RDBパフォーマンス設計大全

2017.09.19　ミック

Database Lounge Tokyo #5

自己紹介

- 退役DBエンジニア。BI/DWHのシステム開発を経験したのち、パフォーマンス専門チームで性能設計、性能試験、チューニングを担当。
  現在は主に技術者教育を担当。

- Tiwtter @copinemickmack

本日のアジェンダ

- データベースがなぜ性能問題の温床なのか、その理由を明らかにし、どのような方針で設計にのぞむのがよいのかを考えてみる。

  - ✓論理と物理の戦い
  - ✓共有地をめぐるトレードオフ
  - ✓リソースはすべてを解決するか

論物戦争

- 論理は物理から独立可能か？ -

論理は物理に従属するか

- RDBは、物理層隠蔽のかなり早い成功事例と言われる。確かに、物理的な概念を抽象化することで、ユーザフレンドリーになったのは事実。
  - ✓ファイル　⇒　テーブル
  - ✓データのアドレス　⇒　データの値
  - ✓列　⇒　属性

- しかし、パフォーマンス観点からは、物理と論理の分離に成功したとは言い難い。論理設計（ERモデリング、UI/UX、そこから生成されるSQL）は、今に至るまで物理から独立に行うことはできなかった。

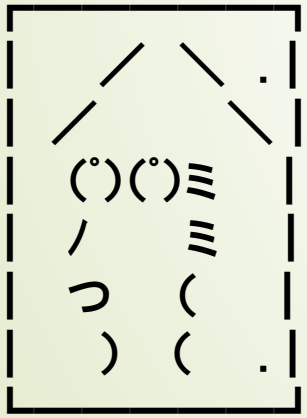
Q. SQLはなぜ遅い？

A. ヒット件数が多すぎる

ヒット件数が多すぎる：パーティション版

ミ(゜)(゜)「億件ごえのトランザクションテーブルか・・・」

ミ(゜)(゜)「これとこれキーにパーティション切ったら50分割くらいできるな・・・よし！ 決まりや！」



特定の1パーティションに8割のデータが集中して無事I/O集中して死亡

ヒット件数の影響は結合が絡むとより顕著

- Nested Loops

  オンラインでもバッチでもパフォーマンスを確保しやすく、重いクエリで多重度がかかった場合でもメモリ消費が少なく、安定性がある。
  どんな環境でも一定水準の仕事をしてくれるDB界のカラシニコフ。

  しかしそれでも問題は残る。

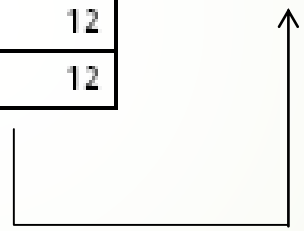
サンプルテーブル

Employees（社員）

emp_id（社員ID）	emp_name（社員名）	dept_id（部署ID）
001	石田	10
002	小笠原	11
003	夏目	11
004	米田	12
005	益本	12
006	岩瀬	12

Departments（部署）

dept_id（部署ID）	dept_name（部署名）
10	総務
11	人事
12	開発
13	営業



Employees

```
DEPT_ID     COUNT(*)
--------------------
10              1920000
11              1280000
12             11710543
13                   63
```

実行計画よりヒット件数（質より量）

APチーム　「テスト環境でSQLの実行計画取ったから遅いクエリがないか見て」

ぼく　「無理です」

APチーム　「ほかに何があればいいの？」

ぼく　「商用環境でのテーブル件数と検索条件・結合条件のキーのカーディナリティがあれば見れます」

ぼく　「ｱｯｲｼﾅｹﾞﾅｲﾃﾞ」

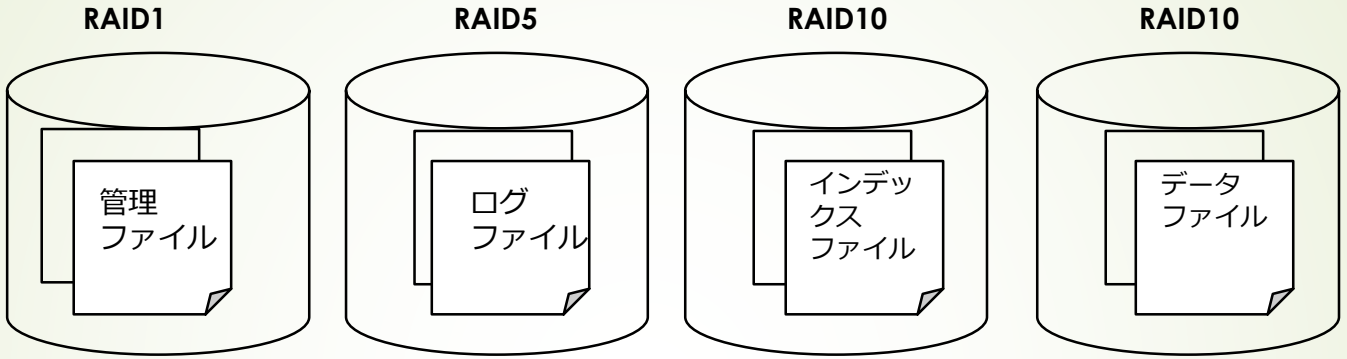
※DBエンジニアに石を投げるのは大変危険ですのでおやめください。

論理は物理に依存する

- DBの性能はアーキテクチャや物理設計が決めると思われているが、実際はERモデリングとUI/UX設計が与える影響が（残念ながら）無視できない。
  - ✓ 巨大なテーブルはどの程度存在するのか
  - ✓ メインテーブルが無駄なコールドデータを保持していないか
  - ✓ 極端なカーディナリティの偏りのある検索・結合キー列が存在しないか
  - ✓ 無駄に自由検索を許していないか
  - ✓ 優先度の低い、それでいてリソース負荷の高い処理に対してリソースキャップはつけられるか。

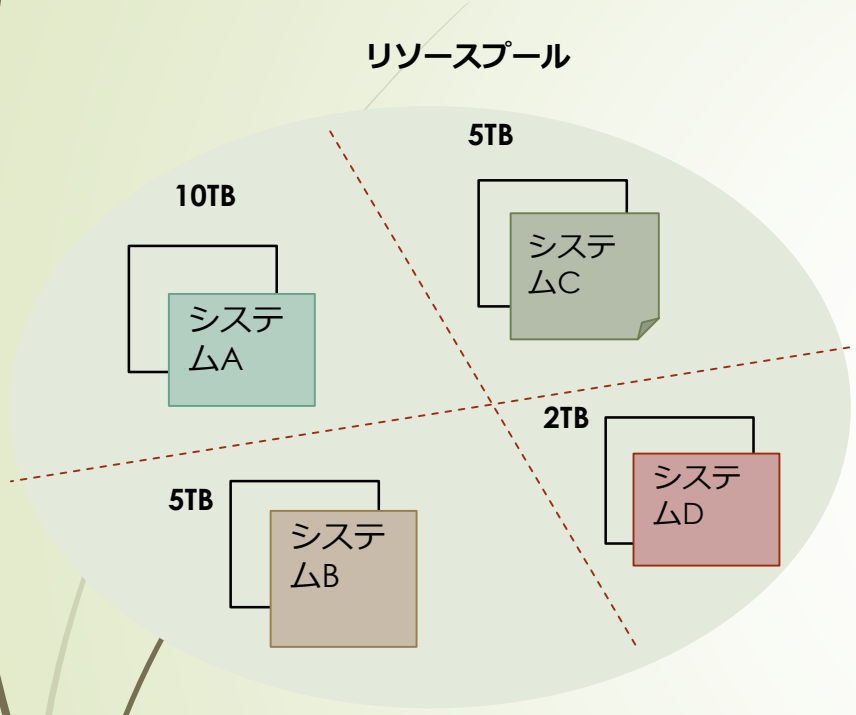
コモンズの悲劇
-共有地をめぐる効率と安全のトレードオフ-

昔の物理設計：分割統治



- メリット
  - ✓ 各領域が物理的に分離されており性能干渉がなく安定
  - ✓ 用途に応じてRAIDレベルやDISK本数を細かく設定できる
- デメリット
  - ✓ リソースが余っていても使いまわせず非効率。
  - ✓ 設計が面倒。

最近の物理設計：シェアリングエコノミー（ごった煮）



- メリット
  - ✓ 無駄にする資源がなく効率的
  - ✓ なんせ設計が簡単
- デメリット
  - ✓ 物理で帯域共有するので巻き込み事故多発

巻き込み事故多発により結局、帯域のキャップ＆トレードが導入された。

他分野でのキャップ＆トレード

システムに限らず、共有可能な資源を扱う経済活動では共有地の悲劇が発生するのは珍しくない。また、それを防止する手段も共通になる。

- **CO2**

  https://www.kankyo-business.jp/news/014472.php

  東京都キャップ＆トレード制度　2015年度のCO2排出量、前年比－1%に

- **水産資源**

  http://wedge.ismedia.jp/articles/-/3477

  漁業 日本のTACはなぜ7魚種しかない？

  ※TAC=Total Allowable Catch: 漁獲可能量

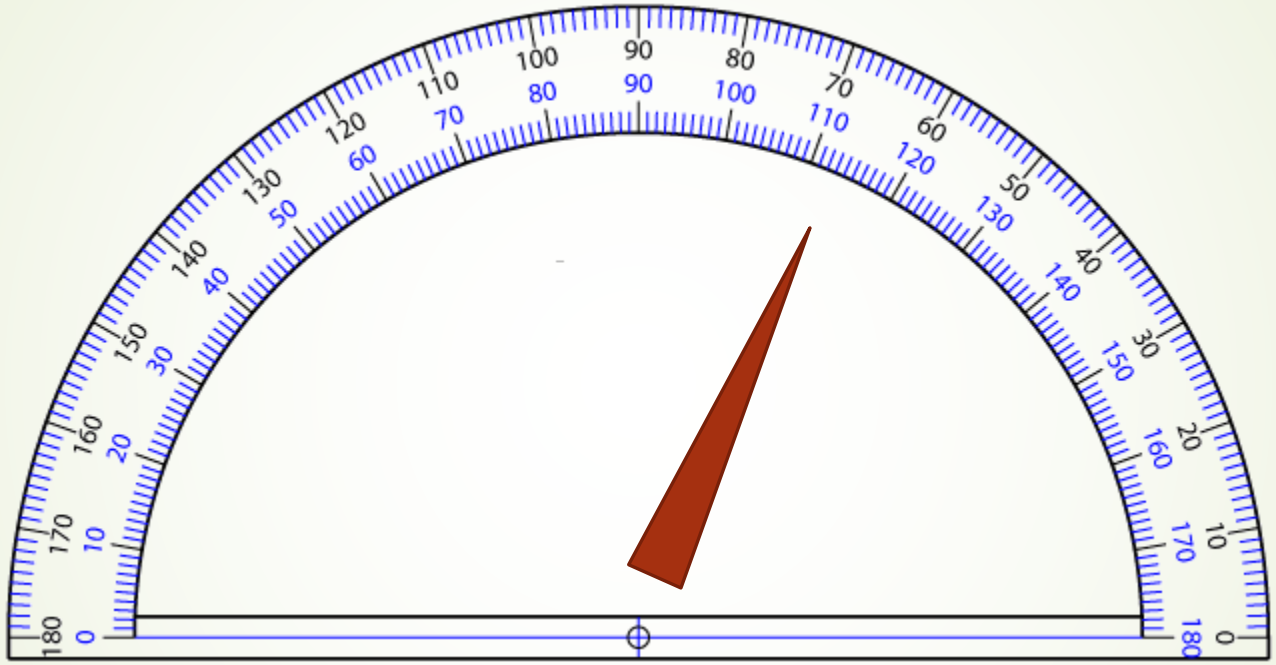
[余談]ごった煮の起源

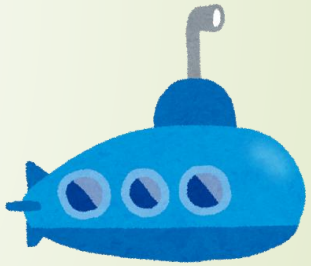
- ストレージの共有リソース化はどのように進展したのか。CPUやメモリなどのリソース仮想化と共有（ex. オーバーコミット）が先に進展し、それに倣う形で進んだ。

- Oracle社は早くからストレージ仮想化の設計方針（S.A.M.E）を提出しており、DBストレージの共有資源化の源流の一つとみなせる。
- https://blogs.oracle.com/dbjp/oracle-database1

“オラクルはディスクへの物理配置の基本的なコンセプトとして「Stripe And Mirror Everything（S.A.M.E.）」、つまり「すべてのディスクに均等に負荷がかかるように、データをストライプしてすべてのディスクに分散配置する」ことを提唱しています。”

安全と効率のトレードオフ





- われわれは常に安全と効率のトレードオフに直面している
- システムも経済活動である以上、コモンズの悲劇を避ける努力（ダメージコントロール）が必要

物量神話
-リソースはすべてを解決するか-

戦いは数だよ、兄貴

- オンメモリ
- リードレプリカ
- フラッシュストレージ
- 次世代不揮発性メモリ
- 量子コンピュータ

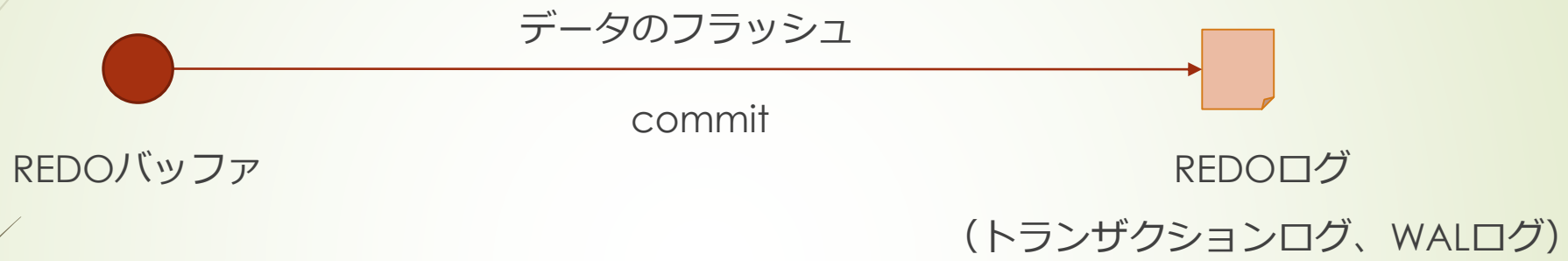
我が国アメリカが世界一の資源豊かな国であることをいつか証明してみせる。

———J.P.モルガン

それでも非効率なAPロジックは残る

- オーバーヘッド・コストが高すぎる
  ex. ぐるぐる系に伴うラウンドトリップ、パース
- AP・ミドルの内部のリソースを制限する機構
  ex. シリアライズ、キュー、帯域制御
- ロック競合
  ex. 典型的には書き込み時の排他ロック

REDOログの書き込み競合



- 根本的にはRDBのWALにスケーラビリティがなく、シングル・ボトルネックポイントであるために生じる。
- 基本はコミット頻度の高いバッチで生じる問題。しかし、オンラインで問題になるケースもある。

Writeによるボトルネックはバッチでしか生じないと思っていた時代が、私にもありました

- ログ・マシンガン

  監査や行動分析ログが大量に吐かれる。別DBかファイルで吐きましょう。テーブルの方が加工が楽？

- セッション情報

  ECサイトでよくおきる。KVSなどキャッシュで受けましょう。なおパッケージの仕様で詰むことも。

- 業務仕様による大量データ登録

  RDBのみでは対応に限界のあるケース。フロントでキャッシュする仕組みが必要になることも。

リソースは大抵の問題を解決するが

- リソースへの投資は費用対効果が高い

  ボトルネックの場所さえ間違わなければ、ハードウェアへの投資は裏切らない。

- APやミドルがリソースを使いきれるか注意しよう

  特にデータベースはシングル・ボトルネックポイントができやすい。

- ビジネスロジックと無縁ではいられない

  どのような性能特性を持っているのかは、ビジネス側の要件と密接に関係しているので、インフラに閉じて解決する問題ではない。

死亡事故で打線組んでみた

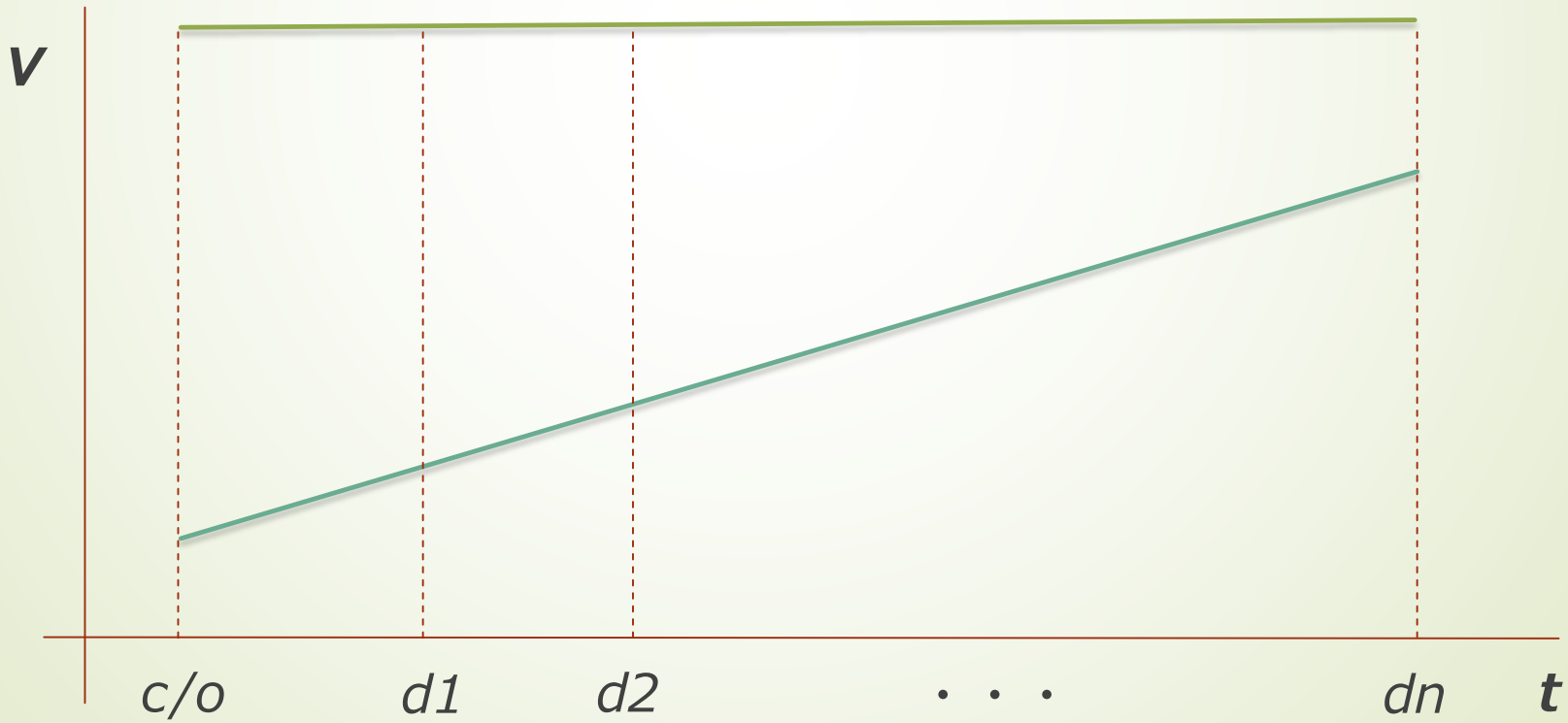
- 1（中） Covering Indexでチューニングしたクエリに機能追加の名のもと列を追加され、無事Uncovering Indexになり死亡。
- 2（右） ユーザ「画像の解像度が粗くて見づらいなあ・・」APチーム「解像度あげましょうか」⇒LOBサイズ増大、I/O帯域を食いつぶし死亡。
- 3（二）ファイルで処理していたぐるぐる系をRDBにリプレースしたらオーバーヘッドだらけでストレージ帯域を一切使いきれず死亡。
- 4（一） BIなのにストレージ接続がNAS。ネットワーク帯域がパンクし、ストレージもサーバもスカスカなのに死亡。
- 5（左）基盤統合の号令のもと、基幹系システムと情報系システムを同じストレージプールで稼働させ情報系にすべてを持っていかれ死亡。ベンダは両方いけるっていってたのになあ。
- 6（三）DBMSの全文検索機能を使ったらインデックス更新時のReadが大量発生して他の業務クエリを巻き込み死亡。検索性能そのものは悪くなかっただけに、あと一歩。
- 7（遊）業務系システムなのにフリーダムUIで検索キーが絞れず死亡。節子、それ業務系と違う。
- 8（捕） Nested Loopsで遅かった実行計画をHASHに変えたら多重度かかってストレージ落ちして死亡。
- 9（投）運用中にデータが増加し、実行計画変動により直下型地震が発生し死亡。

（2016年度 警視庁調べ）

終局とは何（だったの）か

システムのサービス化（as a service）とCI/CDの普及により、「終局」の概念が意味をなさなくなってきた。

Q.どのタイミングで性能を担保すればよいのか？



Nested LoopsとHash

- 可能な限りNested Loopsベースでチューニングするのが基本。それだけに内部表のヒット件数が大きくなるとつらいので、ヒット件数を絞れるように論理設計で可能な限りつぶす。

- Hashはメモリ不足によるストレージ落ちリスクと背中合わせ駆動表が小さいことに確信があるか、多重度が見切れている場合でなければ安易に使うのはリスク含み。

  （MySQLが**HASHを持たない**のは、Webサービスに特化したDBとしては合理的な選択だと思われる）

今後のDBとパフォーマンス設計

- ✓ 物理層の隠蔽は今後もますます進むと思われる。パブリッククラウドを利用しない場合でも、その傾向は止まらない（プライベートクラウド、HCI、コンテナ）。
- ✓ しかし、冒頭で述べたように、論理設計が物理層から自由でない以上、論理と物理の両方のレイヤをカバーするエンジニアが必要になる。
- ✓ システム開発が有限の物理リソースを使う経済活動であるかぎり、共有リソースとトレードオフの二つの経済原則を無視することはできない。

追記：REDOネックを解決する新技術

- 「REDOファイルの書き込み競合を回避する手段はあるか？」という質問に対して、講演後に教えていただいた分散REDOログ技術silo。
  実用化されれば、書き込みをオフロードするためにKVSを使う必要もなく、RDBですべて受けられるようになるかもしれません。

- SILO再考～次世代DBのアーキテクチャとして
  http://d.hatena.ne.jp/okachimachiorz/20161003/1475494676

- Silo – Qiita
  http://qiita.com/kumagi/items/af6bed1424367927fb4a

END