Cloud Database Final Project

Implementation

大致上可以從 Naïve 的版本來修改，最大的差異在於每個機器有自己 Partition 的資料（使用 Round-Robin 去分配）、以及要傳送跟蒐集 Remote Read（依照 Calvin Paper 實作）。

• Benchmark

Package： ***procedure/vanilladddb/calvin***

(1) **MicroBenchMarkProc.java**

因為 Server 運算自己的 Partition 資料是在下層的 CalvinCacheMgr 判斷掉，所以基本上不變。

(2) **SchemaBuilderProc.java**

因為所有的 Server 都要看到同樣的 Schema，所以不作修改。

(3) **TestbedLoaderProc.java**

原本的設計是每台 Server 都會產生出全部資料，這邊修改 *generateItems()* 內部，把 schema 跟 data 包進 RecordKey 中，並使用 partitionMetaMgr 來判斷這筆 record 是否要存進該 server 中，已減少之後搜索的時間。

(4) **TpccStoredProcFactory.java**

基本上執行邏輯沒變動。

• VanillaDdDb

Package： ***cache/calvin***

(1) **CalvinCacheMgr.java**

基本上使用 partitionMetaMgr 來判斷是否要處理這個 RecordKey。在 Calvin 的設計中，需要去讀 Remote Read。創一個<TupleKey, CachedRecord> 的 HashMap 來蒐集 Remote Read。新增兩個 API，*addCachedTupe()* 來蒐集從別台 Server 傳來的資訊、cleanCachedTuples() 則是於該 StoreProcdure 執行完後，清空這個 tx 所存下來的 Remote read。

(2) **TupleKey.java**

在 VanillaDbComm 的設計中，是以 Tuple 作為資料單位。擷取出必要的 RecordKey 跟 txNum 資訊以做區別。

Package： ***remote/groupcomm/server***

(1) **ConnectionMgr.java**

為 Calvin Paper 中所提的第四步，蒐集 Remote Read，並丟入 cacheMgr 中。

Package： ***schedule***

(1) **DdStoredProcedure.java**

新增兩個 API 用以判斷是否為 master 跟 participant node。
（詳情於 Calvin Paper 有說明）

Package： ***schedule/calvin***

(1) **CalvinScheduler.java**

只讓握有相關 record 的 participant node 去拿 lock 跟執行 tx 邏輯。

(2) **CalvinStoredProcedure.java**

照助教 FAQ 所提，在拿到所有 lock 後，直接在 *prepare()*中呼叫 *analyzeRWSet()* 進行 Read/Write Set 的分析。*analyzeRWSet()*為 Schedurer 所要作的第一步驟，實作細節如 Paper 所提，需要注意的地方在於 Load Testbed 時不算拿到 Read/Write Set，所以需要額外處理，另外選一個 Server 當 master（此處選 node id = 0 的 server，因為一定會存在）。

接著在 *execute()*中實作 Calvin Paper 所提的第二、三、五步驟。先 local read 完，把資料包進 Tuple 中，並存入 TupleSet 裡。在呼叫 ConnectionMgr 傳送給其它 server。最後只讓 activeParticipants 去呼叫 performTransactionLogic()。

(3) **CalvinStoredProcedureFactory.java**

邏輯沒變。

Package： ***schedule/naive***

(1) **NaiveStoredProcedure.java**

因應 DdStoredProcedure 實作 *isMaster()*跟 *isParticipant()*，因為 Naïve 是 full-replica 所以每台都要處理並回傳資料。

Package： ***server/task/calvin***

(1) **CalvinStoredProcedureTask.java**

呼叫 isMaster() 判斷自己是否為 Master，只有 Master 才會回傳結果給 Client。

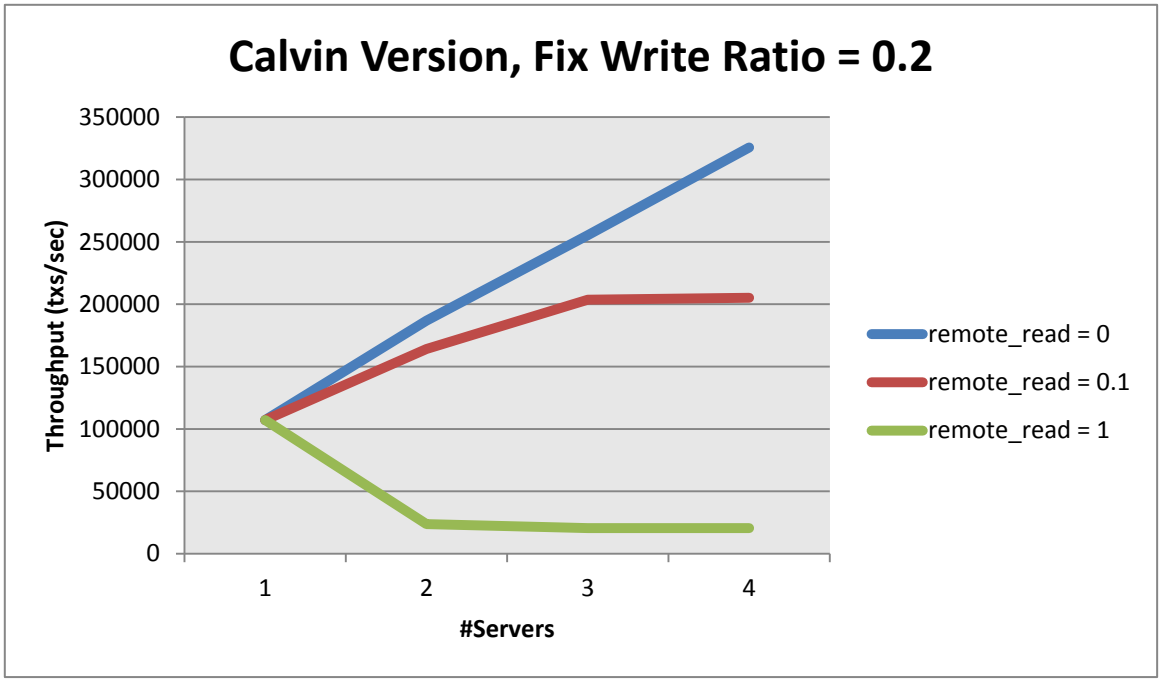
Experiment

- **Environment**

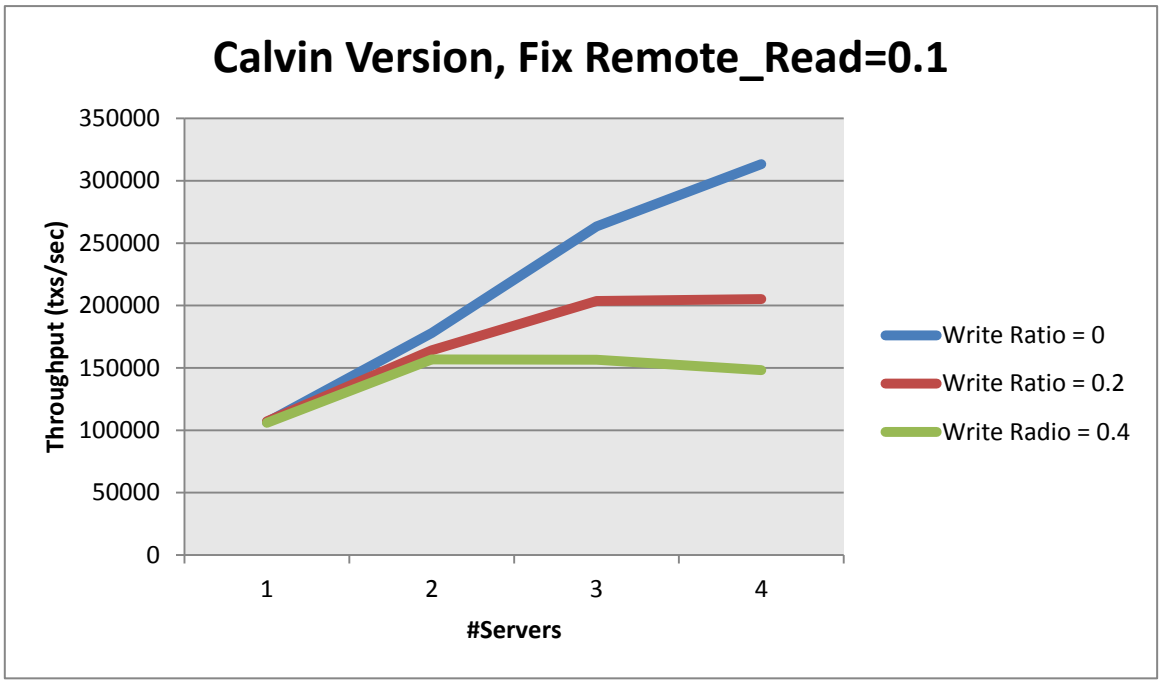
OS	**OS X (FreeBSD Based)**
CPU	2.6 GHz，Intel i5
Memory	4 GB
Disk	HDD

- **Results**

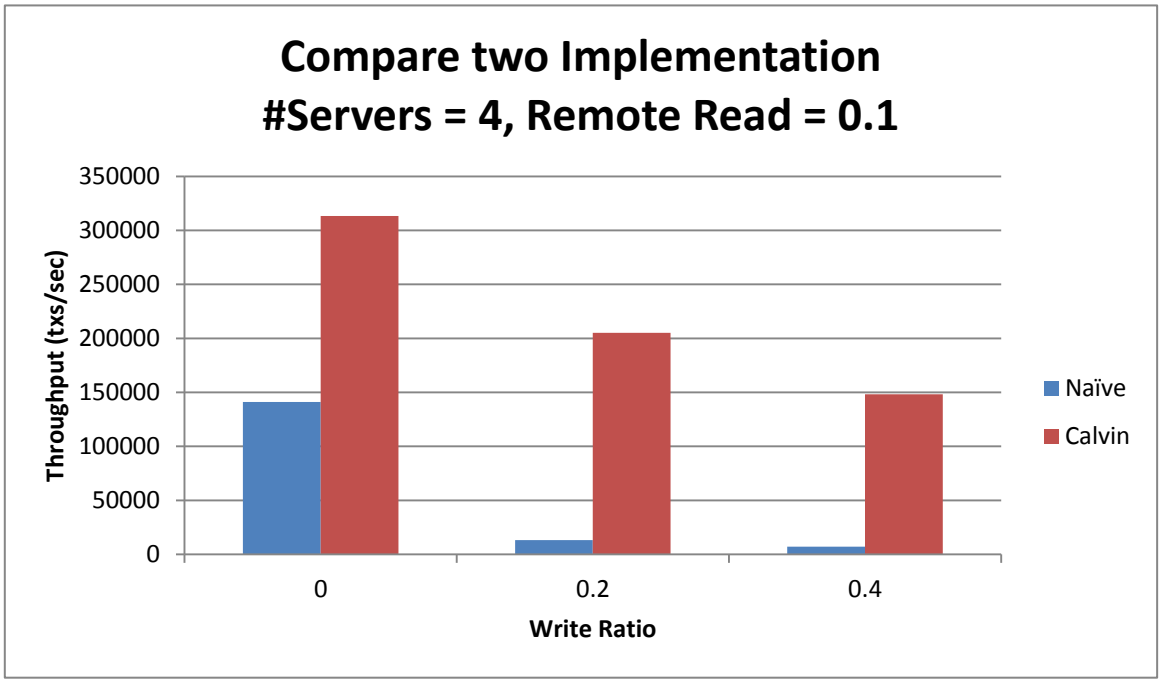
每台 Server 的 partition 為 **100,000** 筆資料，每增加一台 Server，#RTE 增加 **25**
。



大致上與 Paper 的實驗結果相符，remote_read = 1(100% distributed txns)在初期幾個 Servers 效能會降低，照 Paper 的說明是因為等需要的 remote read 的關係。而這種情況在 #Servers 變多的時候會有些許改善，因為 Server 就可以執行其他沒有 conflict 的 tx。



隨著 Write Ratio 的增加，需要更多的 Remote Read，可以想見效能會下降。但基本上 Write Ratio 所佔的比率不太可能出現 40%如此高的值，所以整體來說 Scalability 相當的好。



由於#Servers 少的時候，Full replication 版本所要處理的資料量不大，並且少了 Network Communication 的 Overhead，所以會比 Calvin 有優勢，因此就直接比較#Servers 較多的結果。

可以看出因為拿 lock 以及搜索資料數的差異，讓 Calvin 相較於 Naïve 的版本都有相當大的效能提升，與預期相符。