性能トラブル班
成果紹介

性能トラブル班の活動目的

- テーマ選択にあたっての背景
  - PostgreSQLの普及/利用者拡大に伴い、
    様々なケースの性能トラブルが発生
  - 性能トラブルを未然に防ぐ方法や早期に検知するための
    情報を整理することをWG3で取り組むべき課題として選択

- 目的
  - 性能トラブルを未然に防ぐためのノウハウや
    トラブルを検知するための情報を整理する
  - PostgreSQLのエンタープライズ領域への普及にあたって、
    性能トラブル対策の一助となる情報を提供する

成果物について

- 参画企業が直面した性能トラブルの事例と原因に関する情報を共有し、以下の観点で分析・議論。

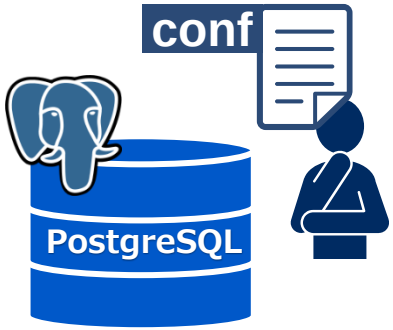
- 性能トラブルを防ぐために、
  - 設計段階での**予防**
  - トラブルの早期**検知**
  - いざ発生した時の**対処**

  の3つの観点で調査/整理したノウハウを紹介

成果物の構成

予防

4章 性能トラブルを予防するために考慮すべき
データベース設計のポイント



検知

5章 性能状態を把握するための監視

対処

6章 ケーススタディ

- 性能トラブル発生から解決までのサンプル事例8件
  - 「複合インデックス追加のみでは性能が出ない場合の対処方法」をケーススタディとして追加

本日説明予定

対処

ケーススタディについて

- トラブルの検知から解決までの方法を事例として紹介
  1. トラブルの内容
  2. 調査と分析の進め方
  3. 改善手法の紹介

- 本年度追記分の事例一つを紹介

対処

複合インデックス追加のみで性能が出ない場合の対処方法-①

■ テーブル、インデックス、SQLの概要

□ テーブル、インデックスの概要

OSS監視ツールのZabbixが収集するイベントを保存するテーブル。

eventid	source	object	objectid	clock	・・・・・
4020491	0	0	16821	1562752071	・・・・・
4018669	0	0	16821	1562751861	・・・・・

Primary keyは、eventid

source, object, objectid, clockに複合インデックス

source, object, clockに複合インデックス

eventid	r_eventid	c_eventid	・・・・・
4020491	4020946	4020945	・・・・・
4018669	4018945	4018946	・・・・・

Primary keyは、eventid

r_eventid、 c_eventid各々にインデックス

対処

複合インデックス追加のみで性能が出ない場合の対処方法-②

- テーブル、インデックス、SQLの概要

  - 検索SQLの概要

eventsテーブルとevent_recoveryテーブルの外部結合、WHERE句の条件で結合結果を絞り込む。

検索結果は、Zabbixの監視画面の表示に使用される。

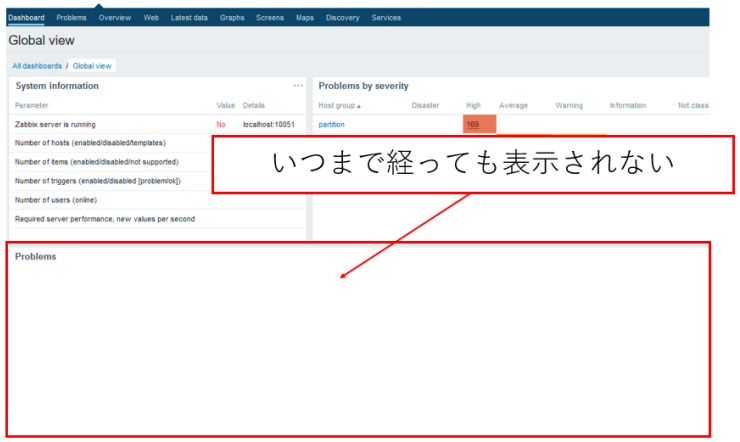
```
SELECT DISTINCT e.eventid,e.clock,e.ns,e.objectid,e.acknowledged,er1.r_eventid
FROM events e LEFT JOIN event_recovery er1 ON er1.eventid=e.eventid
WHERE e.source='0' AND e.object='0' AND e.objectid='16821' AND e.eventid<='4020491' AND e.value='1'
ORDER BY e.eventid DESC
LIMIT 20
;
```

対処

複合インデックス追加のみで性能が出ない場合の対処方法-③

■ トラブルの内容

Zabbixの監視データの増加に伴い障害発生状況を表示するWeb画面のデータがいつまで経っても表示されなくなり、運用に支障が出るようになった。



PostgreSQL側で3秒以上を遅延SQLとする「log_min_duration_statement = 3s」を設定したところ、以下のようなログ出力を確認した。

```
LOG: duration: 26019.451 ms statement: explain (analyze,buffers)
SELECT DISTINCT e.eventid,e.c                    cknowledged,er1.r_eventid
FROM events e LEFT JOIN even                     eventid=e.eventid
WHERE e.source='0' AND e.object='0' AND e.objectid='16821' AND e.eventid<='4020491' AND e.value='1'
ORDER BY e.eventid DESC
LIMIT
```

約26秒

対処

複合インデックス追加のみで性能が出ない場合の対処方法-④

■ 調査と分析の進め方

1. 実行計画中の負荷の高いステップを確認
2. インデックスの活用状況を確認

```
～中略～
-> Nested Loop Left Join (cost=1.00..13161.58 rows=1164 width=36) (actual
time=27.116..12548.490 rows=1005 loops=2)
-> Parallel Index Scan using events_1 on events e (cost=0.56..32574.56 rows=1164
width=28)
(actual time=80.380..21875.993 rows=1005 loops=2)
Index Cond: ((source = 0) AND (object = 0) AND (objectid = '16821'::bigint))
Filter: ((eventid <= '4020491'::bigint) AND (value = 1))
Rows Removed by Filter: 8280
～中略～
Planning Time: 155.445 ms
Execution Time: 254
(17 行)
```

WHERE句の5つのカラム中の3つのカラムを網羅したIndexを使用

インデックススキャン(IndexScan)のcost値が高い

Indexで絞り込むことができなかった8280件をフィルタしている

対象のSQLに対して複合インデックスの列が足りないのでは？

対処

複合インデックス追加のみで性能が出ない場合の対処方法-⑤

■ 改善の選択肢

□ 不要なデータを削除する

この案は紹介しない

運用的に許容されるのであれば、対象データを定期的に削除する。

今回のケースでは、監視データの保存期間をZabbix側で設定することによって定期的に削除することが可能。

□ SQLを改修する

参考情報として改修した結果を後述する

ZabbixというOSSから発行されたSQLなので、改修することはできない。

□ インデックスを追加する

この案を紹介する

Filter処理を使わずにインデックススキャンのみでFilterできるようにするために、WHERE句の5つの列が全て盛り込まれたインデックスを追加する。

対処

複合インデックス追加のみで性能が出ない場合の対処方法-⑥

- 複合インデックスを追加

WHERE句の5つの列が全て盛り込まれたインデックスを追加する。

```
# BEGIN;
# CREATE INDEX events_3_add ON events (source,object,objectid,value,eventid);
# COMMIT;
```

eventidが範囲検索(e.eventid<= '4020491')、それ以外は固定値指定なので、追加する複合インデックスの列順は、WHERE句で指定された

source , object, objectid, eventid, value でなく、

source , object, objectid, value, eventid で定義するほうが、

参照するインデックスのページ数が少なくなることが期待できる。

対処

複合インデックス追加のみで性能が出ない場合の対処方法-⑦

■ 改善結果の確認

改善前：約26秒
改善後：約12秒

複合インデックスを追加した場合の例

```
～中略～
Workers Planned: 1
Workers Launched: 1
-> Nested Loop Left Join (cost=1.00..13161.58 rows=1164 width=36) (actual
time=27.116..12548.490 rows=1005 loops=2)
-> Parallel Index Scan Backward using events_3_add on events e (cost=0.56..3586.37
rows=1164 width=2
Index Cond: ((sourc
AND (eventid <= '4
～中略～
Planning Time: 62.7
Execution Time: 12870.059 ms
(15 行)
```

約12秒なので、ZabbixのWeb画面上の表示速度は改善されなかった。
遅い処理となっているのは「Nested Loop Left Join」で、「events」テーブルと「event_recovery」テーブルを結合する処理。
「Workers Launched」で指定された1つのワーカープロセスが処理を実行している。

対処

複合インデックス追加のみで性能が出ない場合の対処方法-⑧

- 改善案

  - マシンのCPUを増やして、パラレルワーカー起動数を増やす

  パラレルワーカーのプロセス数を増やして、複数ワーカーに処理を分担させることによって処理時間を短縮してみる。

対処

複合インデックス追加のみで性能が出ない場合の対処方法-⑨

- **パラレルワーカー起動数を増やす**

1. PostgreSQLのパラメーターを変更

```
max_worker_processes=20
デフォルト値の8→20に変更
max_parallel_workers=16
デフォルト値の8→16に変更
max_parallel_workers_per_gather=15
デフォルト値の2→15に変更
```

2. マシンのCPUを増やす

ワーカー毎に物理CPUが必要なのでサーバーのCPUを20個に増やす。

3. evnetsテーブルとevent_recoveryテーブルへのparallel workersの数を設定する

```
# alter table events set (parallel_workers=15) ;
# alter table event_recovery set (parallel_workers=15) ;
```

対処

複合インデックス追加のみで性能が出ない場合の対処方法－⑩

- **改善結果の確認**

改善前：約12秒
改善後：約2秒

パラレルワーカー数を増やした場合の例

```
# EXPLAIN (ANALYZE,VERBOSE) SELECT DISTINCT ・・・・・
～中略～
-> Gather (cost=1001.00..5859.80 rows=1979 width=36) (
rows=2010
loops=1)
Output: e.eventid, e.clock, e.ns, e.objectid, e.acknowledge
Workers Planned: 15
Workers Launched: 15
-> Nested Loop Left Join (cost=1.00..4661.90 rows=132 width=36) (actual
time=44.235..2316.473
rows=126 loops=16)
Output: e.eventid, e.clock, e.ns, e.objectid, e.acknowledged,
Inner Unique: true
Worker 0: actual time=40.852..2781.879 rows=166 loops=1
Worker 1: actual time=22.437..3165.144 rows=166 loops=1
～中略～
Worker 14: actual time=58.192..641.991 rows=31 loops=1
～中略～
```

EXPLAINコマンドを、VERBOSEオプション付きで実行して、ワーカー毎の処理時間詳細を出力させる

15個のワーカープロセス（Worker:0～14）が処理を分担した結果、約2.3秒まで短縮できた。

参考）SQLを改修した場合の性能確認

本ケースは、パッケージソフトウェアから発行されたSQLであるため、SQLを書き換えることができませんが、SQL自体の問題として、「LIMIT 20」と「DISTINCT」を併用しているという点があります。

「DISTINCT」があるため、WHERE句で対象となる全件（2010件）を抽出する必要がありました。

「DISTINCT」を無くすことができれば、ORDER BY句のeventidの逆順検索で20件を抽出するにとどめることによって、インデックス追加やパラメータを変更することなく、処理時間を短縮することが可能であることを確認しました。

参考）SQLを改修した場合の性能確認

「DISTINCT」を無くしたSQL

```
SELECT ~~DISTINCT~~ e.eventid,e.clock,e.ns,e.objectid,e.acknowledged,er1.r_eventid
FROM events e LEFT JOIN event_recovery er1 ON er1.eventid=e.eventid
WHERE e.source='0' AND e.object='0' AND e.objectid='16821' AND e.eventid<='4020491' AND
e.value='1'
ORDER BY e.eventid DESC
LIMIT 20
;
```

参考）SQLを改修した場合の性能確認

「DISTINCT」を無くしたSQLの実行計画

前：約26秒
後：約1秒

```
Limit (cost=0.87..2207.66 rows=20 width=36) (actual time=51.519..1441.340 rows=20
loops=1)
-> Nested Loop Left Join (cost=0.87..218362.90 rows=1979 width=36) (actual
time=51.511..1441.321 rows=20
loops=1)
   -> Index Scan Backward using events_pkey on events e (cost=0.44..202083.40 rows=1979
width=28) (actual
time=24.283..1215.563 rows=20 loops=1)
Index Cond: (eventid <= '4020491'::bigint)
Filter: ((source = 0) AND (object = 0) AND (objectid = '16821'::bigint) AND (value = 1))
Rows Removed by Filter: 37875
   -> Index Scan using event_recovery_pkey on event_recovery er1 (cost=0.43..8.23 rows=1
width=16) (actual
time=11.284..11.284 rows=1 loops=20)
Index Cond: (eventid = e.eventid)
Planning Time: 133.092 ms
Execution Time: 1441.580 ms
(10 行)
```

まとめと所感

- 毎年のことだが、公開可能なケーススタディの洗い出しと事象の再現に苦労した。

性能トラブル対処に関するノウハウの収集を継続します！
ノウハウが不足する分野などあれば、アンケートにご記入下さい。

PGECons

PostgreSQL Enterprise Consortium