DB - Final Project

Team 15

104062203 陳涵宇
104062232 廖子毅
104062234 林士軒

- Basic Information
  - Title : Optimistic Concurrency Control
  - Conference : ACM Transactions on Database Systems (TODS) @ June 1981
  - Authors : H. T. KUNG and JOHN T. ROBISON @ Carnegie - Mellon University
- Main Idea

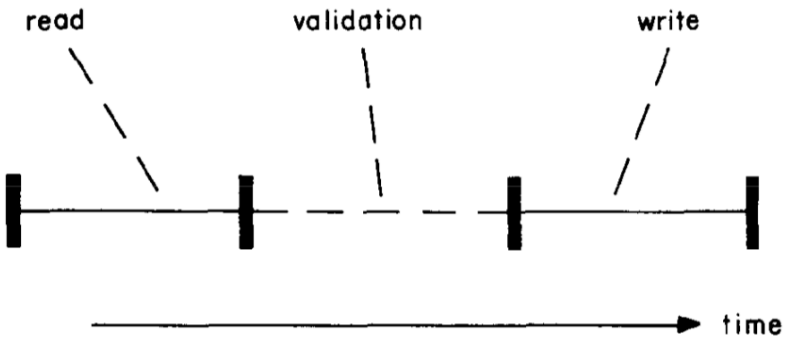


Fig. 1. The three phases of a transaction.

  - 透過將 tx 分為三個 phase (read, validate, write) 來完成一次 tx。
  - 將資料分為四個 Set (create, read, write, delete) 不直接執行 tx, 會將資料 maintain 在 Set 中, 在最後的 write phase 再一次寫入。
  - read phase 就是去接收 query 來修改及 maintain 這四個 set。
  - validation phase 則是需要去檢查這次的 read phase 中讀取出來的值有沒有被別人修改過，在 paper 中，分成 serial 和 parallel 兩種 validation

的方式，serial 只需要檢查從開始 tx 到 commit 之前有 commit 的 tx，而 parallel 的則需要額外去檢查目前也正好在 commit 中的 tx。

    - write phase 是等 validation phase 確定了這次的讀取都是 valid 的，如此一來，這次的 tx 才算是一個有效的修改，才可以寫進 db 中。

- What we implemented

    - 根據上面的 main idea，我們實作了其中的三個 Set (read, write, delete)。其中的 create Set 並沒有特別去實作，原因是 create 只是一個概念，並沒有一個 query 來執行，以最相近的 query INSERT 來說，是一個 create + write，在 paper 上可以看到，這就是一個直接寫入的動作，所以我們這邊就不實作 create set。

    - read Set 部分：read set 中，簡單紀錄是讀取了哪個位置。首先，先判斷是否有在 write Set 中。如果在其中，則直接回傳在 write Set 中的值。若是沒有在 write Set 中，則使用原先的方式查找。

    - write Set 部分：將原先要直接寫入的值，先暫存到自己 tx 中的 private(local) writeSet workspace 內。若有相同位置的新值存入，則取代舊有的值。這就是我們在 as5 中實作的 private workspace

    - delete Set 部分：同 write Set 將欲寫入的值先在本地端維護，在 commit 或 rollback 的時候再一次做完。

    - 這邊是使用 parallel validation 的方式來實作，畢竟使用 serial validation 反而會成為 OCC 的一個 bottleneck。

    - tx commit / rollback 機制：根據上述的OCC，我們在一個 transition 會有三個時期。

        - 1. 讀取時期：執行當前 tx 中的 sql , 並依上述條件進行分類。

        - 2. 驗證時期：驗證 tx 中的 readSet 是否有與其他正在執行的 txs 的 write Set 或是 delete Set 衝突。若衝突發生，則 abort 此 tx。反之，則進入下一個時期。

        - 3. 寫入時期：依據 write Set 或是 delete Set 的順序來刪除資料。

- Evaluation & experiments
  - Env.

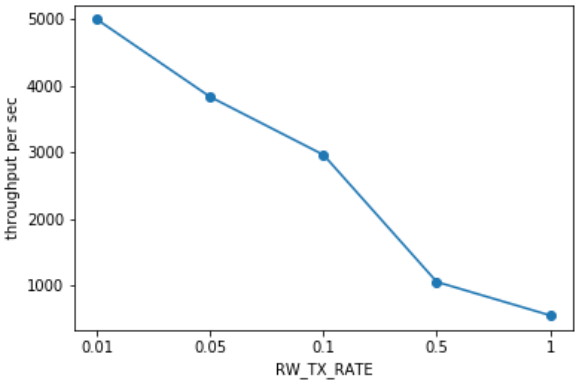
  - Exp. I - Change RW_TX_RATE to verify
  - Configurations

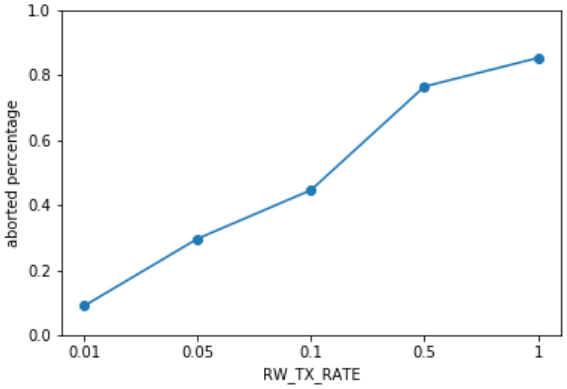
```
# The running time for warming up before benchmarking
org.vanilladb.bench.BenchmarkerParameters.WARM_UP_INTERVAL=30000
# The running time for benchmarking
org.vanilladb.bench.BenchmarkerParameters.BENCHMARK_INTERVAL=60000
# The number of remote terminal executors for benchmarking
org.vanilladb.bench.BenchmarkerParameters.NUM_RTES=10
# The IP of the target database server
org.vanilladb.bench.BenchmarkerParameters.SERVER_IP=127.0.0.1
# 1 = JDBC, 2 = Stored Procedures
org.vanilladb.bench.BenchmarkerParameters.CONNECTION_MODE=2
# The path to the generated reports
org.vanilladb.bench.StatisticMgr.OUTPUT_DIR=
# The granularity for summarizing the performance of benchmarking
org.vanilladb.bench.StatisticMgr.GRANULARITY=3000
# Whether the RTEs display the results of each transaction
org.vanilladb.bench.rte.TransactionExecutor.DISPLAY_RESULT=false


#
# Micro-benchmarks Parameters
#

# The number of items in the testing data set
org.vanilladb.bench.benchmarks.micro.MicrobenchConstants.NUM_ITEMS=100000
# The ratio of read-write transactions during benchmarking
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.RW_TX_RATE=0.01
# The number of read records in a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.TOTAL_READ_COUNT=10
# The number of hot record in the read set of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.LOCAL_HOT_COUNT=1
# The ratio of writes to the total reads of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.WRITE_RATIO_IN_RW_TX=0.1
# The conflict rate of a hot record
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.HOT_CONFLICT_RATE=0.001
```

- Result



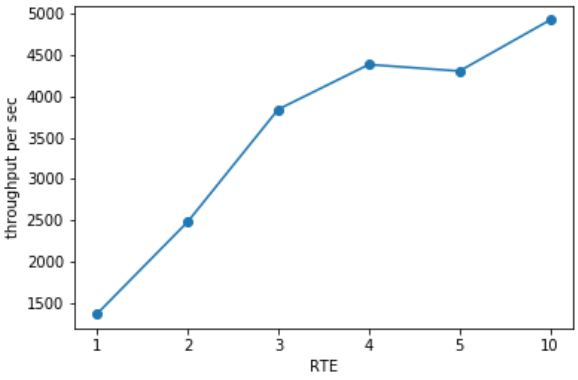


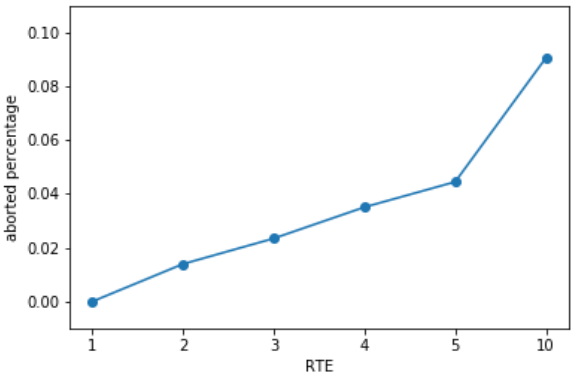
- Exp. I - Change RTE to verify

- Configurations

```
# The number of items in the testing data set
org.vanilladb.bench.benchmarks.micro.MicrobenchConstants.NUM_ITEMS=100000
# The ratio of read-write transactions during benchmarking
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.RW_TX_RATE=0.01
# The number of read records in a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.TOTAL_READ_COUNT=10
# The number of hot record in the read set of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.LOCAL_HOT_COUNT=1
# The ratio of writes to the total reads of a transaction
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.WRITE_RATIO_IN_RW_TX=0.1
# The conflict rate of a hot record
org.vanilladb.bench.benchmarks.micro.rte.MicrobenchmarkParamGen.HOT_CONFLICT_RATE=0.01
```

- Results





- Analysis for experiments
  - 實驗一分析：在寫入上升時整體輸出效率會大幅下降。此外，Tx Abort 率也會飆升。
  - 實驗二分析：
    - 1. 在 RTE 為 1 的時候，因為不會與其他人的 write set 做檢查，所以不會有 abort 。
    - 2. 在RTE 逐漸上升的時候，會達到一個效率的飽和點。原因為系統的資源有限，即時更多工的處理仍無法提升效率。
    - 3. 在多 Tx 並行執行的狀況， abort 率會逐漸上升。原因為當一次執行越多的 Txs，則會有更高的機率發生衝突，導致 Tx Abort。
  - 由實驗一、二結果，可知此次實作 OCC 為成功的。(在低 conflict rate 輸出效能很高; 在高 conflict rate 則輸出大幅下降。)
- Problem we encountered and conquered
  - 由於我們在執行 insert sql 是直接寫入，而在執行 delete sql 時則是先暫存到 private workspace。所以我們實作時在 rollback 中會出現問題。因為原先的 insert 和 delete 會相互交換，所以在 tx 不會 delete 掉先前 insert 值，而是又被放入 delete Set 中。
  - 未注意到在 exception 後才做資料的 commit 及清空，導致程式不正常運作許久，發現後即完成 OCC 演算法的實作。
  - 詢問助教改良 index Lock 部分的方法，將原本的 index Planner 換為最初的 BasicQueryPlanner，發現輸出效能大幅降低。
- Conclusion
  - 此次實作 OCC 應該算蠻成功的。在實驗上，我們有依照論文中的假設讓資料庫原先既有資料量變大，執行 sql 時也盡量避免寫入相同的 Block，才使得圖表符合論文所說。關於實作時遇到的問題，也因為能夠及時地問助教，所以沒有耽擱我們許多的時間，很順利地完成了。