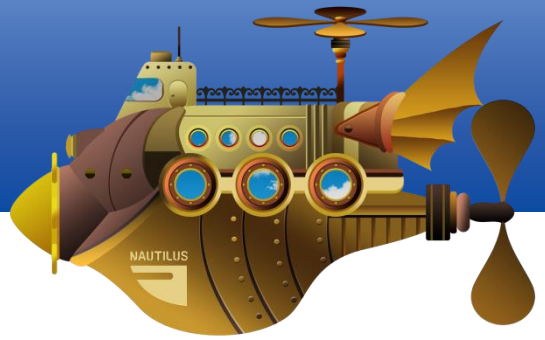
Tsurugi（劒）コア：進捗説明

2021/10



NAUTILUS

概要(再掲)

- Tsurugi（劔）の概要
- OSSで、メニーコア・大容量メモリーの現在のハードウェア環境に合致する、モダンアーキテクチャなRDBをつくる
- RDBだけつくっても使われないので、プロトアプリケーションも作って可能な限り（というか意味がある限り）公開する
- Serializableな（というかのみ！）DBです

復習

- そもそもなんで今更「RDB」なのか。
- 1.環境にあったRDBがいつまで待ってもでてこない
- 2.商用DBはOとMSだけ。OSSはMySQLとPostgreSQL。全部旧来のアーキテクチャ
- 3.すべてlockベース/diskベースなので、普通に大容量メモリー/メニーコアが使い切れない
- 4.RDB/Txの研究はとっくの昔(1980年代以降)にMVCC/Ts/メモリーベースになっている
- 5.「イノベーションのジレンマ」でいつまでたっても出てこない・・・なら、作りましょう

この時代でRDBをつくるのであれば

- OSSは前提 / Apache2.0
  - ただし公金（日本の税金）が入っているので、NEDO一般の制約はつくはず。まだ決まってない

- 一般に使われているOSSの外側はそのまま利用する
  - Postgresを採用
  - ユーザへの敷居を下げる
  - 独自のDBMSを作る意味はほとんどない

世界最強のtransactionプロトコル to date はなにか？

NAUTILUS

- SILO
  - In memory主体 / Timestampベース /Epoch base /コアスケーラブル
  - ベンチマークチャンピオンで10年間君臨
  - なぜか？「できるだけ何もしないから＝楽観処理」

- 商用ベースはまだない
  - なぜ、とっとと実用化されないのか？
  - 致命的弱点があるから

- long transactionがほぼ確実に一つも通らない
  - よほどのことがないと全滅　なので、よって使い物にならない
  - ということで、過去様々なプロトコルが名乗りを上げて、SILOに挑戦してきた
    - Foedus、SSN、Cicada、ERMIA, Tic-Toc、MOCC、etc.
    - 結局、SILOには勝てず・・・

Tsurugi（劒）

- では、どーするか？
  - SILOをうまく利用し、SILOを生かしつつ、long transactionを通るようにする
  - SILO自体はOCCであり、そもそも大量の書き込みが想定されていないので、できるようにする
    - まぁ言うのは簡単ですが、これ水と油を混ぜるようなもので・・・

- **Hybrid Concurrency Controlの導入**
  - OCC+batch用処理（long transaction）のマルチCC
  - OCCだけの状態であれば、そのままOCCで処理
  - long transactionが混在したら、OCCはOCC＋αに、long transactionは独自のプロトコルで処理をする

- **long transactionは独自プロトコル**（Shirakami“白神”）
  - Timestampベースでlock free (read/writeもlockは取らない)
  - Epochベース
  - Write preservationの導入（write lockではなく“表明”的なナニカ）
  - Read alignmentの導入によるconflictのmitigation
  - 悲観処理と楽観処理のハイブリッド
    - 優先度の導入。最優先はすべての処理に勝つ。
    - ただし指定には制限あり、そりゃ全部に最強にしたら意味ないんで

- 基本的にMVCC/Timestampベース/ECCを完全に踏襲している

TimestampベースのRDBです

- **よってSQLはそのまま字義通りに動くが挙動が旧来のlockベースとは根本的に異なる**
  - lock制御はしない
  - txがserializableでなければ、そのままabortする
  - 挙動としてはいわゆる“NOWAIT”っぽい感じになる
    - 制御側で頑張るところまで頑張るがserial実行はしない
- 普通にSQLベースのRDBとしてそのまま稼働する
  - まずは普通に動くはず
  - ただ、コア少なめ＋メモリー少なめ＋SQLオンリーの従来の利用であれば、そのままのPostgresでいいと思います。別にTsurugiじゃなくても・・・
- ただしTSベースなので、SQLはそのまま動きますが、結果の挙動が変わる（結果は同じ）ので・・・
  - Postgres互換は互換ですが、「複雑なPostgresのアプリをなにもしないで放り込んでも（たぶん）動きません」
    - 動きはするがabortが起きやすくなるはず

“アプリケーション”としてのRDBではなく、“ミドルウェア”としてのRDBになる

- いくらでも/いろんな細工ができます。
  - ワークロードを見切って、うまく制御してやると簡単にパフォーマンスがあがる（はず）
    - つまりその逆もあるということです。
  - さらに設定パラメータをうまく最適化すると、さらにパフォーマンスがあがる（はず）
  - さらに提供される「仕組み」に“実装”をうまく入れ込んでやると、さらにパフォーマンスがあがる（はず）
    - このレベルであれば、アプリ専用のDBがserializableで作ることができます。
- **ただし、O社のような“いたりつくせり“は一切期待するだけ無駄です。**