Final Phase One

106062128 陳頌恩
106062328 李思佑

1. Implementation

Communication:

- Client:

  藉由在CalvinConnection中建立VanillaCommClient來接收Server所發送的訊息，以及傳送StoredProcedureRequest給Server。使用CallStoredProcedure實際上是傳送打包好的訊息(SPRequest)給Server，並等待Server回傳結果。當收到Server的訊息時，Client解讀之並傳給CallStoredProcedure。
- Server:

  透過在ConnMgr中建立VanillaCommServer來接收Client的訊息，並達成Server之間的溝通，以及傳遞結果給Client。

  當接收到Client的請求時，Server會先以totalOrder的方式確認每台Server接收的順序一樣，確認之後Server會將請求交給Scheduler處理，再將結果由MasterServer傳回給Client(MasterServer由請求的Partition決定)。Server接收到另一台Server的RemoteRead時，則將其傳給CacheMgr存起來。

Scheduler:

繼承vanilladb.core.server裏的Task（implements Runnable interface）。scheduler擁有一個含有stored procedure request 的 blocking queue，在排程時將 stored pocedures 放入上述的 queue中。接著建造一個新的storedproceduretask（for multithreading）送給lock manager 去做request lock的動作。最後用taskmanager把這個task指派給一個thread去完成。

Metadata:

以每個RecordKey(也就是能唯一識別一個Record的物件，在此用TableName + PrimaryKey來實作)的HashCode來分配Partition，需要事先設定好有多少Partition，也就是有幾台Server。

Recovery:

將StoredProcedureRequest寫到logFile，當需要Redo時，則呼叫ConnMgr將log所含之StoreProcedureRequest重新以TotalOrder送出，以達到Redo之效果。

Concurrency:

依照vanilla core原本的 locktable, 新增requestlock 的函數, 在concurrency manager采用兩階段的lock。

分別是在scheduler 做 register lock(request lock)。以及在storedprocedure(同樣在scheduler的package)做 acquire lock。

Stored Procedures:

主要是改變excute以及prepare的做法, 依照論文中的步驟, 在prepare中進行Read/Write Set分析, 決定各個Record是read或write, 順便分析誰需要RemoteRead。在execute中可以先把lock拿好, 接著利用CacheMgr讀取Local Record, 再來用ConnMgr傳送RemoteRead給需要的Server, 而Server接收到RemoteRead後, 將其存在Cache中, 最後就是執行Transaction logic和Write。

在Bench中也需要改動MicroTxnProc, 使其實際定義Transaction Logic需要做甚麼(會使用CacheMgr來執行讀/寫)。在此也需要事先準備好RecordKey, 以方便read/write set Analysis, 使後續拿lock和確認partition可以順利進行。

Cache:

將實際上的Record與RecordKey做連結並儲存的地方, 也是呼叫Core的各種Plan來實行讀寫之處。負責儲存從Server傳送來的RemoteRead, 以備執行時使用。

2.Experiment

Run Configuration

Program arguments:

除了原本的參數外，需另外加Client/Server的編號

VM arguments:

增加

-Dorg.vanilladb.comm.config.file=target/classes/org/vanilladb/comm/vanillacomm.properties

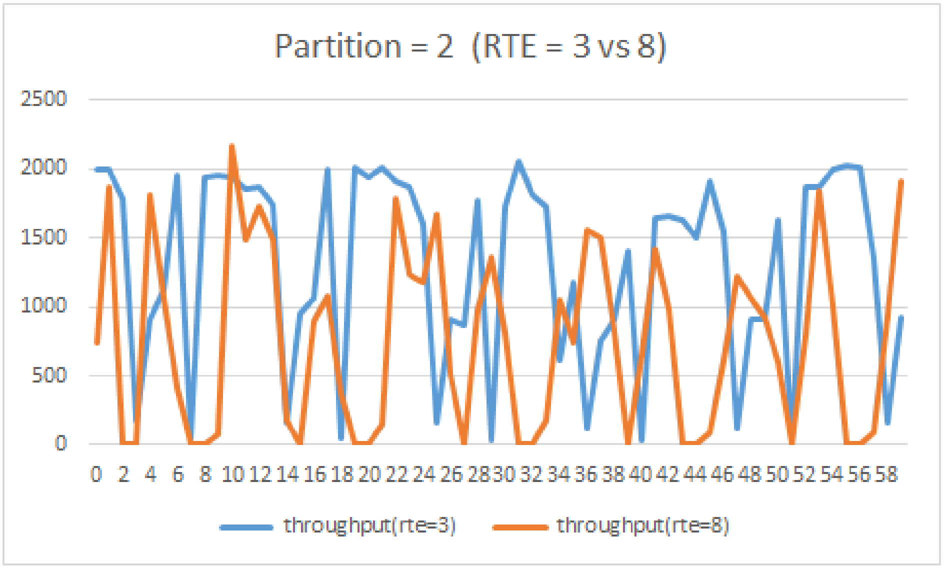
Enviornment:

Intel Core i5-8300H CPU @2.30GHz, 8 GB RAM, 128GB SSD, Windows10

Results:

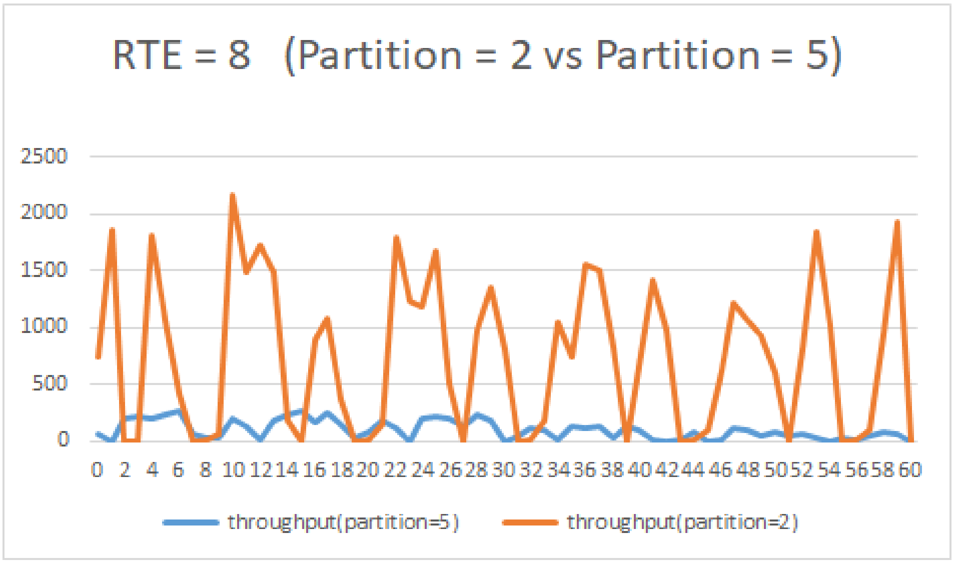
控制變數: RTE

1 client 2 servers Microbench (“Partition” equals “Server”)



控制變數: Partition

1 client Microbench (“Partition” equals “Server”)



- 可以觀察到Throughtput 有週期性的起落，而在觀察ServerDemo與ClientDemo互動時也有同樣的模式，因此推測可能與comm中底層實作有關。

- 測試Partition(Server)數目差異時，可以看到較少的Partition有較好的Throughput，原因可能是Server之間溝通的成本較高，導致Throughput下降。

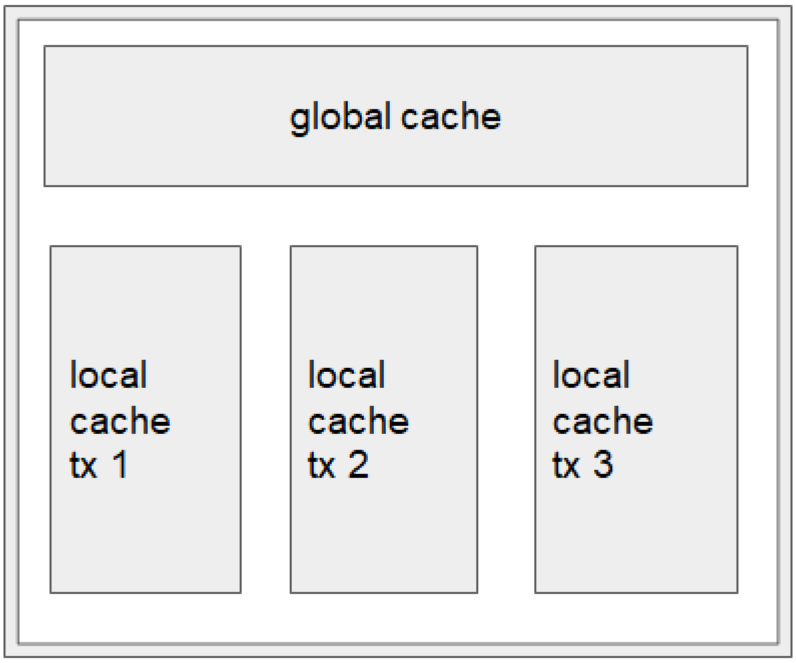
Final Optimization（Phase Two）

106062128 陳頌恩
106062328 李思佑

1. **Optimization**
   目標:**Reduce Latency of Transactions with 3 servers**
   實作內容:
   (a). Find a master server for every transaction
   We choose a server as master, it will collect remote reads from other servers, and then send them to clients. We use this mechanism to avoid sending redundant data to clients. So we can reduce time spent on communication, thus lower latency.
   Note that in our experiment, we find this optimization sometimes lead to higher latency, because the master may finish the stored procedure slow. Therefore, we choose the master in the way it may execute faster. In particular, we choose the server with most local read/write, since it has less remote read and network latency.

   (b). Add global cache in each server

   When cacheMgr read records, we put records into global cache. When the same record is read, we get the record from global cache. This method can reduce I/O time, thus reduce latency.



2. Experiment

Environment :

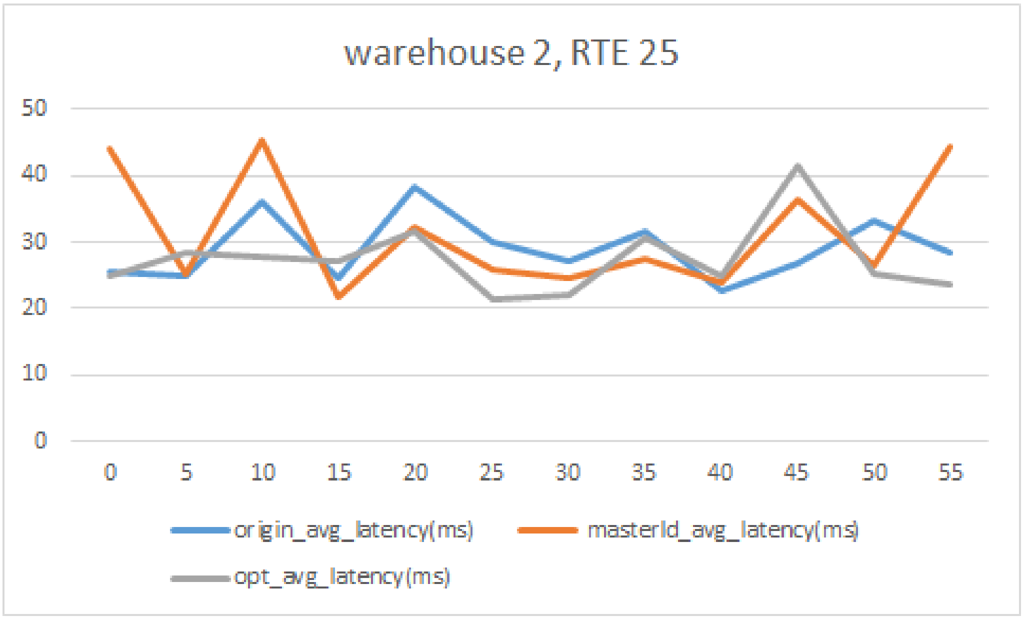
intel i7-8700@3.2GHz, 8GB RAM, 256GB SSD, Windows10

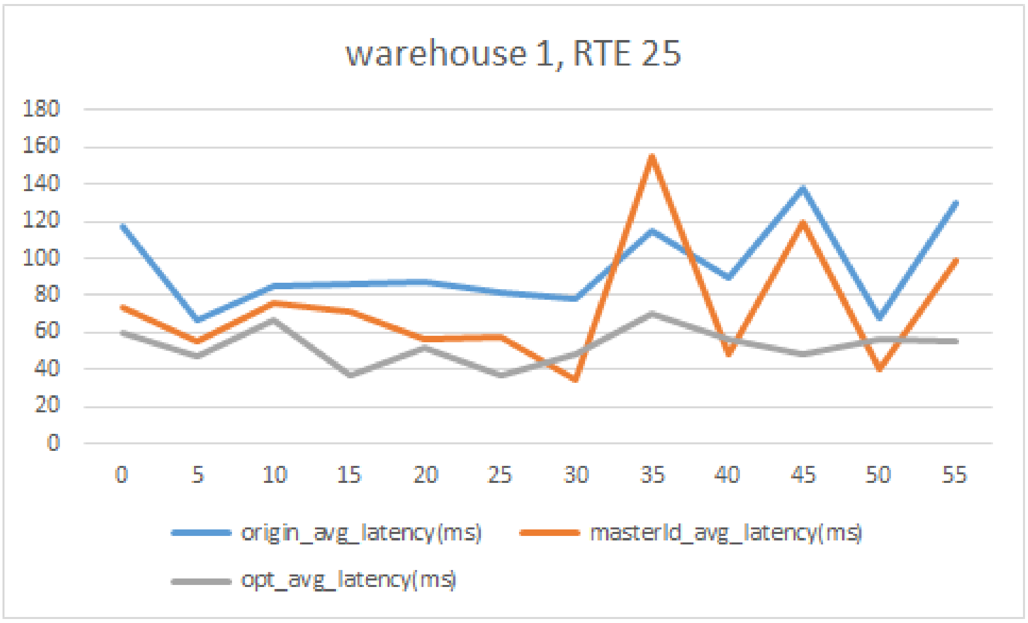
On three different computers in EECS 326

Setting :

partition: 3, server: 3, client: 1, TPCC dataset

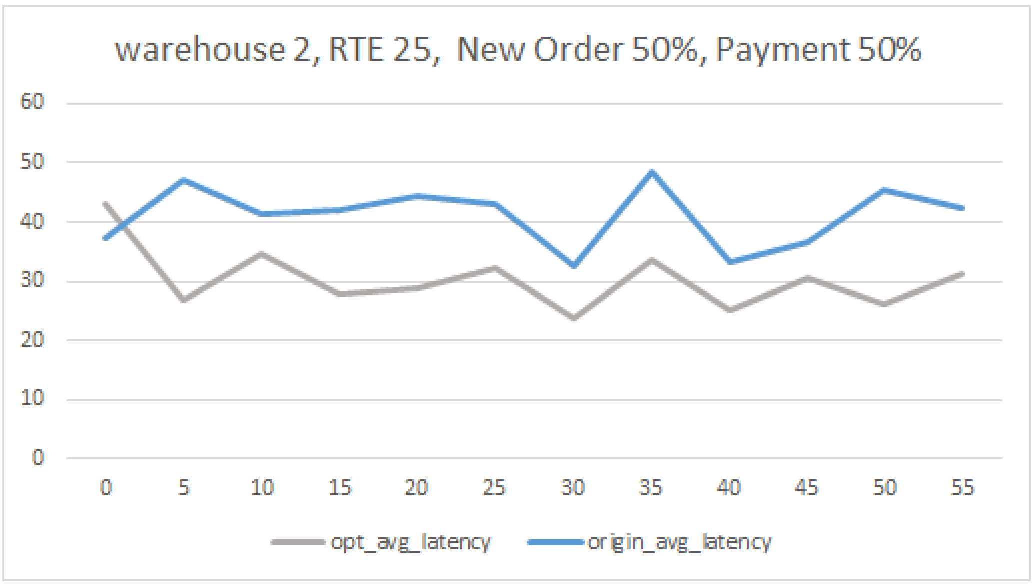
Controlled variable: # of warehouses

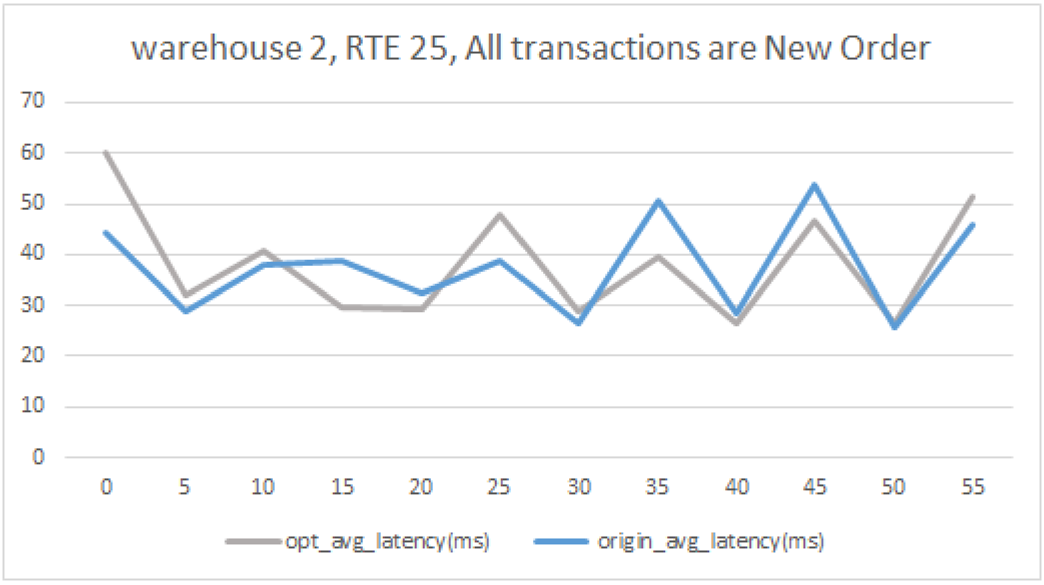


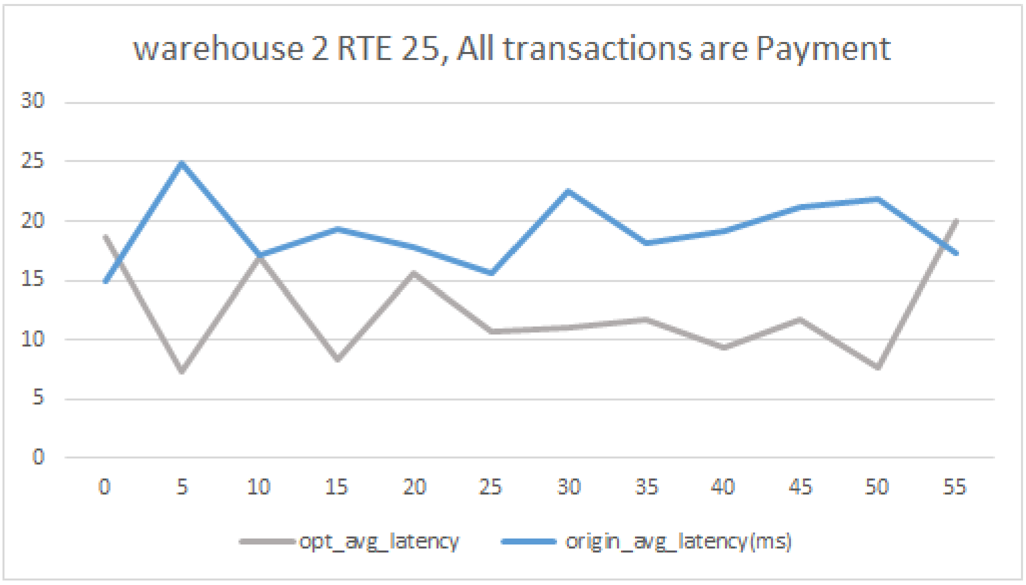


- When number of warehouses is lesser, the locality of dataset increases. Hit rate of global cache also increases, so we can save more I/O time. As a result, decrease of latency is more obvious.
- We can observe a high peak at 35s in orange line in second graph. Because master server may run slower than others, increasing the latency. But on average, adding master server leads to decrease of latency.

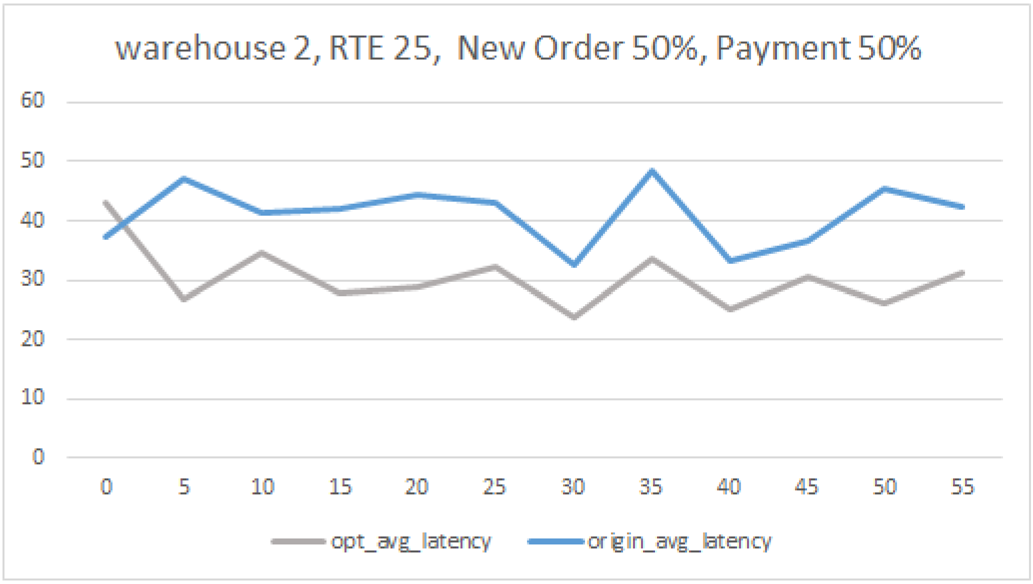
Controlled variable: Type of Transactions

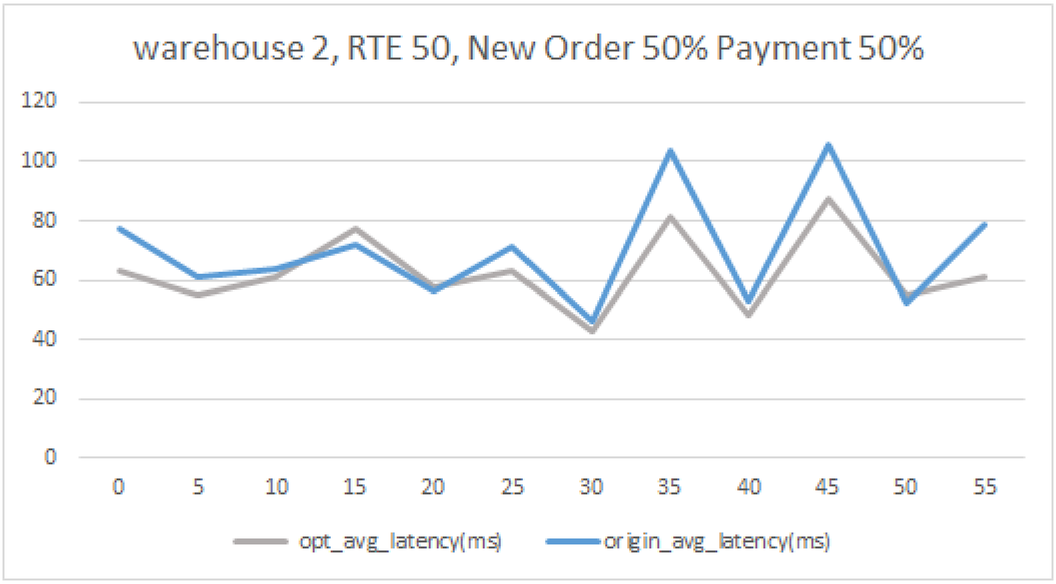






Controlled variable: # of RTEs





- When there are more RTEs, the improvement of latency is lesser.

- **Reason:** When there are more RTEs, there will be more threads to execute stored procedure, which makes the master server more unlikely to be the first one to finish.

  And there's a scene might happen: some active nodes(but not master) have finished the stored procedure before master but still need to wait for the end of master's execution.

  So the optimization of choosing master server may increase latency , which offsets the improvement made by global cache.