

# Team 17

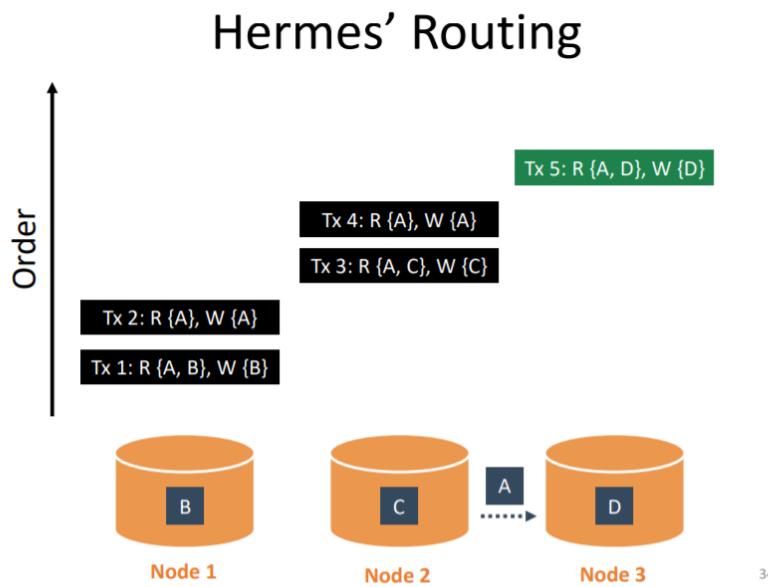
107060008張承勛

107062123鄭宇軒

107062315黃偉哲

## Idea

本次作業我們主要針對hot counter的workload做優化，在原本hermes的routing algorithm下，會將每個transaction放在不同的partition去找出放在哪可以有最少的remote read，但在此情況下，會有資料需要在不同partition之間搬動的情形，



因此我們做了replicate和reorder這兩種優化，為了避免資料頻繁搬動，我們針對最常被存取的hot record做replication，但在replicate的情況下，為了確保每個partition上的資料一致，當有被replicate的資料被修改時，其他partition也要做一樣的修改，如此一來會浪費不必要的時間，而將transaction做reorder可以避免這個情形且同時保持資料正確性，只要將有修改hot record的transaction放在做replicate的transaction之前就可以確保每個partition上的hot record都是最新版本，以維持資料正確性。

## Implementation

針對以上replicate和reorder的優化，我們的algorithm分成4個步驟：

1. 將transaction batch的所有transaction的readset和writeset都看過一遍，並把每個record被存取的次數計下來，而被存取較多次的record我們就先將它定義為hot record
2. 在所有transaction中去檢查此transaction的writeset中是否有包含hot record，如果有的話代表此為修改hot record的tx，並先將這些找到的tx放進T-graph，如此一來就可以讓這些tx先執行，以達成reorder的效果

3. 新增一個負責做replication的transaction, 並將此tx放到每個partition做成TxNode, 再將這些TxNode放進T-graph, 對於每一個做replicate的TxNode, 我們會讓他牽ReadEdge到每個hot record, 代表複製每個hot record到所有partition, 以達成replication的效果, 而此replicate的步驟放在第2步之後是為了把最新版本的hot record複製到每個partition, 來確保資料的正確性。

4. 最後的步驟就是把剩下的tx(沒有修改hot record的tx)放到T-graph, 讓這些transaction在最後才被執行。

## Experiment

- Environment

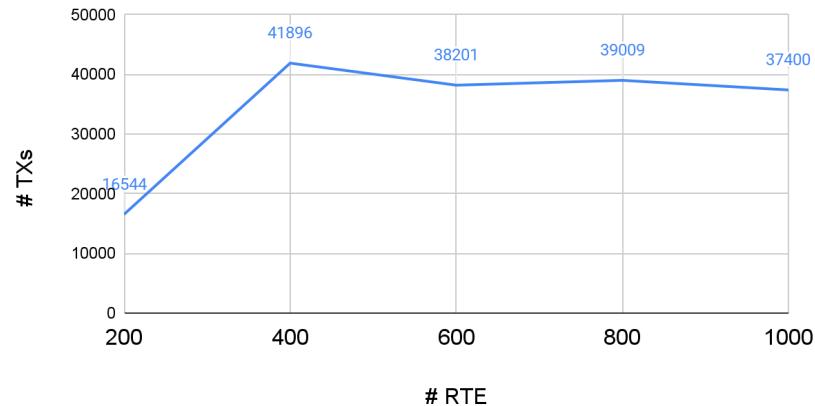
我們使用了周志遠教授的叢集電腦來完成以下實驗

OS	CentOS 7.9.2009
CPU	2 * (Intel Xeon Silver 4110 CPU @ 2.10GHz) per node
Memory	141 GB per node
Connection between nodes	InfiniBand
Configuration	Node 1: Client Node 2: Server 0 Node 3: Server 1 Node 4: Sequencer

- Tunable parameters

為了使實驗能夠有更高的準確性，減少實驗誤差所帶來的影響，應先透過調整參數來使DBMS在機器上發揮最大的效能。而在本次的project, 助教推薦了3項最主要可以調的properties, 我們便依序找到其最適合實驗機器的數值。首先是RTE的數量。藉由實驗可以看出在RTE為400時表現最好，因此在接下來的實驗我們都將RTE設為400。

Hot Counter Workload, benchmark interval = 2 mins



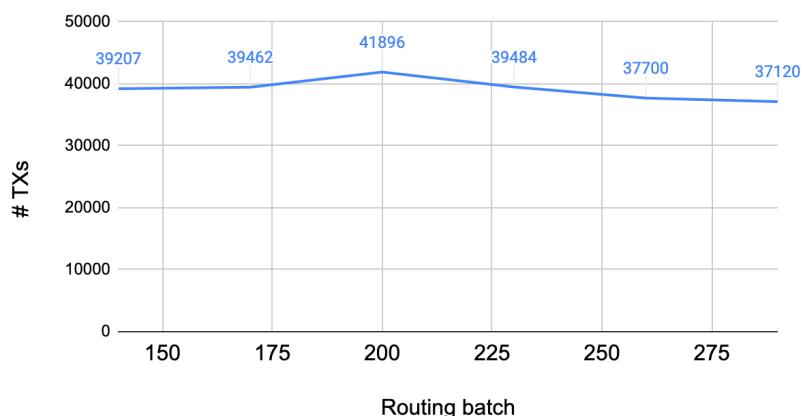
再來是batch size的大小。藉由實驗可以看出在batch size為50時表現最好，因此在接下來的實驗我們都將batch size設為50。

Hot Counter Workload, benchmark interval = 2 mins



最後是routing batch的大小。藉由實驗可以看出在routing batch為200時表現最好，因此在接下來的實驗我們都將routing batch設為200。

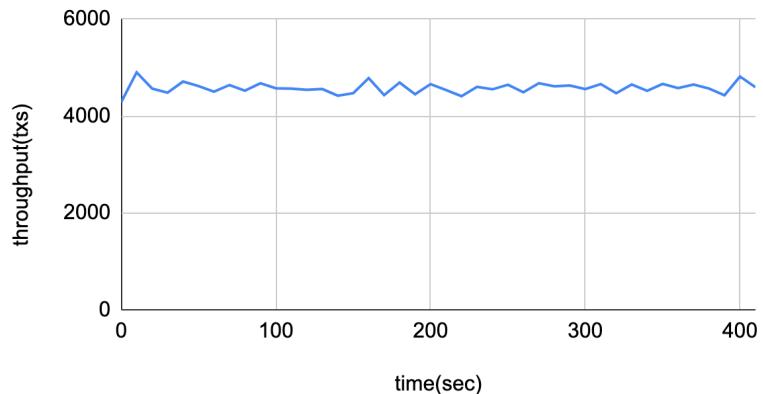
Hot Counter Workload, benchmark interval = 2 mins



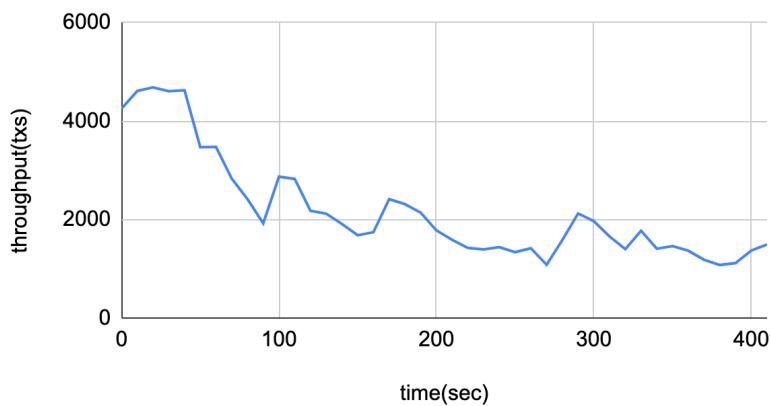
- Hotspot

調整完參數後便可以開始進行實驗。我們依序了三個benchmark，每個皆是先跑原版的，再跑我們優化後的版本。

Hotspot, origin

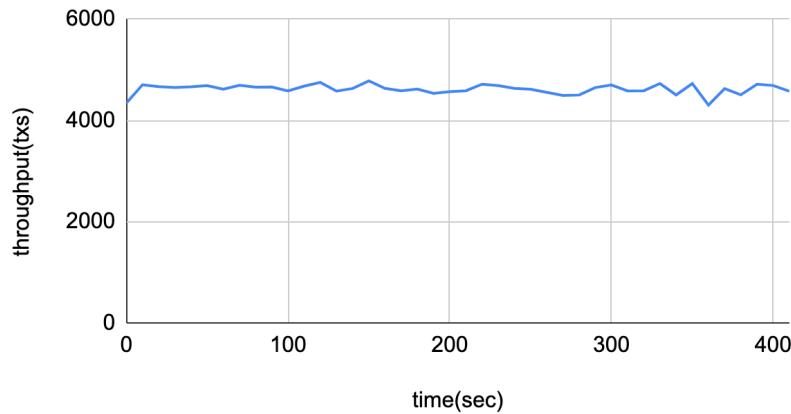


Hotspot, after modified

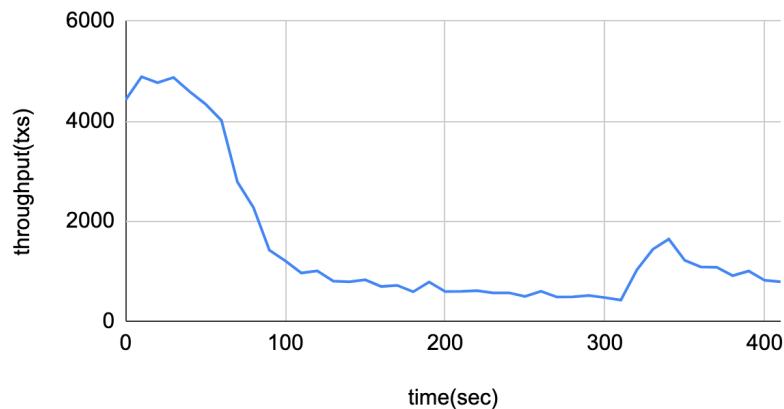


- Google

Google, origin

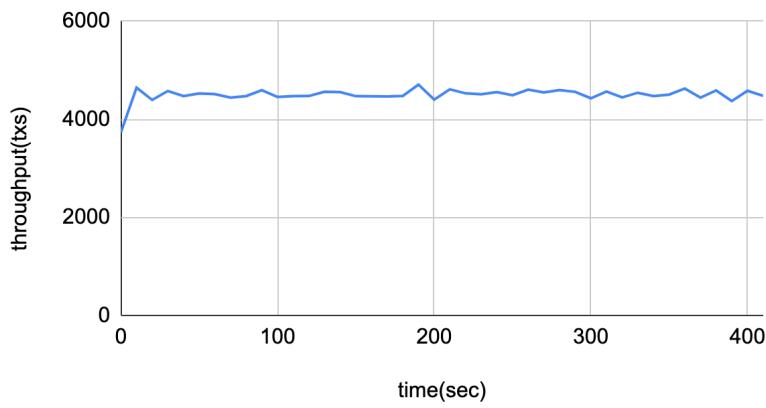


Google, after modified

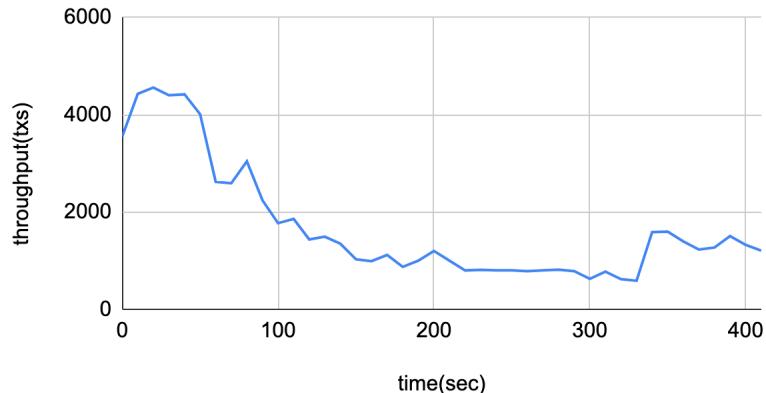


- Hot counter

Hot counter, origin



Hot counter, after modified



## Analyze

由上面的實驗數據可以很明顯的看出我們改完演算法後的結果反而效能變差了，且趨勢幾乎一樣。根據我們的判斷，理由應是Hermes是以Calvin做為分散式儲存系統的基礎；其中，Calvin要求cluster中的每個node須為「shared-nothing」。然而，我們用來跑實驗的cluster是用網路檔案系統(NFS)的方式來把資料串在一起的，也就是說不管是

partition 0還是partition 1, 資料其實都存在Node 1的硬碟裡, 然後再透過InfiniBand傳給Server 0或Server 1。我們演算法優化的目標就是要減少資料藉由網路搬動, 然而在如此的實驗環境之下反倒變成了即使是local read, server仍要藉由網路去Node 1拿資料, 而我們的演算法又多了reorder與replicate的overhead, 因此理所當然的會表現的比原本的版本還差。由下圖可以看出, db0與db1確實都存在同一顆硬碟裡。

```
[didwdidw@art1 db0]$ df -h /home/didwdidw/db0
檔案系統          容量  已用  可用  已用%  掛載點
/dev/mapper/centos-home  1.5T  1.2T  242G  83%  /home
[didwdidw@art1 db0]$ 
[didwdidw@art1 db0]$ 
[didwdidw@art1 db0]$ df -h /home/didwdidw/db1
檔案系統          容量  已用  可用  已用%  掛載點
/dev/mapper/centos-home  1.5T  1.2T  242G  83%  /home
```

## Conclusion

綜上所述, 本次實驗結果不如意暫時可以歸咎於實驗環境不適合。若改成將實驗放在本地跑也會發生類似的問題, 資料都放在同一顆硬碟, 即使是remote read也不用藉由網路去拿資料, 沒辦法體現出local read與remote read的效能差距, 因此我們的演算法的優勢就無法發揮出來。若要克服此問題, 可以改成去租雲端運算的機器來跑, 或是待疫情稍緩, 我們同組的三人再藉由內網連線來共同驗證我們演算法的效能。