といったオペレーション上の問題点が発生します。これは Zabbix の仕様上やむを得ませんので、そのような環境で監視をすることになった際は、「グラフなどの ID から Zabbix の URL のパスを生成するルールを理解しておく」「WebAPI を活用する」といったような、オペレーション上のテクニックを抑えておくと良いでしょう。

3. チェックポイント過多時のレポート

pg_monz はチェックポイントに関して、以下の 3 つの情報を取得しています。

- チェックポイントによる書き込み量
- WAL 書き込み量によるチェックポイント発生タイミング
- 時間経過によるチェックポイント発生タイミング

これらはそれぞれ以下の箇所にて確認可能です。

- 「監視データ」→「最新データ」→「pg.bgwriter」→ 「Buffers_checkpoint」
- 「監視データ」→「最新データ」→「pg.bgwriter」→ 「Checkpoint count (by checkpoint_segments)」
- 「監視データ」→「最新データ」→「pg.bgwriter」→ 「Checkpoint count (by checkpoint_timeout)」

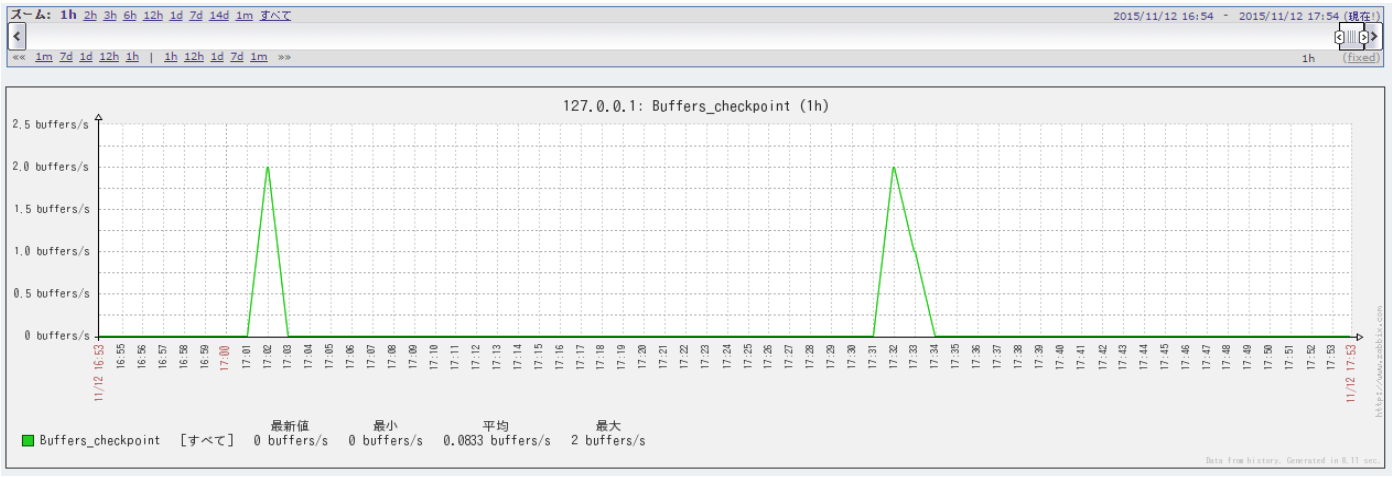


図 3.1:「最新データ」機能でのチェックポイントでの書き込み量

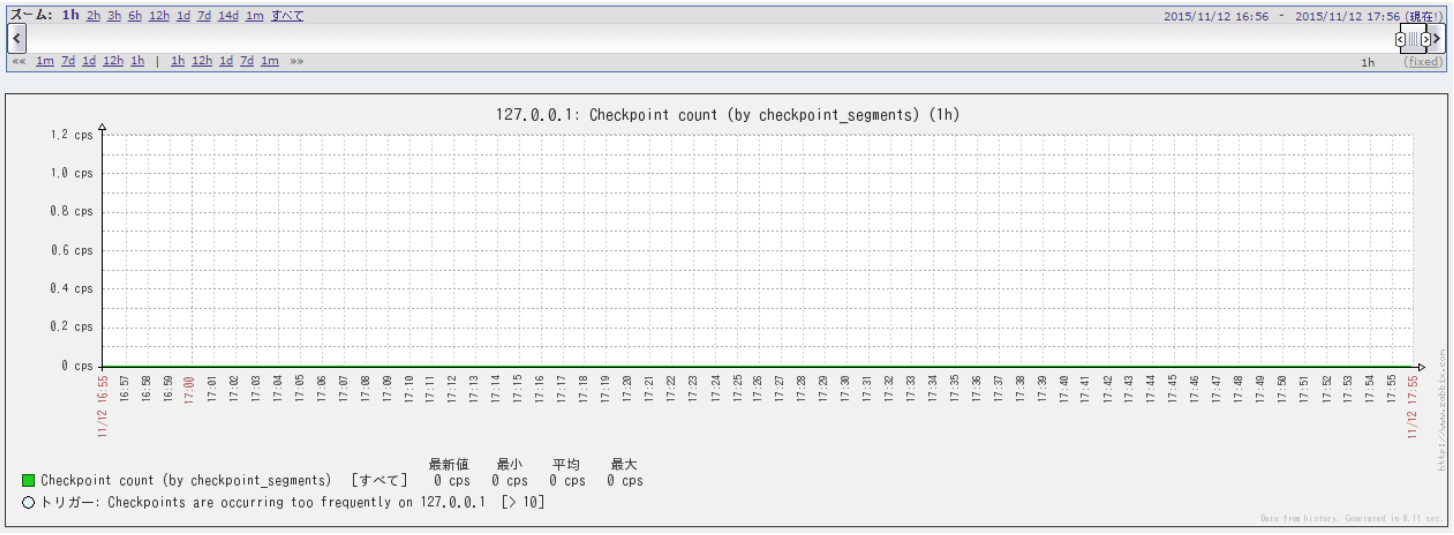


図 3.2:「最新データ」機能での WAL 書き込み量によるチェックポイント発生タイミング

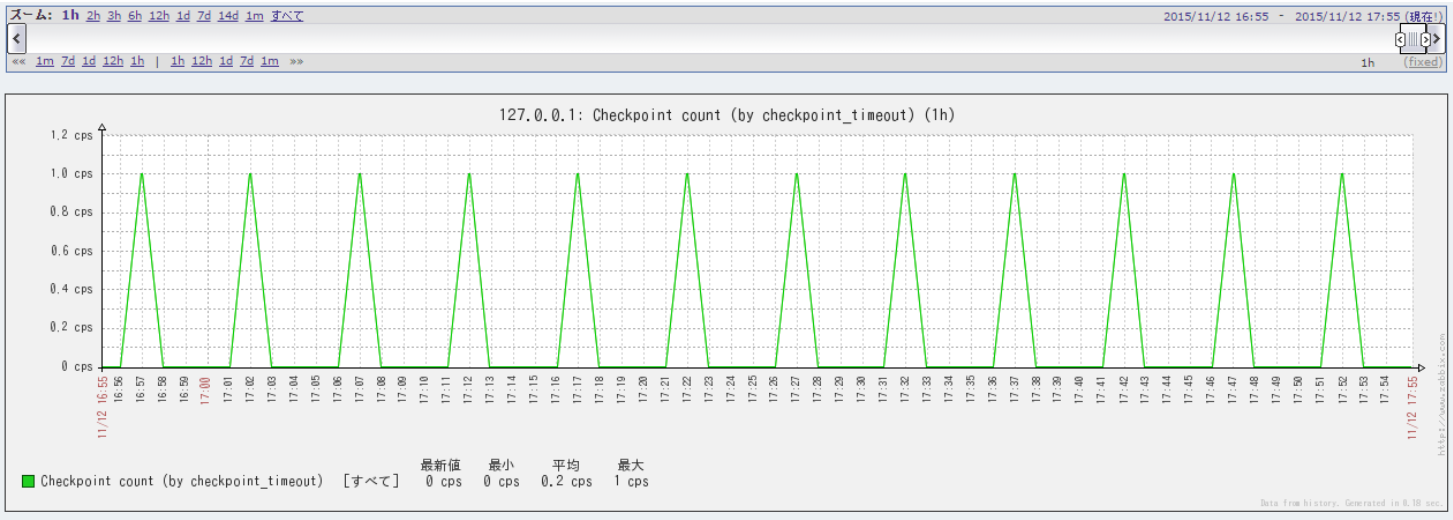


図 3.3:「最新データ」機能での時間経過によるチェックポイント発生タイミング

上記データのうちで特に、WAL 書き込み量によってチェックポイントが頻発しているときは、checkpoint_segments パラメータを、時間経過によってチェックポイントが頻発しているときは checkpoint_timeout を増やすとよいでしょう。

4. Autovacuum 関係のレポート

pg_monz は Autovacuum について、テーブル単位で以下の情報を取得しています。

- autovacuum 実行回数
- vacuum 対象行数
- vacuum 対象の比率

これらはそれぞれ、以下の箇所で確認可能です。

- 「監視データ」→「最新データ」→「pg.stat_table」→「[DB 名] (スキーマ名.テーブル名) autovacuum count」
- 「監視データ」→「最新データ」→「pg.stat_table」→「[DB 名] (スキーマ名.テーブル名) number of dead tuples」
- 「監視データ」→「最新データ」→「pg.stat_table」 →「[DB 名] (スキーマ名.テーブル名) Garbage ratio %」

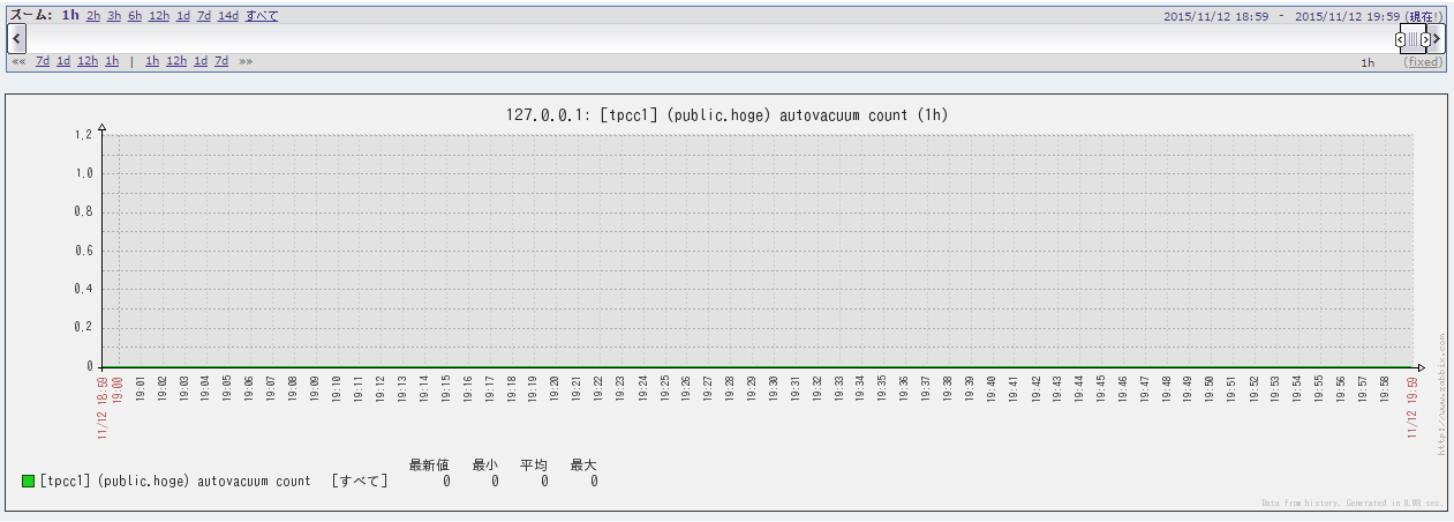


図 4.1:「最新データ」機能での autovacuum 実行回数

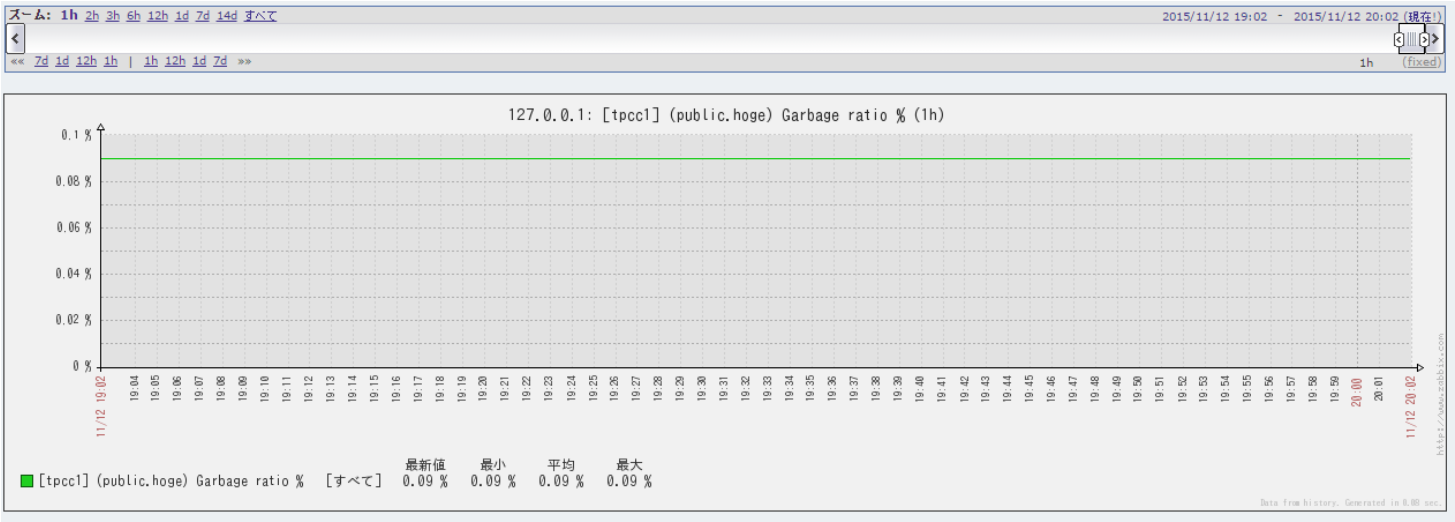


図 4.2:「最新データ」機能での autovacuum の対象行数の比率

また pg_monz は、autovacuum に関係のあるデータ複数を纏めた、以下のカスタムグラフも用意されています。

- 「監視データ」→「最新データ」→「pg.stat_table」→「[DB 名](スキーマ名.テーブル名) Table total size and garbage ratio」
- 「監視データ」→「最新データ」→「pg.stat_table」→「[DB 名](スキーマ名.テーブル名)vacuum and analyze activity」

こちらも使いやすいので、併せて利用するとよいでしょう。

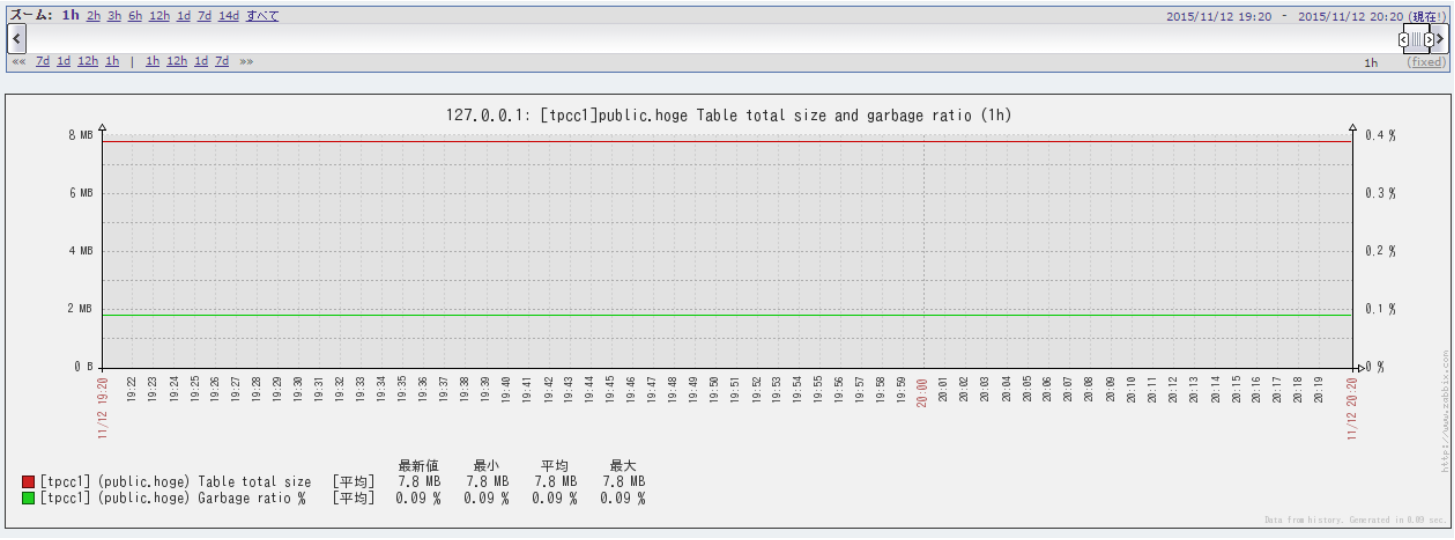


図 4.3:「カスタムグラフ」機能でのテーブルの容量と vacuum 対象タプルの比率

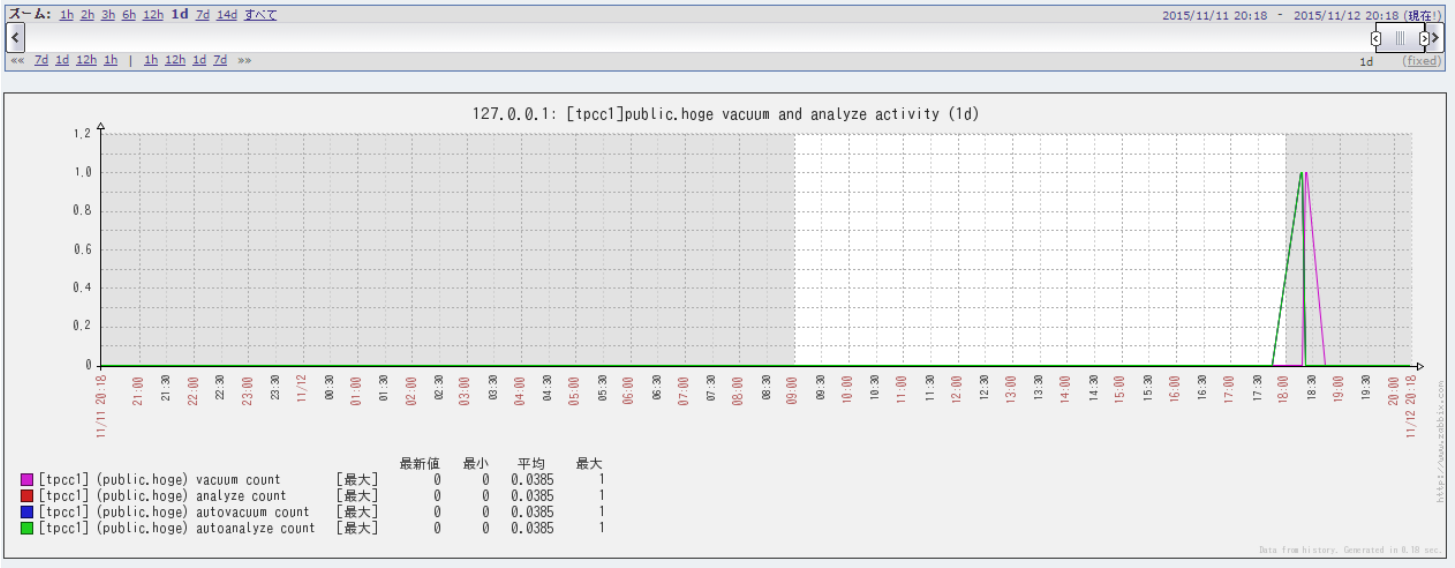


図 4.4:「カスタムグラフ」機能での vacuum と analyze の実行回数

以上 autovacuum について取得している情報量としては十分ではないかと思います。ただしこれもテーブル単位のグラフであり、監視するテーブルが多く、アイテムが非常に多くなった際は、オペレーションが難しくなる点には注意が必要です。

5. ディスク使用量 関係のレポート

テーブルスペースに対するディスク使用量は、Zabbix の本体が提供しているテンプレートで提供している、ディスク使用量の監視機能機能で代替可能です。そのため、pg_monz としてはこの機能を提供してません。

データベースに対する容量は以下で確認することが可能です。

- 「監視データ」→「最新データ」→「pg.size」-> 「[DB 名] DB Size」

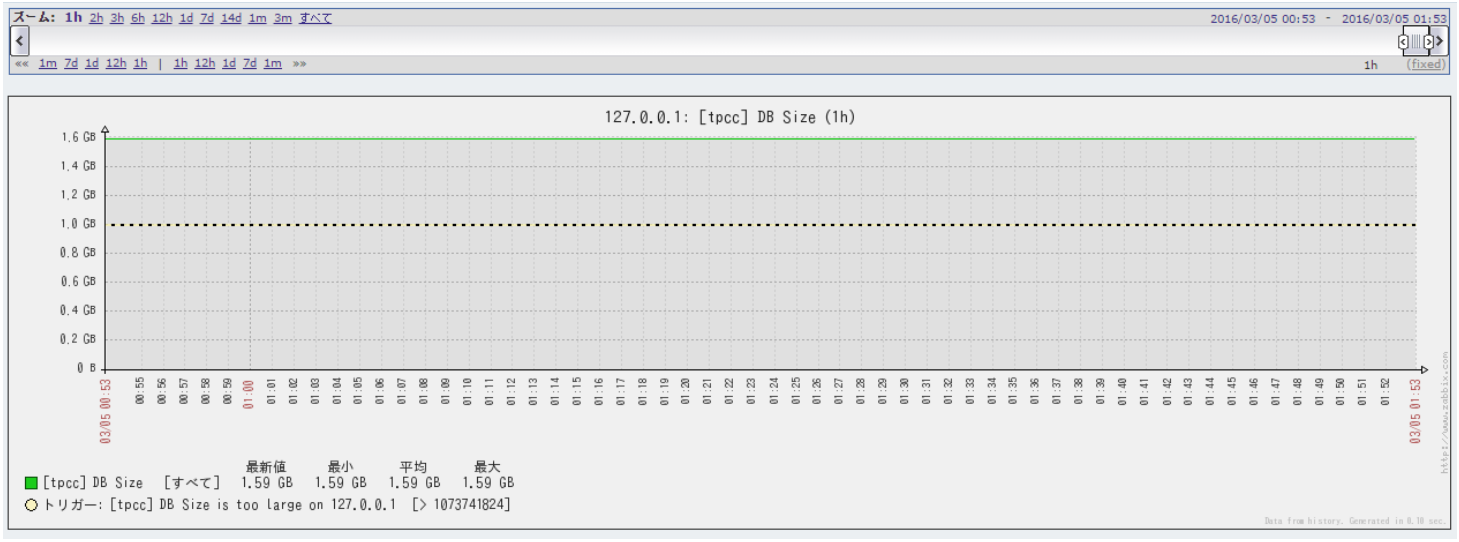


図 5.1:「最新データ」機能でのデータベースの容量

デフォルトでトリガーが設定されており、それをグラフ上の破線で確認ができます。ただしデフォルトの閾値が 1GB と小さく、環境によっては、絶えず警告状態になってしまうので、この値は調整したほうが良いかもしれません。

pg_monz はテーブルの容量データも取得しており、以下で確認することが可能です。

- 「監視データ」→「最新データ」→「pg.stat_table」-> 「[tpcc] (スキーマ名.テーブル名) Table total size」

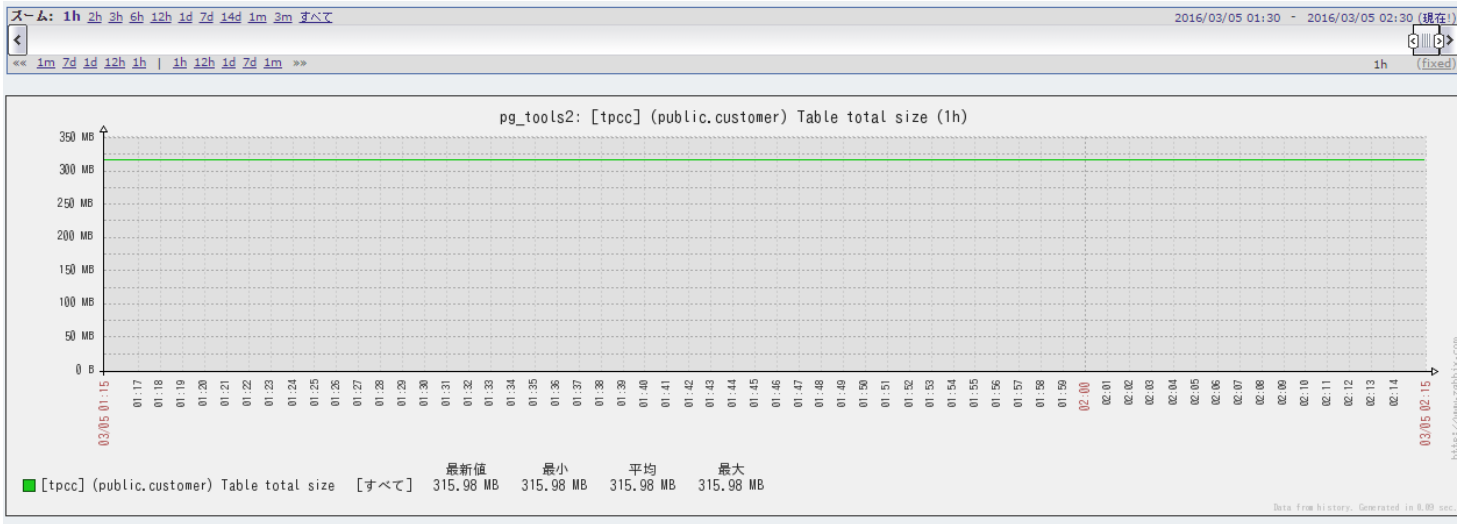


図 5.2:「最新データ」機能でのテーブルの容量

なお、この値は Table という名前ですが、実際はインデックスや TOAST などの領域も含んだ容量になりますので、ご注意ください。

6. インデックス使用 関係のレポート

pg_monz はインデックス単位の情報を取得していないので、原則としてインデックスの使用状況を確認することはできません。必要になった際は Zabbix のアイテムとして自分で追加する必要があります。

ただし、テーブル単位でインデックススキャン回数などの情報を取得していますので、インデックスが 1 つしかないテーブルに限っては、これらである程度代用することは可能です。

pg_statsinfo と pg_monz のディスク使用量検証

1. 目的

PostgreSQL の性能監視ツール「pg_statsinfo」と「pg_monz」に両者ついて、以下の知見を公開することです。

・両ツールのリポジトリデータベースに必要な大まかな容量
・両ツールのリポジトリデータの持ち方に関する特性

2. 概要

今回の調査内容及び結果の概要は以下となります。

1. サンプルのデータベースを、pg_statsinfo と pg_monz 両ツールにて監視をし、1 週間監視データを取得した
2. リポジトリデータベースのテーブルごとの必要容量を調査し、データ量の観点で重要なテーブル・インデックスを選定した
3. 選定したテーブルの 1 レコードあたりの必要容量を算出した
4. レコードあたりの必要容量とレコード数を決定するファクタで必要容量を見積もる 1 次の近似式を作成した
5. pg_statsinfo と pg_monz の両近似式を比較した結果、以下のことが分かった

・ 監視データを同期間の間保管した場合、pg_statsinfo の方が必要な容量は大きい
  - ただし、pg_monz はデフォルトで保存期間を長くとろうとするので、実際のデータ量はケースバイケース
・ 容量の差は、pg_statsinfo のみ取得している、「インデックス単位のデータ」「(テーブルの)列単位のデータ」の有無が大きい
  - 逆にインデックス数/列数が少ないと、インスタンス単位で取得するデータ量の差で両者の差は縮まる

特に例として近似式に具体的なパラメータを与えた際に必要なデータの見積もり量は以下となります。

表 2.1: リポジトリデータベースに必要なデータの見積もり量比較

インスタンス数	総データベース数 (※1)	総テーブル数 (※2)	平均インデックス数 (※3)	平均列数 (※3)	関数数	pg_statsinfo データ見積もり量(GB) (※4)	pg_monz データ見積もり量 (GB) (※5)
1	1	10	2	4	0	0.239	0.457
1	5	50	2	4	0	0.718	1.50
1	1	100	2	4	20	2.44	2.63
1	1	100	16	16	20	11.8	2.63
1	30	500	2	4	20	12.0	12.8
1	30	500	16	16	20	58.5	12.8

(※1) 監視対象の全インスタンスのデータベース
(※2) 監視対象の全データベースに存在する、全テーブルの総和
(※3) 監視対象のテーブル 1 つあたりの数
(※4) スナップショット取得間隔は 10 分(デフォルト値)、保管期間は 90 日 (pg_monz のヒストリデフォルト値に寄せた値)
(※5) DB/Table のアイテムの取得間隔は 60 分、ヒストリ保管期間は 90 日、トレンド保管期間は 365 日(全部デフォルト)

3. 方法

以下の環境に対して、JDBCRunner(http://hp.vector.co.jp/authors/VA052413/jdbcrunner/) で提供されている Tiny TPC-C テストキット用データをロード(scale factor 16) し、「Tiny TPC-C 用データベース」「pg_statsinfo リポジトリデータベース」の 2 つを 1 週間監視しました。

表 3.1: データ量の見積もり比較用環境

利用ソフトウェア	バージョンなど
OS	CentOS 6.7
PostgreSQL	9.4.5
pg_statsinfo/pg_monz 用リポジトリ DB	PostgreSQL 9.4.5
pg_statsinfo	3.1.0
Zabbix	2.4.6
pg_monz	2.0(※)

(※)今回は性能監視ツールという観点から pg_monz のテンプレート群のうちで、「Template App PostgreSQL」のみを対象としている。
また、デフォルトではデータベースやテーブルの性能情報取得が無効化されているが、今回はすべて有効化している。

その後、pg_statsinfo、pg_monz 両方のリポジトリデータベースにて、以下の SQL 文を実行し、pg_class.reltuples と pg_class.relpage の情報からデータベース内で容量を要するテーブルの選定及び、レコード辺りのデータ量の算出を行いました。

pg_statsinfo のデータ量調査

```
statsinfo=# select
statsinfo-#   relname,
statsinfo-#   relkind,
statsinfo-#   reltuples,
statsinfo-#   relpages,
statsinfo-#   relpages * 8192 / 1024 "(KB)",
statsinfo-#   case
statsinfo-#     when reltuples = 0 then null
statsinfo-#     else relpages * 8192 / reltuples
statsinfo-#   end "byte/row",
statsinfo-#   (sum(relpages::real) over (order by relpages desc) / sum(relpages) over ())::numeric(3,2) "relpages の
累積比率"
statsinfo-# from
statsinfo-#   pg_class
statsinfo-# where
statsinfo-#   relnamespace = 16410
statsinfo-# order by relpages desc
statsinfo-# ;
```

pg_monz のデータ量調査

```
zabbix=# select
zabbix-#   relname,
zabbix-#   reltuples,
zabbix-#   relpages * 8192 / 1024 "(KB)",
zabbix-#   relpages * 8192 / reltuples "byte/row",
zabbix-#   relkind
zabbix-# from
zabbix-#   pg_class
zabbix-# where
zabbix-#   relname like '%history_uint%' or
zabbix-#   relname in ('history', 'history_1') or
zabbix-#   relname like '%trends%'
zabbix-# ;

※
pg_monz については、Zabbix マニュアルに、データ見積もりに重要なデータは「History」「Trends」「Events」の 3 つであると書かれている点、及び Events は pg_monz 内で発報関係の扱いが難しい点から、予め「Histroy」と「Trends」関係のテーブルに絞っています。
```

上記 SQL 文で得られる結果から、1 レコードあたりのデータ量を取得できます。一方でレコード数について、

pg_statsinfo は、取得対象オブジェクトの数とスナップショット間隔、及びレコード保存期間で算出できます。一方で pg_monz は取得アイテム数ごとにレコード記録間隔を変えることが可能であり、なおかつデフォルトの時点それぞれ間隔が異なります。そこで、今回はデフォルトのレコード記録間隔を採用したと仮定して、以下の SQL 文でデフォルト記録間隔を調査しました。

pg_monz のデフォルトレコード記録間隔の分布調査

```
zabbix=# select
zabbix-#   i_temp.data_type,
zabbix-#   i_temp.item_unit,
zabbix-#   60 * 60 * 24 / i_temp.delay "history data per day",
zabbix-#   i_temp.history "history keep days",
zabbix-#   24 "trends data per day",
zabbix-#   i_temp.trends  "trends keep days",
zabbix-#   count(*)
zabbix-# from (
zabbix(#   select
zabbix(#     case
zabbix(#       when i.value_type = 0 then 'float'
zabbix(#       when i.value_type = 2 then 'log'
zabbix(#       when i.value_type = 3 then 'int'
zabbix(#       when i.value_type = 4 then 'discovery'
zabbix(#     end data_type,
zabbix(#     case
zabbix(#       when i.delay <> 0 then i.delay
zabbix(#       else case
zabbix(#         when a.name = 'pg.bgwriter' then 60
zabbix(#         when a.name in ('pg.stat_database', 'pg.stat_table', 'pg.size') then 3600
zabbix(#         else 300
zabbix(#       end
zabbix(#     end delay,
zabbix(#     case
zabbix(#       when parent_i.key_ like 'db_table.list%' then 'table'
zabbix(#       when parent_i.key_ like 'db.list.%' then 'database'
zabbix(#       else 'database cluster'
zabbix(#     end item_unit,
zabbix(#     i.history,
zabbix(#     i.trends
zabbix(#   from
zabbix(#     items i
zabbix(#   inner join
zabbix(#     items_applications ia
zabbix(#       on i.itemid = ia.itemid
zabbix(#   inner join
zabbix(#     applications a
zabbix(#       on ia.applicationid = a.applicationid
zabbix(#   inner join
zabbix(#     hosts h
zabbix(#       on i.hostid = h.hostid and
zabbix(#          h.host = 'Template App PostgreSQL'
zabbix(#   left outer join
zabbix(#     item_discovery id
zabbix(#       on i.itemid = id.itemid
zabbix(#   left outer join
zabbix(#     items parent_i
zabbix(#       on id.parent_itemid = parent_i.itemid
zabbix(#   ) i_temp
zabbix-# where i_temp.data_type in ('int', 'float')
zabbix-# group by i_temp.data_type, i_temp.delay, i_temp.item_unit, i_temp.history, i_temp.trends
zabbix-# order by i_temp.data_type, i_temp.item_unit, i_temp.delay
zabbix-# ;
```

最後に、今回 pg_statsinfo と pg_monz のリポジトリデータベースから選別したテーブル(とそれに紐付くインデックス)に必要な容量は、保管しているレコード数に依存し、なおかつ正比例すると仮定して、近似式を算出しました。この近似式は具体的には以下となります。

$$\sum_t B_t R_t$$

t : 今回選別したテーブル
B_t : テーブル t の 1 レコードあたりの平均データ量
R_t : テーブル t のレコード数

4. 結果

pg_statsinfo リポジトリデータベースのデータ量を調査した結果、以下の様になりました。

表 4.1: pg_statsinfo リポジトリデータベースのテーブルごとのデータ量

テーブル名	レコード数 (tuples)	総データ量(MB) (※1)	レコード単位のデータ量 (bytes/tuples)	データ量の累積比率(%)
column	461358	90.2	205	73.3
function	26623	10.2	402	81.6
table	28528	9.47	348	89.3
index	24390	8.41	362	96.1
database	1348	0.555	432	96.6
その他	-	4.21	-	100

(※1)「総データ量」はテーブルとそれに紐付くインデックス、パーティショニングの子テーブルすべての総和

今回の環境では、column, function, table, index の 4 テーブルで 96.1%のデータ量になりました。これは十分な比率だと判断して、近似式の算出ではこの 4 テーブルと database テーブルの計 5 テーブルで近似式を求めることにしました。なお、今回データ量の小さい database テーブルを追加したのは、pg_monz にとってデータベース数が重要なファクタであり、その比較のためです。

なお、pg_statsinfo の上記テーブルは、スナップショット取得タイミングで対象の全オブジェクト分のレコードが追加されます。そのため、レコード数は「対象オブジェクト数」と「データ保存期間」と「スナップショット間隔(の逆数)」の積で見積もることが可能です。

一方で pg_monz リポジトリデータベースの主要テーブルに関するデータ量を調査した結果、以下の様になりました。

表 4.2: pg_monz リポジトリデータベースの主要テーブルのテーブルごとのデータ量

テーブル名	テーブル説明	レコード数 (tuples)	総データ量(MB) (※1)	レコード単位のデータ量 (bytes/tuples)
history	浮動小数型のヒストリ(※2)	5412	0.797	154
history_uint	整数型のヒストリ(※2)	188057	20.1	112
trends	浮動小数型のトレンド(※3)	5678	0.883	163
trends_uint	整数型のトレンド(※3)	32404	3.56	115

(※1) 「総データ量」はテーブルとそれに紐付くインデックスの総和
(※2) 「ヒストリ」とは、Zabbix が短期で保存するための詳細データのこと
(※3)「トレンド」とは、Zabbix が長期で保存するために「ヒストリ」1 時間分を集計したサマリデータのこと

また、pg_monz はアイテムごとにレコード記録頻度が異なります。レコード数算出に必要な、このデフォルトのレコード記録間隔の分布を調査した結果、以下の様になりました。

表 4.3: pg_monz のデフォルトレコード記録間隔の分布と1日単位の取得レコード数

テーブル名	データ保管日数	1日ごとの記録回数	記録1回ごとのレコード数	1日ごとの総レコード数
history	90	24	2×[データベース数] + 3×[テーブル数]	48×[データベース数] + 72×[テーブル数]
history_unit	90	24	13×[データベース数] + 16×[テーブル数]	312×[データベース数] + 384×[テーブル数] + 17856×[データベースクラスタ数]
		288	17×[データベースクラスタ数]	
		1440	9×[データベースクラスタ数]	
trends	365	24	2×[データベース数] + 3×[テーブル数]	48×[データベース数] + 72×[テーブル数]
trends_uint	365	24	13×[データベース数] + 16×[テーブル数] + 26×[データベースクラスタ数]	312×[データベース数] + 384×[テーブル数] + 624×[データベースクラスタ数]

ここまで調査した、「1レコードあたりのデータ量」と「レコード量」の両者を元に、総データ量の近似式を算出しました。
pg_statsinfo の総データ量は以下の式で見積もることができます。

$$\left(0.593D+\left(0.335+0.348I+0.197c\right)T+0.552F\right)*k/s \quad \text{(MB)}$$

D: 総データベース数
I : 1テーブルあたりの平均インデックス数
c : 1テーブルあたりの平均カラム数
T: 総テーブル数
F: 総ユーザ定義関数数
k: データ保管日数 (day)
s: スナップショット取得間隔 (min)

一方で pg_monz は以下の式で総データ量を見積もることができます。

$$\left(196.7C+18.88D+24.15T\right) \quad \text{(MB)}$$

C: データベースクラスタ数
D: 総データベース数
T: 総テーブル数
※データ保管日数とデータ取得はデフォルト値を採用した場合

両近似式のファクタに両ツールのデータの取得方針がある程度表れています。特にポイントとなるのは以下の2点です。

1. pg_statsinfo のみ、インデックス単位、表の列単位、ユーザ定義関数単位のデータを取得している
   - pg_statsinfo のみ上記3つのファクタが式に存在しています。

2. pg_monz はアイテム単位でレコード記録頻度や保存期間を調節できる
   - pg_monz デフォルトでは、データベースクラスタ単位で取得するデータのみ高頻度で取得しています。
     その結果、データベースクラスタ単位の影響を無視できず、pg_monz 近似式でこのファクタを残しました。
   - アイテムごとに記録頻度や保存期間を調整できることを簡易な式に書き下すことは困難です。
     その結果、pg_monz の近似式ではデフォルト固定としました。その結果このファクタが消えました。

上記近似式は必要の容量の規模が直感的には分からないので、以下で具体的なパラメータを与えたうえでのデータ量の

変遷を描写しました。

pg_statsinfo については、以下の様な規模のデータベースの情報を取得する時、数 100MB～数 GB のデータ用の領域が必要になりそうです

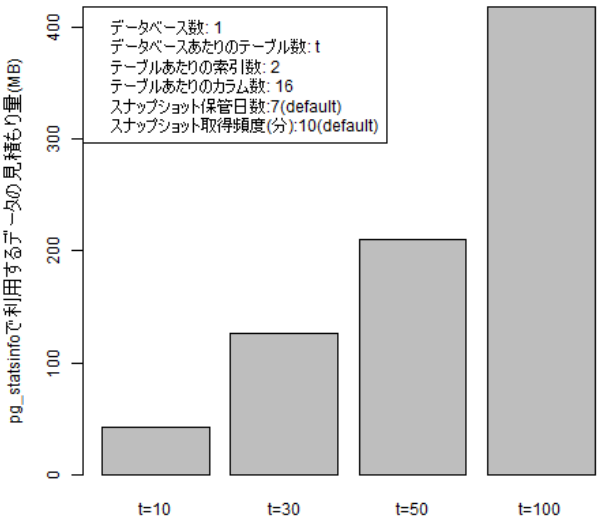


図:テーブル数の変化に伴うpg_statsinfoデータ量の変遷

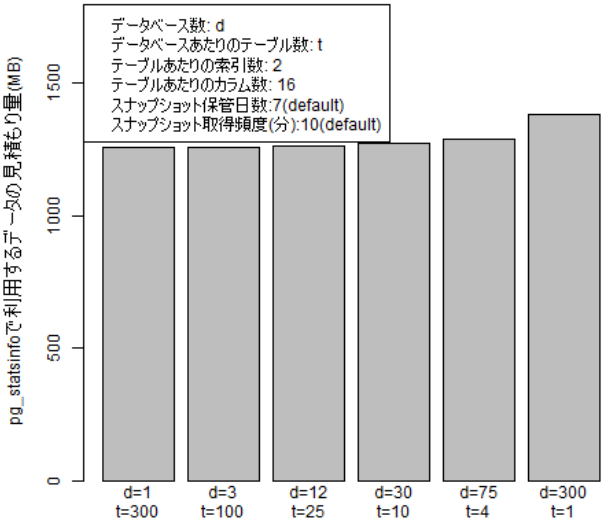


図:データベース数の変化に伴うpg_statsinfoデータ量の変遷

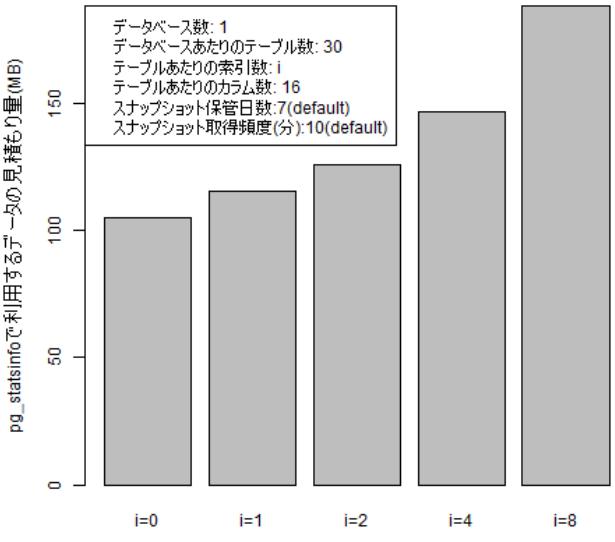


図:テーブルあたりの索引数の変化に伴うpg_statsinfoデータ量の変遷

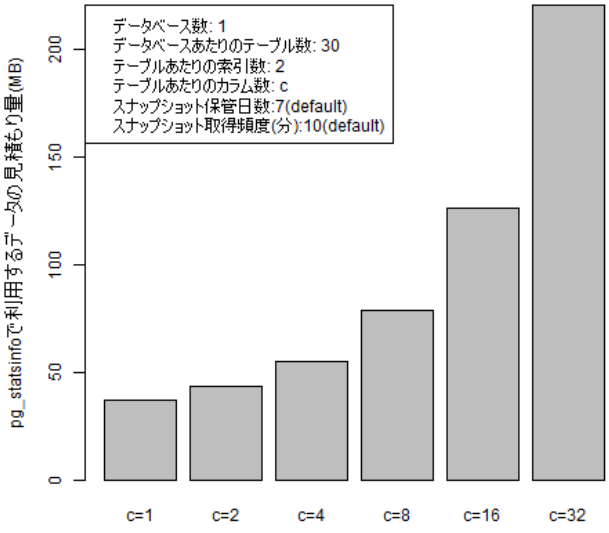


図:テーブルあたりのカラム数の変化に伴うpg_statsinfoデータ量の変遷

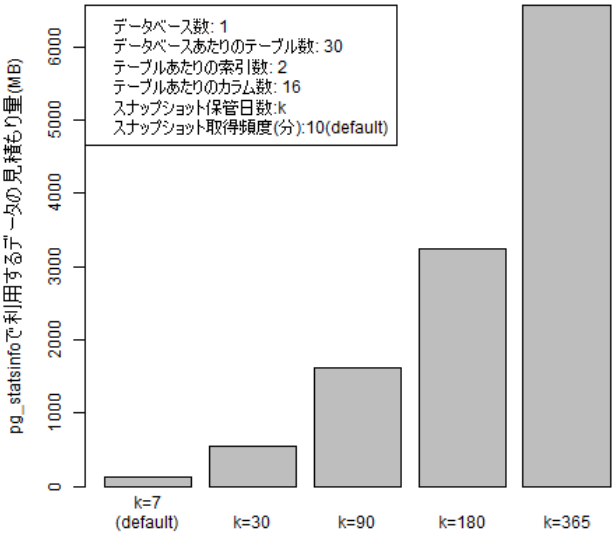


図:スナップショット保管日数の変化に伴うpg_statsinfoデータ量の変遷

一方、pg_monz については、以下の様な規模のデータベースの情報を取得する時、数 GB～数 10GB のデータ用の領域が必要になりそうです

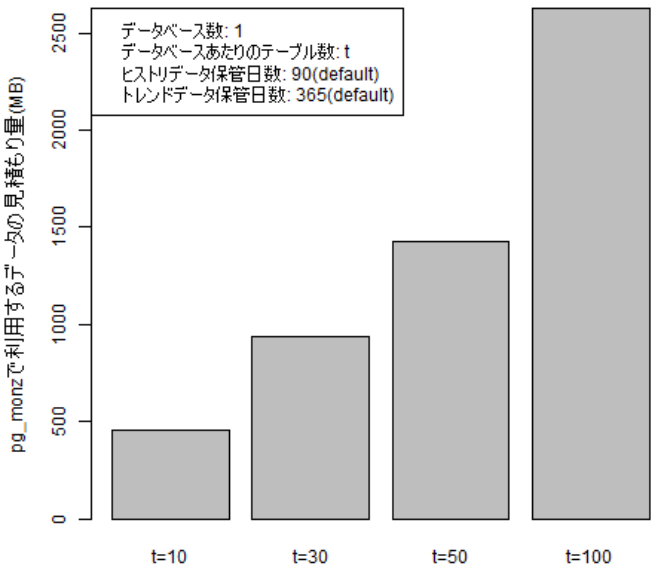


図:テーブル数の変化に伴うpg_monzデータ量の変遷

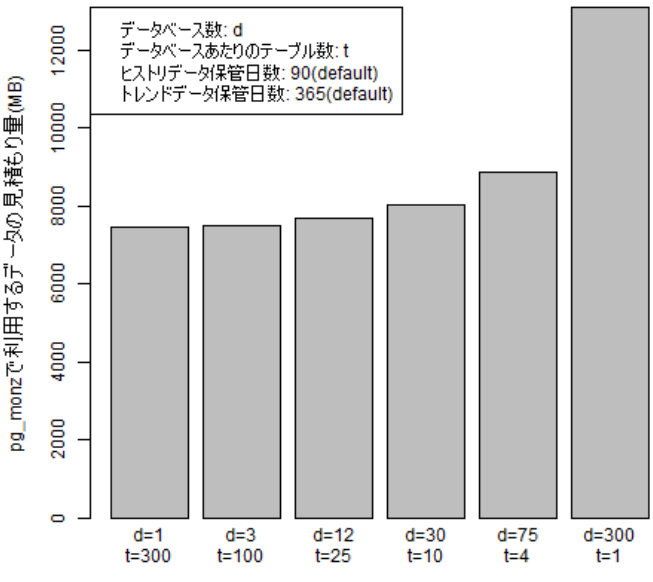


図:データベース数の変化に伴うpg_monzデータ量の変遷

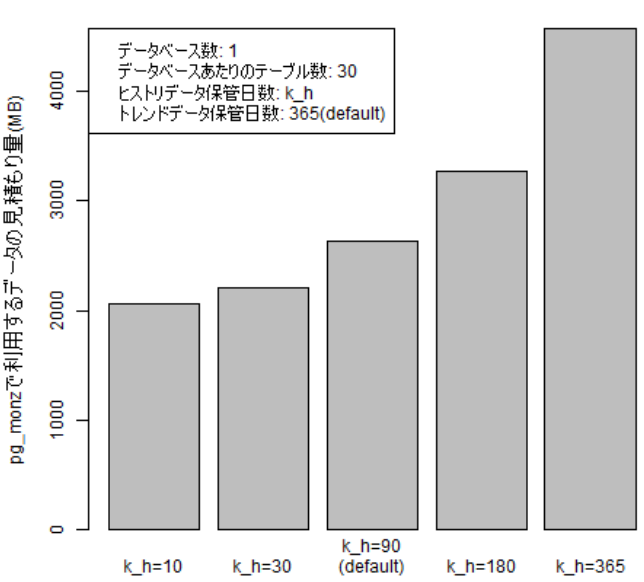


図:ヒストリデータ保管日数の変化に伴うpg_monzデータ量の変遷

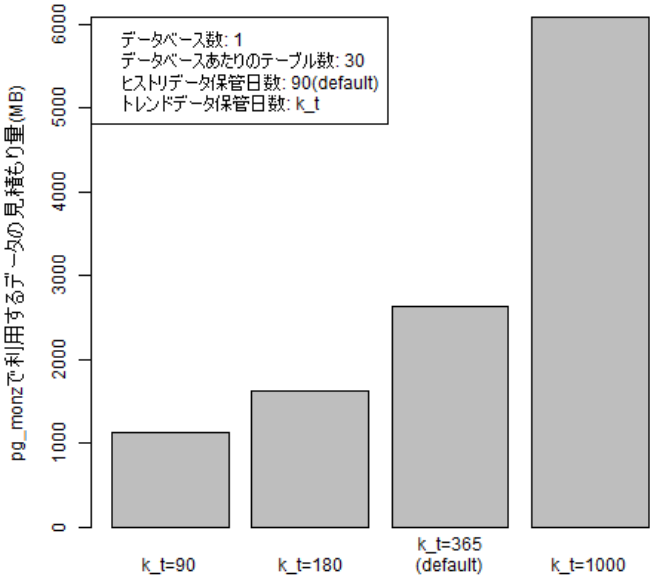


図:トレンドデータ保管日数の変化に伴うpg_monzデータ量の変遷

ここまで pg_statsinfo と pg_monz では、前者の方が取得している情報が多く、結果として必要なディスクの容量も多いと書いてきました。しかし、前者の方がデフォルトのデータ保存期間がかなり短く、意識せずに導入すると、上記グラフのように、pg_monz の方が必要な容量が多くなることもあるでしょう。両者の保存期間とデータ量の関係は以下の様に比較できます。

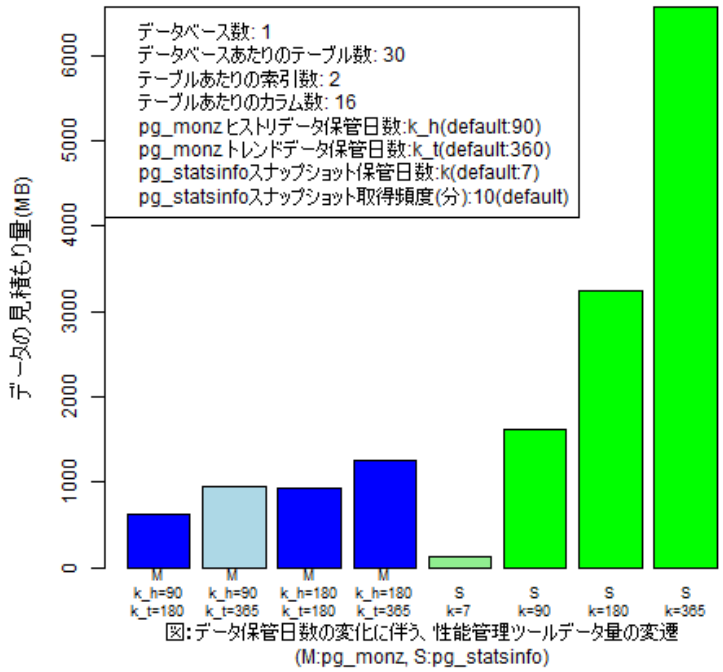


図:データ保管日数の変化に伴う、性能管理ツールデータ量の変遷
(M:pg_monz, S:pg_statsinfo)

5. 補足

今回の検証では、幾つかの項目の検討がされていません。お使いの環境にとってこの項目が重要な場合、今回の見積もりデータ量と実際のデータ量の間に大きな隔たりが生じることがあり得ます。今回の検証で重要ではないかと推測していたものの、十分な検証をすることができなかった項目を以下に挙げます。

5.1. ログ

pg_statsinfo と pg_monz の両者、PostgreSQL のログを監視する機能を有しています。ログで出力する情報は環境に大きく依存するため、今回の検証では検討項目から除外しました。しかし、ログデータは膨大な量になる可能性があります。

もしもお使いの環境でログに監視対象となる文言が大量に出力される場合は、別途ログデータ量の見積もりをしたり、取得するログデータを絞ることを検討するとよいでしょう。ログデータのフィルタリングについては、pg_statsinfo はメッセージレベルやユーザで可能なので細かな調整が可能です。pg_monz については、明示的にフィルタリング専用の機能があるわけではありませんが、実装が Zabbix 標準のログ取得機構を使ったものなので、Zabbix 上の設定変更をすることで比較的容易に実施可能です。

5.2. 運用期間

監視ツールのリポジトリ DB は、ハウスキープ機能が正しく働いてさえいれば、レコード数は必ず定常値に落ち着きます。これは、データの追加間隔が一定だという特性によるものです。しかし、レコード数が一定でバキュームが正しく実施されていたとしても、運用期間が長くなると、少しずつ必要なディスク容量は増えていきます。今回は検証期間の関係上、1 週間と短い期間後の状態でしか検証できていません。長期の運用の際は定期的にリポジトリ DB 自身の容量も確認するようにしたほうが良いでしょう。

pg_statsinfo はデータ量の大きいテーブルについてはパーティショニングが実装されており、古い子テーブルを drop するメンテナンスになっているため、この影響は比較的小さいことが予想されます。しかし、pg_monz 上は Zabbix 上でそのような工夫がされているわけではないので、特に注意が必要でしょう。

pg_monz に対する WebAPI 利用

1. pg_monz における WebAPI について

pg_monz は WebAPI 機能によって、リモートのプログラムで監視情報の取得ができたり、各種設定変更が実施できます。これは、pg_monz が監視ツール Zabbix 上のテンプレートとして実装されているためです。pg_monz が Zabbix 上で動いているため、Zabbix が提供する機能をそのまま活用できます。WebAPI は Zabbix が提供する便利な機能の1つです。

2. 簡単な使い方

WebAPI の簡単な使い方のチュートリアルとして、「zabbix のユーザ認証を実施」して「管理しているホスト一覧を取得する」処理を linux の curl コマンドを使って行う例を挙げます。

Zabbix の WebAPI は /zabbix/api_jsonrpc.php へ JSON データを Post することで利用可能です。Zabbix のユーザにパスワード設定をして運用している場合では、まずユーザ認証が必要になります。ユーザ認証機能は JSON の method に'user.login'を指定することで使うことができます。

Zabbix のユーザ認証 WebAPI 利用例

```
$ curl -H "Content-Type:application/json-rpc" -d '
{
  "auth":null,
  "id":1,
  "jsonrpc":"2.0",
  "method":"user.login",
  "params":{
    "password":"zabbix",
    "user":"Admin"
  }
}
' http://my-zabbix-server.jp/zabbix/api_jsonrpc.php

{"jsonrpc":"2.0","result":"d987532e03e7748f262b3c7c3eecd6fc","id":1}
```

上記のように、sessionid を取得することができます。今後はこれを auth に設定することで、特定ユーザの権限が必要な機能も使うことができるようになります。ここでは、method に 'host.get' を指定して、ホスト一覧取得機能を利用してみます。

Zabbix のホスト一覧取得 WebAPI 利用例

```
$ curl -H "Content-Type:application/json-rpc" -d '
{
  "auth":"d987532e03e7748f262b3c7c3eecd6fc",
  "id":1,
  "jsonrpc":"2.0",
  "method":"host.get",
  "params":{
    "output": ["hostid", "host"],
    "filter": {
      "host": "127.0.0.1"
    }
  }
}' http://my-zabbix-server.jp/zabbix/api_jsonrpc.php | jq .

{
  "jsonrpc": "2.0",
  "result": [
    {
      "hostid": "10110",
      "host": "127.0.0.1"
    }
  ],
```

```
  "id": 1
```

上記のように、WebAPIを活用して pg_monz で管理しているホスト一覧を取得できました。

これは非常に簡単な利用例ですが、他の WebAPI 機能やより詳細な使い方を知りたい場合は、Zabbix のマニュアルにをご確認ください。
https://www.zabbix.com/documentation/2.4/manual/api

3. 応用例（テーブルキャッシュヒット率のランキング機能の実装）

WebAPI 機能を使うことで、プログラムから pg_monz で取得したデータを活用したり、Zabbix のコントロールができたりします。ここでは簡単な応用として、pg_monz のデータを取得してキャッシュヒット率のランキング機能を実装してみます。

pg_monz は問題個所の調査が若干やり難いです。閾値を設定しての通知機能(トリガー)がありますが、トリガーはその性質上「通知必要状態/通知不要な状態」の2値しか取れません。複数のトリガーを組み合わせることもできますが、それでもこれだけで問題になりそうな項目を洗い出すのはやや苦しいです。特に監視対象データベースにテーブルやインデックス数が多い場合、登録アイテムの数が膨大になり重要な情報が埋もれやすく、性能分析がし難くなります。

性能問題の分析における有力なツールとしてランキング機能があります。しかし、ランキング機能はアーキテクチャ上、Zabbix の機能としては実装が難しいです。そこで WebAPI で取得できるデータを活用して、Zabbix の外部から実行するプログラムとして実装してみます。

WebAPIを利用したテーブルキャッシュヒット率のランキング機能の実装

```
#! /usr/bin/env ruby
# coding: utf-8

require "net/http"
require "json"

#
# Zabbix API を叩くためのクラス
#
# 説明の為に以下の処理を簡略化している点に注意
#
#  ・ 情報の取得(get)しかできない
#    - Zabbix API は情報取得以外も可能
#  ・ ネットワーク関係
#  ・ 認証(ユーザ/パスワード) 関係
#  ・ その他エラー処理など諸々
#
class ZabbixAPI
  #
  # コンストラクタ
  #
  # 認証(user.login)だけここで済ます。
  # sessionid を記録して、get メソッド時に使いまわす
  #
  def initialize(uri, user, password)
    @uri = URI.parse(uri)
    @http = Net::HTTP.new(@uri.host, @uri.port)

    auth_params = {
      auth: nil,
      id: 1,
      jsonrpc: '2.0',
      method:  'user.login',
      params:{
        password: password,
        user: user
```

```
      }
    }

    res = @http.post(
      @uri.request_uri,
      auth_params.to_json,
      { "Content-Type" => "application/json-rpc" }
    )

    json = JSON.parse(res.body)
    @auth = json['result']
  end

  #
  # 情報の取得メソッド
  #
  # 引数の prams を使って、
  # type.get することで情報を取得する
  #
  def get(type, params)
    data = {
      auth: @auth,
      id: 1,
      jsonrpc: '2.0',
      method: "#{type}.get",
      params: params
    }

    res = @http.post(
      @uri.request_uri,
      data.to_json,
      { "Content-Type" => "application/json-rpc" }
    )
    json = JSON.parse(res.body)
  end
end

# Zabbix API を叩くためのインスタンス
api = ZabbixAPI.new('http://my-zabbix-server.jp/zabbix/api_jsonrpc.php', 'Admin', 'zabbix')

################################################################
#
# 全テーブルのキャッシュヒット率のメタデータ(アイテム)を取得するパラメータ
#
params = {
  output: ['itemid', 'name'],
  searchWildcardsEnabled: true,
  sortfield: 'itemid',
  sortorder: 'ASC',
  filter: {
    host: '127.0.0.1'
  },
  search: {
    name: '*heap cache hit ratio %*'
  }
}

# Zabbix の API を叩いて以下のデータ構造を取得する
#
# [
#   {'itemid' => 1, 'name' => '[statsinfo] (statsrepo.log_20151117) heap cache hit ratio %'},
#   {'itemid' => 2, 'name' =>
#   ...
# ]
```

```
items = api.get('item', params)['result']

#
# Hash を使いやすいように変形する
#
# [
#   {'1' => '[statsinfo] (statsrepo.log_20151117) heap cache hit ratio %'},
#   {'2' => '[statsinfo]...
# ]
#
item_hash = {}
items.each { |h| item_hash[h['itemid']] = h['name'] }


############################################################
#
# 記録しているキャッシュヒット率データ(history)のうち、
# 2015/11/15 11:00:00 -12:00 の間に記録されたものを
# 取得するためのパラメータ
# 実際には引数などで指定できるようにするべきだが、
# 本質ではないので、分かりやすさのためここでは省略
#

target_time = Time.new(2015, 11, 15, 11, 0, 0) # 2015/11/15 11:00:00

params = {
  output: 'extend',
  history: 0,                                # キャッシュヒット率はfloatなので, 0を指定
  itemids: item_hash.keys,                   #  上で取得した全キャッシュヒット率のアイテムID
  time_from: target_time.to_i,               # 2015/11/15 11:00:00
  time_till: (target_time + 60 * 60).to_i, # 2015/11/15 12:00:00
  filter: {
    host: '127.0.0.1',
  }
}

# Zabbix のAPI を叩いて以下のデータ構造を取得する
#
#  [
#    {"itemid": "1", "clock": "1447648524", "value": "100.0000", "ns": "0"},
#    {"itemid": "1", "clock": "1447652124", "value": "100.0000", "ns": "0"},
#    ...
#  ]

histories = api.get('history', params)['result']

# キャッシュヒット率(value) でソートして、先頭の10個を取得することで、
# キャッシュヒット率ワースト10位を記録する
worst_hist = histories.sort_by {|hash| hash['value'].to_i}[0..9]

# 'history' では itemid しか取れないので、
# 人間にも読めるように、アイテム名を合わせてHashに詰める
worst_tables = []
worst_hist.each do |hist|
  worst_tables << {
    'itemid'   => hist['itemid'],
    'value'    => hist['value'],
    'get_time' => Time.at(hist['clock'].to_i),
    'name'     => item_hash[hist['itemid']]
  }
end

# キャッシュヒット率ワーストをJSONで出力する
puts worst_tables.to_json
```

上記のスクリプトを実行すると、以下のような結果を得ることができます。

pg_statsinfo のデータ量調査

```
$ ./hoge.rb | jq .

[
 {
  "itemid": "26300",
  "value": "53.0000",
  "get_time": "2015-11-15 11:35:42 +0900",
  "name": "[tpcc1] (public.warehouse) heap cache hit ratio %"
 },
 {
  "itemid": "26311",
  "value": "63.0000",
  "get_time": "2015-11-15 11:35:52 +0900",
  "name": "[tpcc_large] (public.customer) heap cache hit ratio %"
 },
 {
  "itemid": "26309",
  "value": "73.0000",
  "get_time": "2015-11-15 11:35:50 +0900",
  "name": "[tpcc_large] (public.warehouse) heap cache hit ratio %"
 },
 ...
```

上記のように、キャッシュヒット率の低いテーブルを順番に確認することができました。

ただし、Zabbix の WebAPI の注意点として、テーブルの結合にあたる処理が実施できません。スクリプト内ではアイテム(監視の内容)の取得とヒストリ(監視の実データ)の取得を一括取得ができないために、それぞれ個別に取得して、自前で SQL 文の JOIN にあたる処理をしています。このことから、WebAPI 機能はあまりに複雑な分析には不向きであるともいえるでしょう。

Zabbix の WebAPI 機能は pg_monz に大きな拡張性を提供します。癖はあるものの、上手く活用すれば日々の DBA 業務を効率化したり、より大きなプログラムと pg_monz の連携ができたりするかもしれません。