國立政治大學資訊科學系

Department of Computer Science
National Chengchi University

碩士論文

Master's Thesis

私有區塊鏈上的兩步驟拜占庭共識演算法

A Simple Two-Step Byzantine Fault Tolerance in Private Blockchains

研究生: 蘇鑑微

指導教授：郭桐惟 博士

中華民國 一零八年 九月

September, 2019

DOI:10.6814/NCCU201901225

誌謝

兩年的碩士生涯在此將要告一個段落，首先要誠摯的感謝我的指導教授郭桐惟老師，在專業領域上，給我許多寶貴的建議；在求知路上，教導我如何發現問題、解決問題。從老師的身上我才體認到學習專業知識固然重要，但求知的精神與做事態度更為重要，真的非常感謝老師的用心教導。在此也要感謝口試委員陳恭老師及陳昶吾老師，百忙之中撥空審閱此論文，並於口試時提供許多寶貴的意見，使得本論文得以更加嚴謹及完整。

感謝 Charles 協助教導我雲端系統的架設，謝謝俊偉學長在當兵時依舊撥空指導我實作區塊鏈系統。同時也要感謝實驗室的夥伴，堯祺、友謙、瑞祥、筆翔、奕寧、海坤、清昱豐富我在實驗室的碩士生活。謝謝承軒、元慧、冠博、元芙以及身旁的所有的朋友，謝謝你們包容我的一切，在我低落的時期給予我很大的支持與鼓勵，也感謝你們陪伴我度過這些年快樂的生活。

感謝政大資科讓我加入這個大家庭，讓我在政大認識許多有趣的朋友，能夠成為其中的一份子我感到無比驕傲與感恩。最後感謝父母為我所做的犧牲與苦心栽培讓我無後顧之憂的完成學業，希望我的努力及成長能讓你們感到欣慰及驕傲。

DOI:10.6814/NCCU201901225

摘要

區塊鏈技術至今已發展十多年歷程。區塊鏈應用也從一開始數位貨幣衍伸生出多樣化的應用與服務。區塊鏈是一種分散式帳本技術，帳本內容是由多個網路節點共同維護，而非受到單一節點所控制。為了確保安全性，系統內所有節點需要擁有相同帳本資料; 換句話說，節點間須對帳本內容達成共識，而達成共識的方法稱共識演算法。為因應不同的應用情境，區塊鏈分為公有鏈與私有鏈。公有鏈對所有人皆開放，而私有鏈只開放特定人群或企業加入。本篇論文針對私有鏈上的共識演算法進行探討及研究。一般而言，私有鏈共識演算法需要三個步驟的訊息交流，才能確保共識結果是正確的。我們設計了一個兩步驟的共識演算法-TwoStepBFT。此演算法能夠容許拜占庭節點錯誤，且保有安全性及活性。為了架設大規模節點的私有鏈，我們整合了多樣的自動化雲端部署套件，此套件幫助我們在雲端平台上自動產生多個節點。實驗結果顯示我們的方法在一百個節點依舊能有 300TPS 的共識效率。關鍵字: 區塊鏈、共識演算法、拜占庭容錯。

DOI:10.6814/NCCU201901225

Abstract

Blockchains have been developed for more than a decade. The very first application of blockchains is digital currency. Within a decade, blockchains have found applications in various fields. A blockchain can be viewed as a distributed ledger, whose content is maintained by multiple network nodes rather than by a single node. In order to ensure correctness, all nodes in the blockchain need to have the same content of the ledger. In other words, all nodes must reach consensus on the content of the blockchain. Toward this goal, all nodes must execute the same protocol that specifies the rules of adding new blocks to Blockchain. Such a protocol is called a consensus algorithm. There are two types of blockchains, public blockchains and private blockchains. Public blockchains have no access control, and is open to everyone. On the other hand, private blockchains can only be accessed by admitted nodes. This thesis focuses on the design and analysis of consensus algorithms for private blockchains. In general, consensus algorithms for private blockchains involve three steps of information exchange to ensure correctness. In this thesis, we design a two-step Byzantine tolerant consensus algorithm, TwoStepBFT. In order to evaluate the performance of TwoStepBFT

DOI:10.6814/NCCU201901225

on a 100-node blockchain, we integrate a variety of automated tools to deploy blockchains. Experimental results have shown that our consensus algorithm can reach a throughput of 300 TPS on a 100-node Blockchain.

Keywords: Blockchain, Consensus Algorithm, Byzantine Fault Tolerance.

DOI:10.6814/NCCU201901225

目錄

DOI:10.6814/NCCU201901225

DOI:10.6814/NCCU201901225

圖目錄

DOI:10.6814/NCCU201901225

第一章　緒論

從 1969 年網際網路誕生以來，眾多數位化的商品與服務不斷推陳出新，進而大幅改變了人們的生活習慣。然而，貨幣的數位化卻是直到近年來區塊鏈的誕生才得以真正的被實現。良好的數位貨幣系統需要具備下列三特性:（一）交易紀錄難以被竄改、(二）避免雙花（Double-spending)、(三）避免由某個單一角色控制該數位貨幣系統，下列分別介紹此三項特性。

（一）交易紀錄難以被竄改: 交易紀錄與網路訊息一樣，只是單純的位元碼。如何確定使用者擁有某數位貨幣資產，並保證交易紀錄無法遭人輕易竄改，是非常重要的問題。

（二）避免雙花: 在數位貨幣系統中，可能存在同一筆數位資產，因為不當操作而被重複使用的情況，這類問題也被稱之為「雙花問題」。

（三）避免由某個單一角色控制該數位貨幣系統: 如果系統中存有一個中央集權的角色，例如: 中央政府或金融企業等，來協助交易的清算及交易記錄的保存，那麼前兩項特性是容易被滿足的。這類做法雖然頗具效率，但是賦予單一集權角色過大權力卻可能導致不好的結果。例如: 國家能夠透過加印鈔票來稀釋鈔票的價值，進而控制金融市場；遊戲公司也可能透過發行新的虛擬點數，讓原先的虛擬點數一文不值。

區塊鏈便是能同時滿足難以被竄改、無法重複支付且去除中央集權的安全系統。區塊鏈是一種基於分散式系統（Distributed system）的節點備份技術，系統中每一位參與者為稱之為節點。節點間權力相互平等，不需透過中央集權角色來

DOI:10.6814/NCCU201901225

管理系統運作。每一個節點都各自擁有一份帳本資料，節點間會相互溝通合作，來確保所有節點的帳本資料是一致的。因此，攻擊者難以透過攻擊單一節點來癱瘓整個系統運作。在區塊鏈系統裡，所有節點所看到的帳本內容是相同的，帳本內交易順序相同，如此一來節點就能根據帳本內的交易順序了解交易是否生效。區塊鏈是由連續的區塊所組成，區塊內記錄了節點們所提交的交易內容。區塊之間通過保存哈希值（Hash）實現鏈接。為了增加惡意攻擊難度，區塊內會包含該區塊產生的時間戳（Timestamp）。為了防止惡意節點任意更動過去的帳本資料，每一個區塊會包含一個獨一無二的哈希值，該值是根據前一個區塊的內容所產生的。惡意攻擊者無法任意更動前面的區塊內容。一旦區塊內容改變，哈希值將無法符合，這樣的設計使得區塊鏈變得更加安全。第一個區塊稱為創世區塊（Genesis block），創世區塊是所有區塊中，唯一不需要包含前一個區塊哈希值的特例區塊。

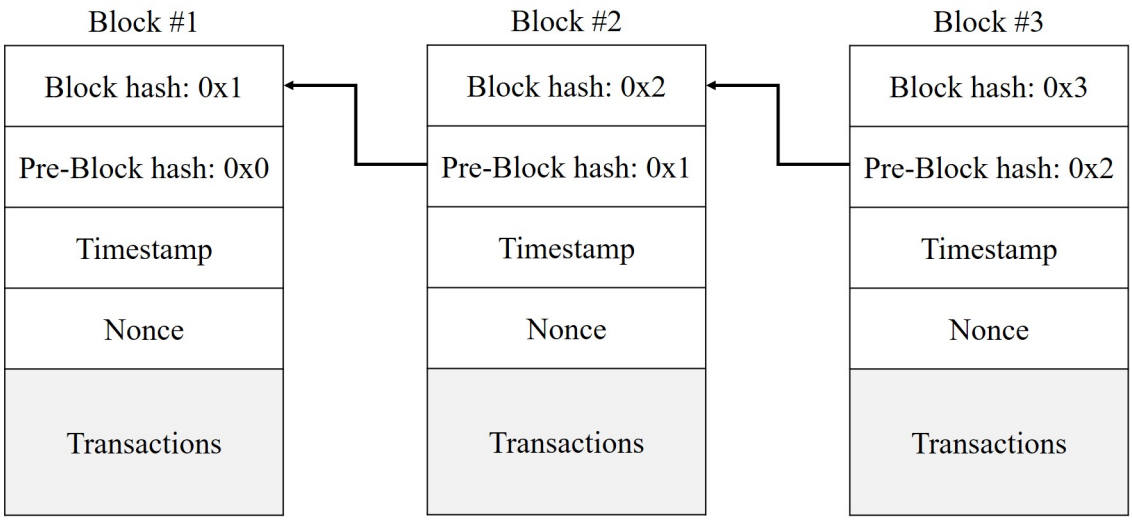

圖 1.1: 區塊鏈示意圖。

因應不同用戶需求及應用場景，區塊鏈系統架構及規模也不盡相同。區塊鏈主要類型可分為公有鏈以及私有鏈。公有鏈的應用如比特幣 [17] 或以太幣 [4] 等，其特性為任何人都能參與該區塊鏈，加入公有鏈不需要任何人授權，可以隨時自由加入或退出。不同於公有鏈，私有鏈大多是指多個機構共同參與管理的區塊鏈，每個機構執行一個或多個節點，參與私有鏈的節點須有嚴格規範。私有鏈上知名

DOI:10.6814/NCCU201901225

例子包含 Facebook Libra [2]、R3 Corda [1] 等。私有鏈中的資料只允許系統內不同的機構進行讀寫，並且共同記錄及備份資料。由於參與私有鏈的節點是有限制和可控制的，因此私有鏈往往可以有較快的交易速度、更好的隱私保護且不容易被惡意攻擊。本論文將聚焦於私有鏈上。因為區塊鏈是一個分散式架構，節點間平權管理，所以需要透過運行相同的「共識協議」來維護相同的區塊順序。這類的共識協議則稱為共識演算法。公有鏈上的共識演算法大多採工作量證明（PoW）來達成共識。PoW（Proof-of-Work）透過競賽解題，第一個解出題目的節點將獲得廣播區塊的權力，共識過程中大部分的運算效能都會花在這個解題步驟（俗稱挖礦），因此區塊產出時間會被拉長，導致共識效率低落。然而私有鏈因為節點需經授權才可加入，這類演算法大多透過投票來選擇區塊的廣播者，透過輪流廣播來維持區塊鏈內的平權治理。

私有鏈上通常運行 BFT 共識演算法，BFT 是拜占庭容錯（Byzantine Fault-Tolerant）的縮寫。在拜占庭容錯的區塊鏈上，即使系統中部分節點當機或存在惡意節點情況下，區塊鏈依舊具備安全性（Safety）與活性（Liveness）等性質。安全性能夠保證區塊鏈上每個參與區塊鏈系統的節點，都擁有相同且排序一致的帳本內容。活性則讓區塊鏈是一個永續的系統，能夠容忍錯誤發生且避免因為單一節點故障而導致系統停擺。目前已經有許多 BFT 共識演算法，傳統的 BFT 演算法通常需要多個回合才能達成共識。每一個回合會有一個節點擔任提議者（Proposer）。一般來說，一個回合需要執行三個步驟: 第一個步驟的目的在於讓提議者廣播其提議，若該提議者廣播多份提議，則區塊鏈可能發生分叉（Fork）進而導致內容衝突。這讓攻擊者可能重複花費同一筆金錢；第二個步驟的目的即為讓所有節點檢查該提議者是否只有廣播一份提議。若提議者只有廣播一份提議，在第三個步驟中，節點便可以開始投票（將收到的提議廣播給其他節點），若一個節點蒐集到足夠多的同意票，該節點便可以確認該提議為最終共識結果；第三個步驟的目的是為了克服訊息傳輸延遲，知名的 BFT 演算法 PBFT [6] 便是根據上述三個步驟設計。

DOI:10.6814/NCCU201901225

過去也有少部分的 BFT 演算法在一個回合中只需要兩個步驟，例子包含 FaB [3]、Zyzzyva [14]、SBFT [16]、Hydrachain [19]。然而，FaB 與 Zyzzyva 已被指出無法保證活性，換句話說，共識演算法可能永遠無法達成共識。另一方面，Hydrachain 也被指出無法保證安全性，也就是說，不同的正常節點可能會有不同的共識節果，這在區塊鏈上及代表分岔。雖然 SBFT 能夠保證安全性與活性，但是 SBFT 在節點發生故障或是網路延遲過大的情況下，會改用類似 PBFT 的三步驟設計。因此，在系統狀況良好時，SBFT 每回合只需要執行兩個步驟。但是在系統狀況不良時，SBFT 每回合仍需要執行三個步驟。本論文探討的議題為一個新的兩回合拜占庭共識演算法。更明確的說我們希望在不犧牲安全性及活性情況下，設計一個每回合只需要兩個步驟的 BFT 演算法。重要的是，該 BFT 演算法即使在系統狀況不佳時，每個回合仍只需要兩個步驟。以下是本篇論本的研究目標:

- 設計一個兩回合的共識演算法-TwoStepBFT 並保證安全性及活性。
- 將 TwoStepBFT 演算法實作於以太坊（Geth version:1.67）上，並保留以太坊上其他功能。
- 整合多樣化雲端部署套件，快速在雲端伺服器上建立多台環境一致的機器作為私有鏈共識節點。
- 自動化生成以太坊節點，將節點部署模組化，讓系統根據使用者需求，快速搭建該私有鏈。
- 針對不同實驗環境進行效能測試，進行共識效率比較。

DOI:10.6814/NCCU201901225

第二章　系統模型

我們在 2.1 節描述網路模型與系統假設並在 2.2 節描述本論文共識假設與論文常見符號定義

2.1　網路模型與系統假設

早在 1985 年 Fische 等人，提出了 FLP 不可能證明 [11]，該論文證明在一個完全異步的網路系統中，假使存在節點失效（即便只有一個節點失效），不存在一個可以達成共識的演算法。所以我們將系統假設為部分同步的網路延（Partially synchronous）模型。更明確地說，若 $D(t)$ 代表一個在時間 t 傳遞訊息所會遭遇的延遲時間，則 $D(t)$ 成長的速度不可能永遠大於 t 的成長速度。部分同步的網路延遲模型具有以下的性質: 若是每個回合的長度都是上個回合長度的兩倍，則一定存在一個回合 i，在回合 i 中前半段傳遞的訊息都能在回合 i 的後半段收到。因此，在我們設計的 BFT 演算法中，每個回合的長度會是前一個回合長度的兩倍。

2.2　共識假設與定義

我們假設 n 為共識問題中的參與共識的總節點數量（包含故障節點的數量），f 則為錯誤節點上限。Martin 與 Alvisi 在 Fast Byzantine Consensus [16] 該論文證明了，如果要設計一個拜占庭容錯共識演算法，且每個回合只能執行兩個步驟，為了確保系統共識結果一致，n 必須至少大於 $5f$。因此，我們假設 $n = 5f + 1$。下列我們定義一些共識演算法中的基本元素。

DOI:10.6814/NCCU201901225

- 區塊: 以太坊的基本區塊包含 Header 與交易以下會以 b 表示。
- 節點: 參與共識演算法的個別節點以 u。
- 高度: 共識的區塊高度以 h 表示，區塊鏈會從 $h = 0$ 開始進行共識。
- 回合: 每次共識回合則用 r 表示，每回合會從 $r = 0$ 開始進行共識。
- 提議：在每個回合開始時我們會從所有參與共識的節點中，選出一位 Proposer 進行廣播，由它提出當回合共識的區塊 b。我們稱 Proposer 其所廣播的提議為 Proposal，以 p 表示。我們以 $p(h, r, u, b)$ 表示此 Proposal 是在第 h 的高度、第 r 的回合、來自於 Proposer u 且包含一個區塊 b。
- Vote：Vote 為一種訊息型態，在共識期間節點之間會互相廣播 Vote。我們以 $v(h, r, p, u, b)$ 表示一張 Vote，代表此 Vote 在第 h 的高度、在第 r 的回合、投給提議 p 且來自於 u 節點。在演算法系統裡，Vote 會紀錄區塊 b 的 Hash 值而非整個區塊結構。
- Lockset：節點會將 Votes 儲存於 Lockset 以進行將來的共識。我們以 $l(h, r, u)$ 表示一個 Lockset 在第 h 的高度、在第 r 的回，來自於 u 節點。且所有在此 $l(h, r, u)$ 裡，每一張收到的 $v(h, r, p, u, b)$ 中，h 與 r 都是必須相同，但這裡 Vote 所投的區塊卻不一定需要相同。一個 Lockset 需要包含大於 $\frac{4}{5}\ n$ 的 Votes 才能成為一個合法的 Lockset ，且 Vote 必須來自於不同節點。
- Timeout: Timeout 定義了段一個時間長度，在演算法裡 Timeout 分成了接收 Proposal 的時間閘 $TOvote$ 與接收 Vote 的時間閘 $TOcommit$，在我們的演算法裡 $TOvote = TOcommit$。

DOI:10.6814/NCCU201901225

第三章　共識演算法

3.1 共識流程

我們的目標是設計一個每個回合只須執行兩個步驟的 BFT 演算法。更明確地說，在一個步驟中，一個節點可以依序做以下事情: （一）廣播訊息。（二）接收訊息。（三）做本地端的運算（Local Computation）。在我們提出的 BFT 演算法中，每個回合包含兩個步驟: 在第一個步驟中，共識演算法會用依序循環（Round-Robin）的方法在每回合選出一名提議者。只有該名提議者可以在該回合廣播提議。在第二個步驟中，所有節點會投票（廣播訊息）和收集其他節點的選票（接收訊息）。我們的演算法受到 MSIG-BFT [7] 的啟發，MSIG-BFT 是一個每回合需要執行三個步驟的共識演算法。與 MSIG-BFT 演算法一樣，如果網絡狀況不佳，提議者有可能會被禁止廣播提議。下面將詳細描述這兩個步驟。

3.1.1 Propose

第一步: 在第一個步驟中的主要目標為讓提議者廣播提議。我們稱這個廣播的訊息為 Proposal 訊息。若 p 為一個 Proposal 訊息，則我們用 $p.B$ 代表該 Proposal 訊息中的提議值。舉例來說，在區塊鏈中，$p.B$ 可能是該提議者產生的區塊。另外我們用 $p.R$ 代表該 Proposal 訊息被產生的回合數。舉例來說，若 p 是第二回合的提議者廣播的 Proposal 訊息，則 $p.R = 2$。第一步驟的虛擬碼如圖 Algorithm 1 所示。我們會在解釋完第二個步驟後詳細討論第一個步驟。

DOI:10.6814/NCCU201901225

Algorithm 1 Propose: 從節點 u 來觀察

Input: 節點 u 具有初始值 b_{in}，以及上一回合的票集合 $lockset(r-1)$
ldr = 在 r 回合的廣播提議者
if u=ldr **then**
  製作一個提案 p
  $p.R \leftarrow r$
  // 確認 $p.B$ 並且將 p 廣播給所有節點
  if r=*1* **then**
    $p.B \leftarrow b_{in}$ 並且將 p 廣播給所有節點
  end
  else if $|lockset(r-1)| \geq 4f$+*1* **then**
    if $\exists$ 一個 b，且 $b \neq \varnothing$ 且 $|\{v|v \in lockset(r-1), v.B = b\}| \geq 2f$+*1* **then**
      b 可以為任何滿足區塊限制的值
      $p.B \leftarrow b$ 並且將 p 廣播給所有節點
    end
    else
      $p.B \leftarrow b_{in}$ 並且將 p 廣播給所有節點
    end
  end
end

3.1.2 Vote

第二步: 第二個步驟的主要目標為讓所有節點對接收到的 Proposal 訊息 p 投票，並在收到足夠多選票的情況下，提交（Commit）提議值（在區塊鏈的應用中，提交代表將區塊鏈接在區塊鏈的尾端）。為了實現這個目標，一旦節點 u 收到 Proposal 訊息 p，則節點 u 首先驗證提議值 $p.B$ 的正確性。舉例來說，在區塊鏈的應用中，節點 u 會檢查提議值 $p.B$ 是否為一個合法的區塊。如果提議值 $p.B$ 通過驗證，則 u 會廣播一個包含 $p.B$ 的投票訊息（選票）。每個節點在一回合中最多只能廣播一個投票訊息。如果節點在某個預定的時間 $TOcommit$ 內收到 $4f+1$ 個投票訊息，並且這些 $4f+1$ 的投票訊息包含相同的非空（Non-empty）提議值 b，則節點提交提議值 b。另一方面，如果節點 u 在時間 $TOcommit$ 到之前無法提交，那麼節點 u 需要紀錄這個回合廣播的投票訊息中的候選值 b，並進入下個回合。更明確地說，若 b 是在第 i 個回合中，由 u 廣播的投票訊息中的候選值，則在第 $i+1$ 個回合中，如果 u 在另一個的時間 $TOvote$ 到期之前，仍然無法收到有效的 Proposal 訊息，則 u 廣播的投票訊息會包含提議值 b。值得注意的是，在第一個回

DOI:10.6814/NCCU201901225

合時，如果 u 在 $TOvote$ 到期之前無法收到有效的 Proposal 訊息，則 u 廣播的投票訊息會包含空提議值 $\varnothing$。每當進入下一回合時，上述兩個預定時間（$TOcommit$ 與 $TOvote$）的長度會加倍。若 v 為一個投票訊息，我們使用 $p.R$ 和 $p.B$ 表示 v 被產生的回合數和 v 中包含的提議值。第二步驟的虛擬碼如圖 Algorithm 2 所示。

Algorithm 2 Vote: 從節點 u 來觀察

```
// 確認 v.B 並且將 v 廣播給所有節點
if 在 TOvote 到期之前，u 在第 r 回合收到一個合法的 Proposal p then
    v.B ← p.B 並且將 v 廣播給所有節點
end
else
    if r=1 then
        v.B ← ∅
    end
    else
        v' ← u 在 r-1 回合廣播的投票訊息
        v.B ← v'.B
    end
    廣播 v
end
if 在 TOcommit 到期之前，u 在第 r 回合收到 4f + 1 張投票訊息且包含一樣的 b 且 b ≠ ∅ then
    Commit b
end
else
    進到 r + 1 回合
end
```

3.1.3 Commit

Commit 階段是各節點根據上一個步驟的投票結果，判斷是否可以進行提交區塊如果存在 $lockset(h,r) = b$，則提交區塊 b 至區塊鏈上完成高度 h 的共識。如果 $lockset(h,r) = b$ 不存在，則進入第一階段 Propose 且進入 $r + 1$ 回合。

3.2 Timeout

共識中的 Timeout 會隨著 R 的增加進行指數型成長其成長公式為:

$Timeout_{X,R} = Timeout_X * TimeoutFactor^R$

DOI:10.6814/NCCU201901225

其中 X 為 $TOvote$ 或 $TOcommit$，$Timeout$ 的起始值其值為 5 秒，$TimeoutFactor$ 預設則為 2。

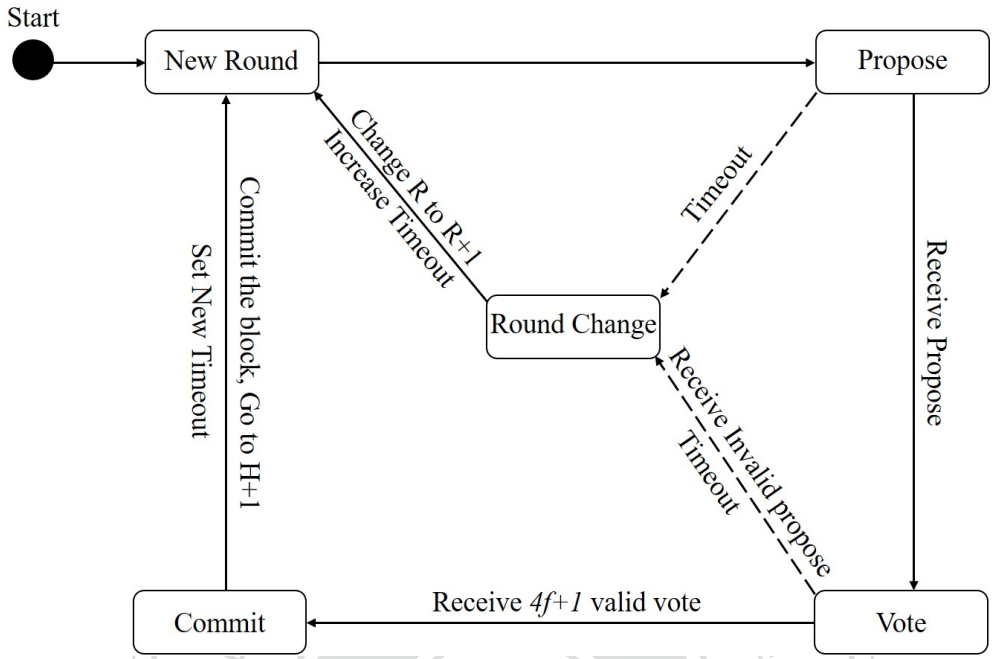

圖 3.1: TwoStepBFT 演算法流程圖。

DOI:10.6814/NCCU201901225

第四章　演算法分析

一個好的共識演算法必須具備安全性（Safety）與活性（Liveness），才能讓該系統保持內容一致，並且保證安全且永不出錯。

4.1 安全性（Safety）

共識演算法必須要確保整個網路不會有分支（Fork）產生，以確保區塊鏈的安全性。為了證明我們的演算法能夠確保安全性，我們需要證明以下兩個定理。

- 如果兩個非故障節點 u_1 和 u_2 分別在同一回合 r 中提交值 b_1 和 b_2，則 $b_1 = b_2$。

  證明: 我們使用反證法，首先假設 $b_1 \neq b_2$，如果有一節點 u_1 收到 $4f + 1$ 張提交 b_1 的選票；與 u_1 相同的 u_2 收到 $4f + 1$ 張提交 b_2 的選票。因為扣除掉可能重複投票的錯誤節點數 f，在回合 r 裡至少有 $3f + 1$ 個非故障節點投給 b_1；相同的扣除掉可能重複投票的錯誤節點數 f，在回合 r 裡至少有 $3f + 1$ 個非故障節點投給 b_2。兩個集合共 $6f + 2 > 4f + 1$（非故障節點個數）。非故障節點一回合只會對一個提議投票，故矛盾。

- 一旦有一個非錯誤的節點在 r 回合成功的提交了區塊 b 時，在未來的回合 r'，此 $r' > r$，只有區塊 b 能被提交。

  證明: 假設 N_b 是在第 r 回合中投票給 b 的節點集合。因為 b 是在回合 r 中被提交，代表 $|N_b| \geqq 4f + 1$。假設 G_b 是僅包含非故障節點的 N_b 的子集。

DOI:10.6814/NCCU201901225

因此，$|G_b| \geqq 3f+1$。根據演算法規定，我們能夠觀察到如果 $r+1$ 回合的廣播者擁有 r 回合的有效票集合，那麼此票集合必須至少包含從 G_b 發送的 $2f+1$ 個投票消息。此外，對於每個候選值，假設 $b_1 \neq b_2$，最多只有 $2f$ 張票在票集合中的投票信息有包含 $b2$。因此，如果下回合 $r = r+1$ 的廣播者可以發送提案 p，該 $p.B$ 必須維持是 $b1$。否則，如果 $r+1$ 回合沒有 Proposal，則 Gb 中的所有節點，仍會投票支持他們在第 r 輪投票的值，即 $b1$。在這兩種情況下，Gb 中的所有節點仍然投票對於 $r+1$ 回合中的 b，然後通過歸納進行 Commit。

4.2 活性（**Liveness**）

我們能夠證明演算法可以保證在部分同步模型（Partially Synchronous Model）下，能夠保有的活性。

在部分同步模型的假設裡，我們假設每回合的 Timeout 時間都會加倍。在部分同步模型下，存在一些回合 r。並且這些 $r < r'$ 所有非故障節點都可以在 Timeout 前接收到彼此傳來的訊息。因此，在這些 r 的下一回合 $r+1 < r'$ 回合中，如果廣播者是非故障節點那麼廣播者必須有一個有效的票集合，並且必須有一個有效的 Proposal 消息。然後，所有 $4f+1$ 非故障節點在 r' 回合中投票給 $p.b$。由於所有這些 $4f+1$ 投票消息可以及時接收，所有非故障節點都可以在循環 r' 中提交 $p.b$。

DOI:10.6814/NCCU201901225

第五章　Geth 共識模組程式架構與私有鏈系統部署

在 5.1 將會介紹我們如何修改 Geth（以太坊 go 語言版本）的共識模組，並且介紹共識模組彼此關係。5.2 將會介紹如何將修改後的共識模組放到雲端平台上並有效率的讓這些雲端平台上的機器組成私有鏈。

5.1　Geth 共識模組程式架構

本章節介紹我們如何實作拜占庭容錯共識的架構。我們的程式架構主要參考 NCCU-BFT [18] 與 MSIG-BFT [7]。共識模組底下包含處理訊息的 Message Handler、同步訊息的 Synchronizer、與三個主要做共識的模組，Consensus Manager、Height Manager 與 Round Manager。共識模組間的關係如圖 5.1。各共識模組之間相互負責不同事情。共識模組透過 go 語言來進行實作，模組間參數時常共同管理（透過 go 語言裡 Channels 來傳遞參數），因此需要透過讀寫鎖來控制讀寫流程，才不會造成多個模組同時讀寫情形。節點間訊息傳輸前都需透過橢圓數位簽名演算法進行簽名（ECDSA）[13]，以確保訊息沒有被竄改。

DOI:10.6814/NCCU201901225

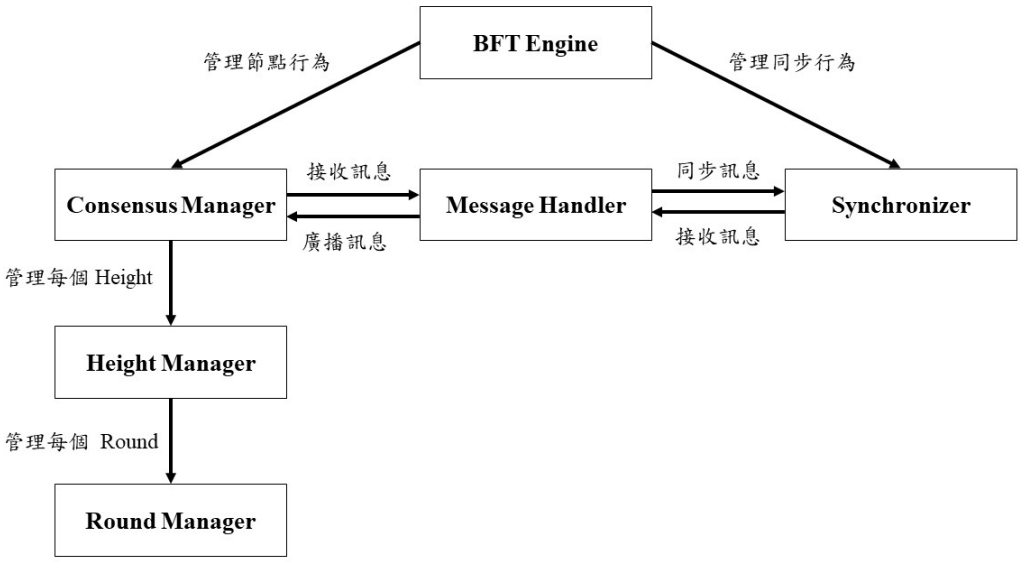

圖 5.1: 共識模組關係圖。

5.1.1 Consensus Manager

Consensus Manager 主要管理節點間連線，並且紀錄參與共識時所需的各式資訊，如: 私有鍵 ID、各節點地址、共識 Timeout 等。並且檢查區塊內的交易是否合法。Consensus Manager 核心技術在於定時的檢查自身狀態是否符合下列三點。(一) 廣播: 檢查是否為該回合的廣播提議者，如果是則打包資料庫裡交易紀錄提出 Proposal; 如果不是則等待 Proposal 被提出。(二) 投票: 檢查是否有收到 Proposal，如果有則檢查是否合法；如果沒收到則對上回合的提議再次投票。(三) 提交: 檢查票集合是否有足夠票數的票據，如果有則提交區塊。如果滿足上述任何一個狀態，則透過呼叫下層 Height Manager，進一步處理共識。同時也要將需要廣播的資訊交給 Message Handler 進行廣播。

5.1.2 Height Manager

此模組主要功能是處理某一個高度的共識，透過管理一或多個 Round Manager 與其他節點溝通。(一) 廣播: 如果節點為該回合的 Proposer 且擁有前一回合票集合，呼叫下層 Round Manager 廣播新的共識提議；如果節點不是此回合的

DOI:10.6814/NCCU201901225

Proposer，等待新的共識提議被提出。（二）投票: 呼叫 Round Manager 幫忙進行投票。（三）提交: 確認高度是否已達成共識，如果還沒則呼叫 Round Manager 進行提交共識。

5.1.3 Round Manager

每個高度可能會需要一到多個回合，最終才能達成共識。因此 Height Manager 可能須要建立多個 Round Manager 來進行共識。Round Manager 會蒐集來自 Height Manager 的訊息（目前共識高度訊息），經過判斷後採取對應動作。（一）廣播: 對所有節點廣播共識提案、（二）投票: 在一個回合裡，Round Manager 只會對一個合法共識提案簽章並廣播，因此 Round Manager 會檢查是否已經對某高度投過票。（三）提交: 將區塊內容同步到自身資料庫，並進入下一個區塊高度。

5.1.4 Message Handler

Message handler 主要負責接收與廣播共識之中需要的訊息，在接收到來自其他節點的訊息時，進行判斷是否交給 Consensus Manager，或是過濾掉重複接受的訊息。

5.1.5 Synchronizer

Synchronizer 主要目的為同步節點間狀態。因為網路間可能存在延遲或訊息也可能丟失，因此可能導致部分節點沒有收到新的共識提議而無法執行後續共識。這時候節點必透過 Synchronizer 先與其他節點索取必要資訊（例如: 沒收到的共識提議）並且同步到最新狀態。

5.2 私有鏈系統部署

TwoStepBFT 共識演算法相較於過去傳統三回合的共識演算法，能在網路傳輸穩定情況下能較快達成共識。我們的目標是測試 TwoStepBFT 在多個節點的私有

DOI:10.6814/NCCU201901225

鏈效率，實驗將在 AWS 雲端伺服器上進行。因此，實驗目標將分為（一）雲端虛擬機器設定: 依照使用者需求，在雲端平台上開啟多台規格一致的機器作為私有鏈節點。（二）私有鏈系統設定: 將雲端平台上的機器依使用者設定參數客製化搭建私有鏈。為了達成上述兩項目標，我們整合了許多自動化的開發套件來協助我們完成實驗。實驗系統架構主要如下圖 5.1 所描述。該系統內含一個 Control Tower，

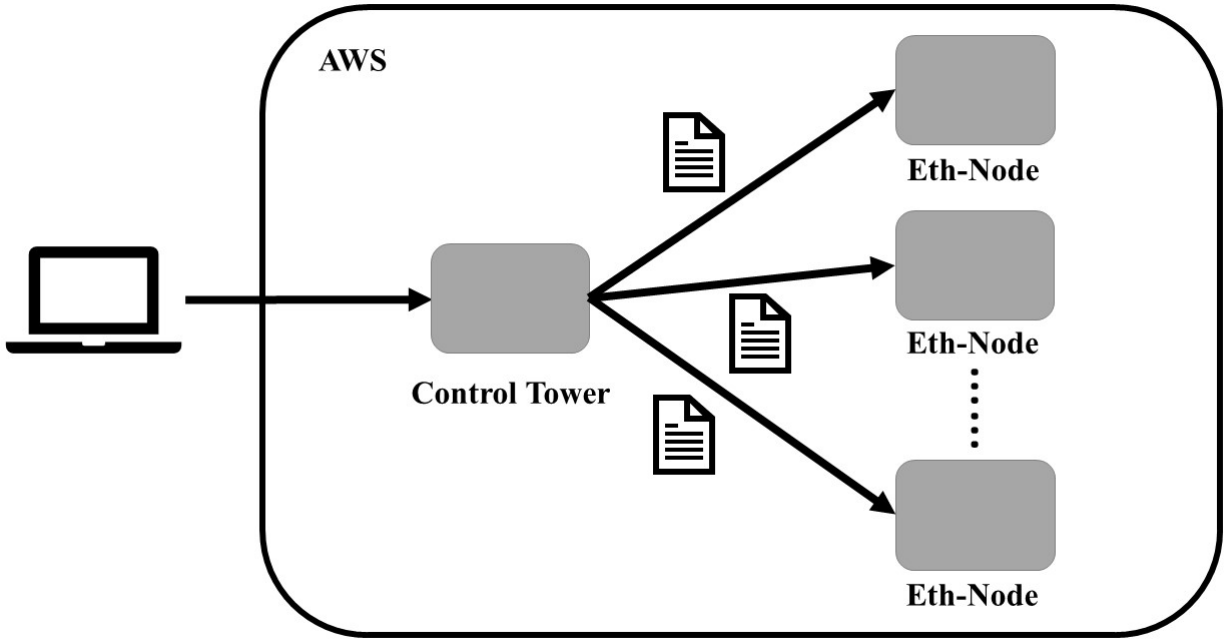

圖 5.2: 實驗系統架構圖。

此台電腦類似實驗的中央控制中心，負責部署實驗環境到其他雲端機器上，並且收集其他機器所產生的實驗數據。依照不同需求，系統內包含數台規格一致機器作為區塊鏈節點，這些節點運行 TwoStepBFT 演算法以達成區塊鏈共識。

5.2.1 雲端虛擬機器設定

為了能夠在雲端平台上，快速開啟多台環境一致的機器，這些機器將成為區塊鏈的節點。我們的步驟為下（一）透過 Packer 將系統所需套件打包成映像檔。該映像檔裡需包含 TwoStepBFT 共識專案以及程式所需的各試工具套件，例如:Go 語言套件、Node.js 等套件，以便後續私有鏈搭建。將所需套件打包成映像檔好處如

DOI:10.6814/NCCU201901225

下。

- 快速的部署：映像檔內已包含所須套件，不需要逐一安裝套件，且也不怕遺漏安裝套件。
- 跨平台且可攜性：不同平台打包出的映像檔，可以獲得相同的運行環境且這個映像檔能夠兼容不同作業系統。

（二）依照使用者需求，透過 Terraform 快速在雲端平台上開啟多台規格一致的機器，並將步驟一產生的映像檔植入這些機器。Terraform 能夠快速將 AWS 環境架設起來，依照使用者所撰寫程式碼搭建出所描述的架構，Terraform 好處是能徹底實現基礎架構即代碼 IaC（Infrastructure as code），利用代碼來配置實驗環境，讓環境安全且方便管理。Terraform 的優點如下。

- 將基礎架構使用語法進行描述，可讓建構計劃像一般程式碼一樣進行版本控管與追蹤。
- Terraform 會自動分析本地端計劃與遠端是否一至，自動化修改從而避免許多可能的人為操作錯誤。

5.2.2 私有鏈系統設定

此小節將會介紹如何將修改後的共識模組放到雲端平台上，並有效率的讓這些雲端平台上的機器搭建出一條私有鏈。為了方便的控制雲端平台上的所有機器，我們使用 Ansible 來管理節點間的行為。Ansible 能夠定義群組，一個群組裡能夠管理多個 IP，透過此方法我們能為雲端平台上的機器定義出 Control Tower 與 Eth-nodes。有了 Ansible 後，我們只需要操作 Control Tower 就能同時控制所有 Eth-nodes，不需登入所有 Eth-nodes 機器進行操作。Ansible 的優點如下。

- 輕量級套件，無需在客戶端安裝 agent，更新時，只需在操作機上進行一次更新即可。

DOI:10.6814/NCCU201901225

- 可將任務寫成腳本批次執行。
- 使用 python 編寫，維護更簡單。

Go-ethereum 搭建私有鏈大致分為以下步驟（一）創建節點帳號（二）搭建私有鏈節點（三）連接各節點，下面說明各步驟。

（一）創建節點帳號: 透過 Control Tower 命令 Eth-nodes 產生節點帳號資訊，在我們的實驗裡是透過 Ethereum Wallet api 產生節點資訊（公私鑰、節點地址等等），並將各節點產生的節點地址回傳 Control Tower，待後續初始化私有鏈使用。

（二）搭建私有鏈節點: 依照使用者需求創造出 Genesis.json。Genesis.json 裡記錄了此私有鏈網路的所有初始條件。我們將 Control Tower 收集的節點地址寫入 Genesis.json，用來分配初始節點資產。透過定義 Genesis.json 裡的 GasLimit 來制定私有鏈區塊大小。Genesis.json 上還包含許多資訊，例如: 私有鏈 ID、挖礦難度等等。因應使用者需求，Genesis.json 被設計成依照使用者需求動態產生，以方便實驗進行。Control Tower 會將 Genesis.json 回傳給 Eth-nodes，讓 Eth-nodes 產生符合私有鏈的節點。

（三）連接各節點: 將各節點產生的節點 ID 回傳給 Control Tower 整合出 Static-node.json，並發送回給所有節點。節點連線時會優先訪問 Static-node.json 進行連線，此目的是為了讓 P2P 網路成為完全圖（Complete graph），讓任兩節點間皆存在 P2P 連線。

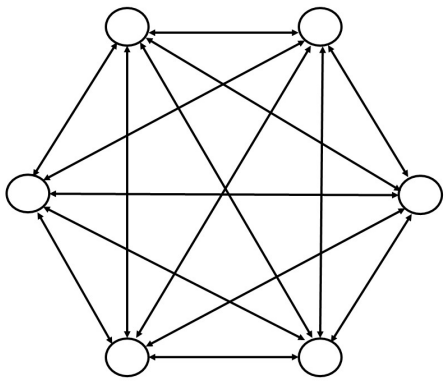

圖 5.3: 一個網路拓樸為六個節點完全圖的 P2P 網路。

DOI:10.6814/NCCU201901225

第六章　實驗結果

本章節對我們實作的 TwoStepBFT 進行測試。我們將節點運行在 Amazon 雲端平台上，測試的雲端虛擬機器分別為 Amazon EC2 t2.small 與 Amazon EC2 t2.large。t2.small 的硬體規格為 1 個虛擬 CPU 與 2GB 的記憶體；t2.large 的硬體規格為 2 個虛擬 CPU 與 8GB 的記憶體，探討不同硬體規格的虛擬機器對共識效率是否有影響。交易型態則是基本的以太坊轉帳交易。我們以交易的吞吐量 Throughput（每秒共識多少筆交易量）與區塊的延遲性 latency（完成一個區塊共識平均需要多少時間）來衡量共識的效率。我們也測試共識演算法的延展性（Scalability)，以瞭解當參與共識的節點數量變多時，共識效能會受到怎樣的影響。最後，我們將節點分散部署在世界各地，讓該系統較貼近真實的應用場景並且與同資料中心實驗比較。

6.1　環境建設

利用第五章所述方法部署私有鏈後，我們先讓節點互相交易約 15 分鐘，待交易存入交易池內，這可以讓共識時產生的區塊塞滿足夠的交易。接下來將運行共識演算法約 1 小時，並將結果記錄下來。

6.2　同資料中心吞吐量與延遲性測試

此小節我們將探討我們的演算法在同資料中心的實驗環境下且分別運行在不同規格虛擬機器時，觀察吐量與延遲性的變化。我們將機器都運行於美國-奧勒岡資

DOI:10.6814/NCCU201901225

料中心。對於測試基本的交易吞吐量與延遲性測試，我們透過控制創世區塊裡的 GasLimit 來控制單個區塊能包含的最大交易數量，區塊大小分別為每個區塊包含 500、1000、4000、8000 筆交易。實驗總共進行了五種節點數分別是 20、40、60、80、100 個節點。由於實驗並沒有加入錯誤節點的假設，故我們將 TimeOut 設定成 600000 秒，避免因為初始 TimeOut 設定過短而頻繁的更換回合。

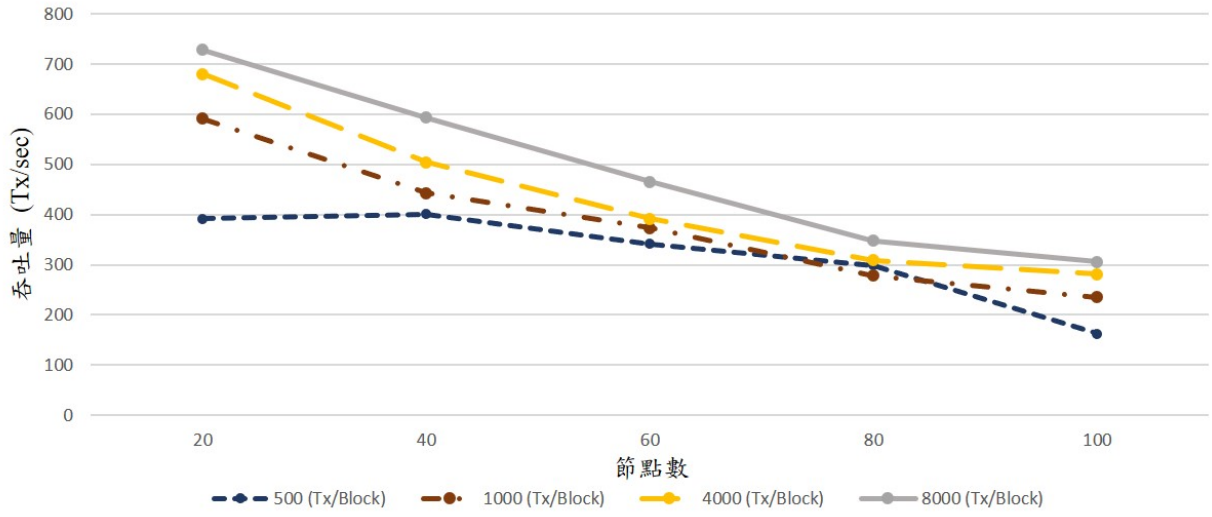

圖 6.1: 同資料中心的交易吞吐量表現 (t2.small)。

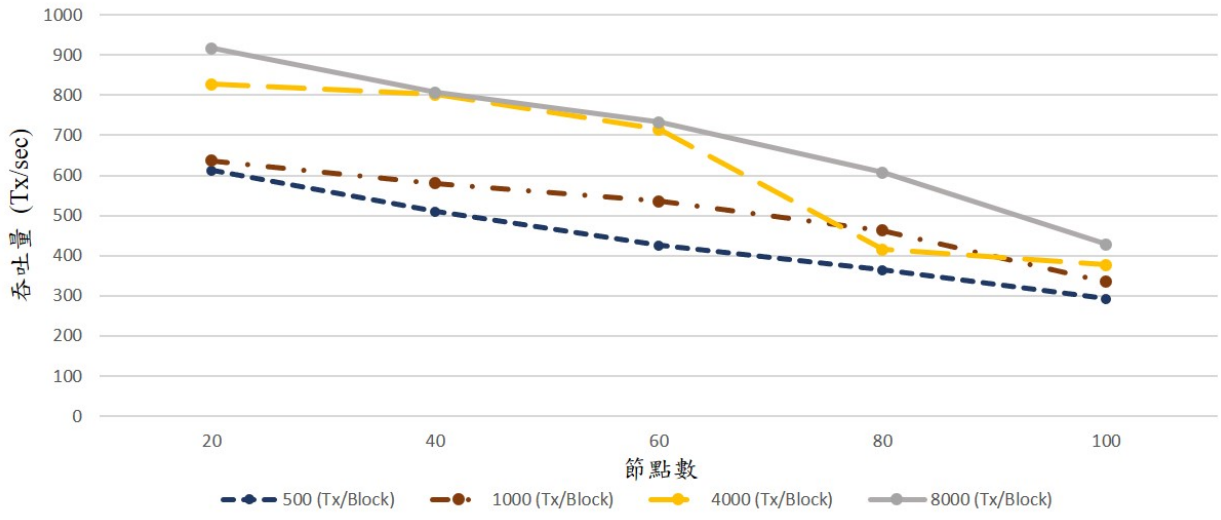

圖 6.2: 同資料中心的交易吞吐量表現 (t2.large)。

DOI:10.6814/NCCU201901225

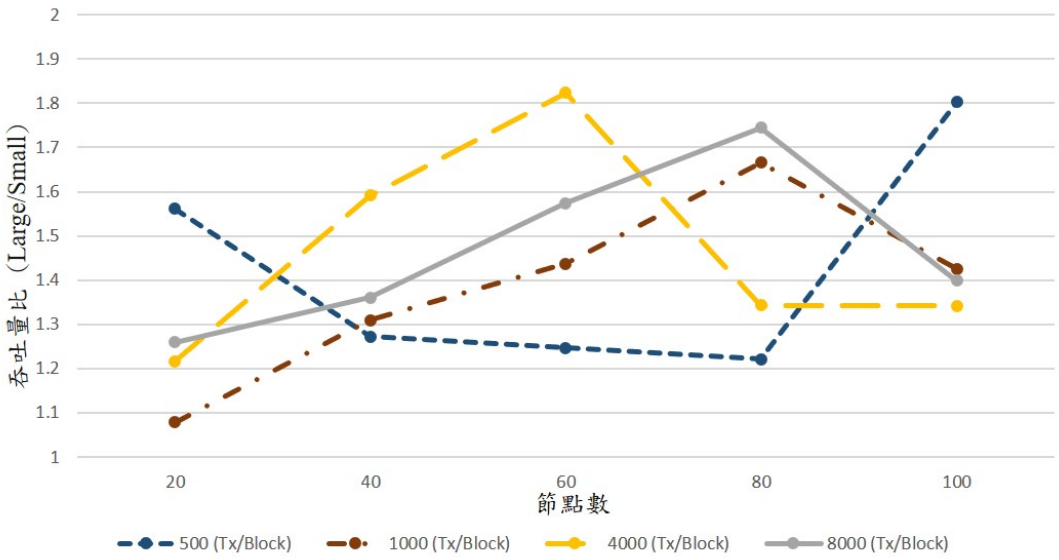

圖 6.3: 同資料中心不同虛擬機器吞吐量比 (t2.large/t2.small)。

從圖 6.1、圖 6.2 中可以看到我們的共識演算法在節點數增加時，吞吐量會隨之降低；區塊大小增加時，吞吐量會隨之增加。在 100 個節點參與共識時，吞吐量依舊能達到每秒 400 筆交易左右的共識速度，對於節點數的容忍能具有良好的延展性。從圖 6.3 中能夠觀察到 t2.large 相較 t2.small 皆具有較高的吞吐量。圖 6.4、圖 6.5 可以看到我們的共識演算法在節點數增加時延遲也會隨之增加。當區塊大小控制在 500 筆交易時，延遲皆低於 3.1 秒。

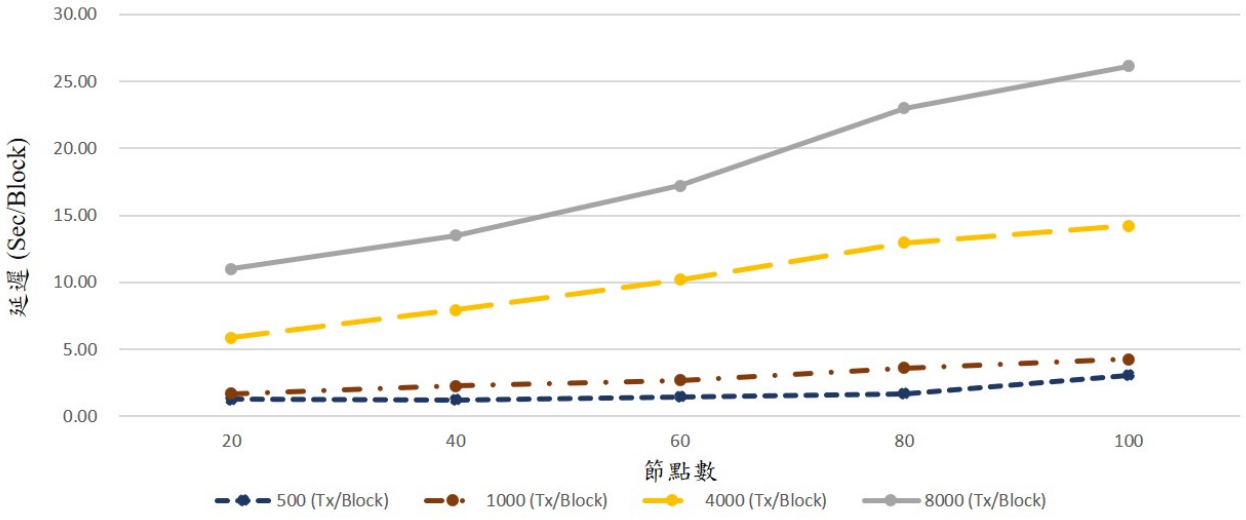

圖 6.4: 同資料中心的延遲性表現 (t2.small)。

DOI:10.6814/NCCU201901225

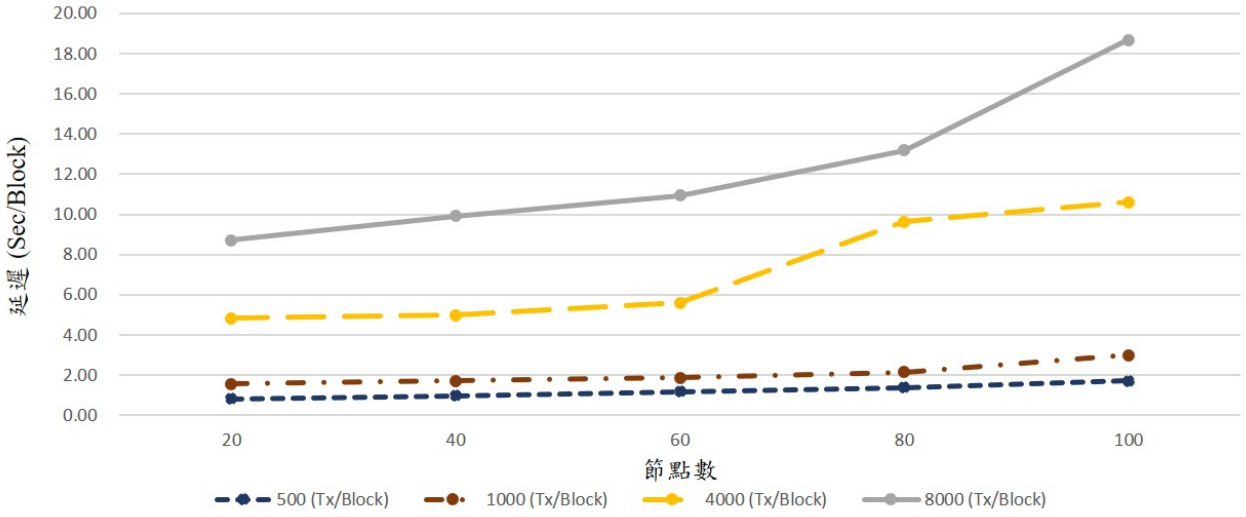

圖 6.5: 同資料中心的延遲性表現 (t2.large)。

6.3 跨資料中心吞吐量與延遲性測試

此小節我們將探討我們的演算法在跨資料中心的實驗環境下，觀察吐量與延遲性的變化，並與同資料中心實驗進行比較。與同資料中心的吞吐量與延遲性測試實驗一樣，我們先讓節點互相交易約 15 分鐘，讓交易存入交易池內。接下來將運行共識演算法約 1 小時，並將結果記錄下來。區塊大小分別為每個區塊包含 500、1000、4000、8000 筆交易。實驗總共進行了五種節點數分別是 20、40、60、80、100 個節點。由於實驗並沒有加入錯誤節點的假設，故我們將 TimeOut 設定成 600000 秒，避免因為初始 TimeOut 設定過短而頻繁的更換回合。不同的是我們將節點分散的部署在世界各地，這讓實驗更加貼近真實應用。在實驗裡我們將節點平均分散在（美國-奧勒岡、美國-維吉尼亞北部、日本-東京、新加坡、倫敦）。

DOI:10.6814/NCCU201901225

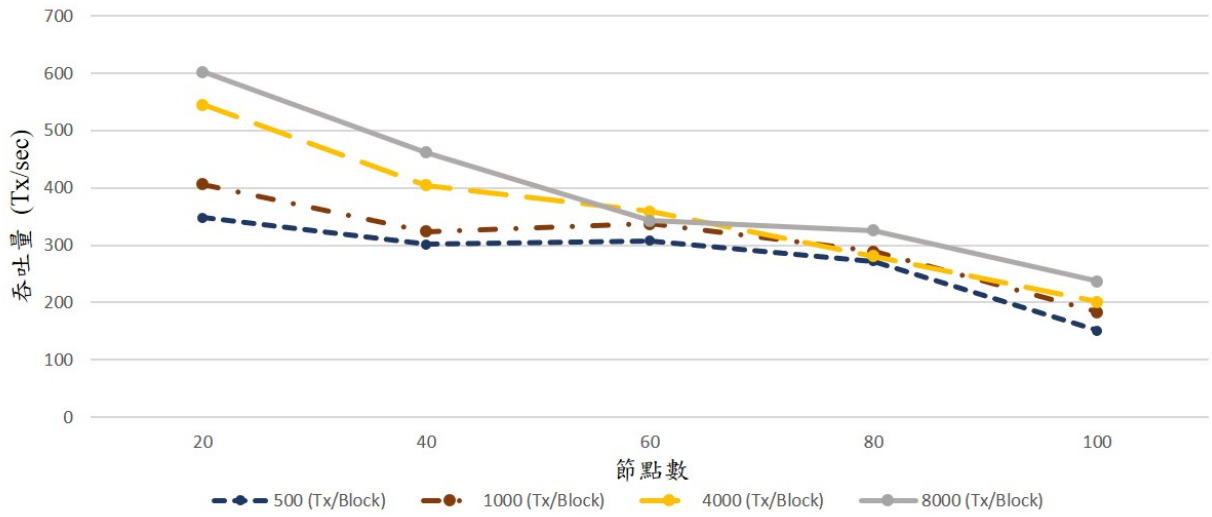

圖 6.6: 跨資料中心的交易吞吐量表現 (t2.small)。

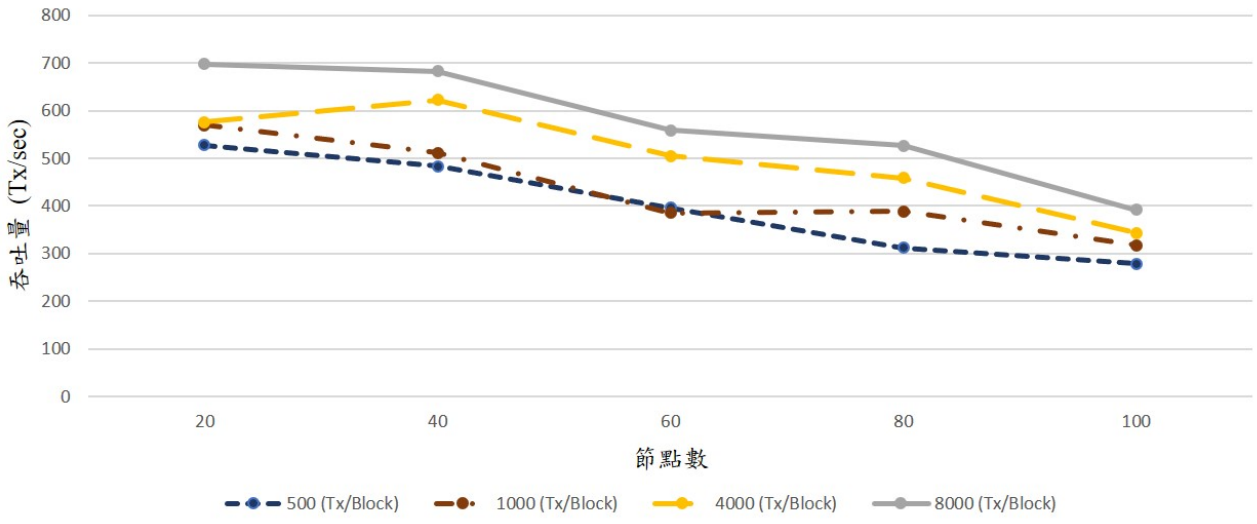

圖 6.7: 跨資料中心的交易吞吐量表現 (t2.large)。

DOI:10.6814/NCCU201901225

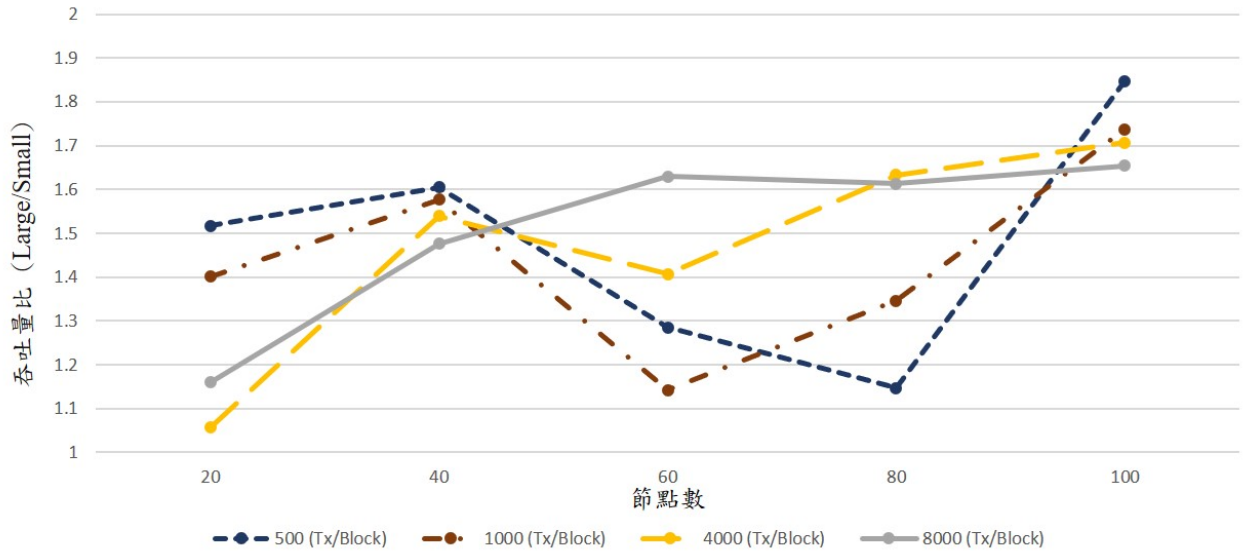

圖 6.8: 跨資料中心不同虛擬機器吞吐量比 (t2.large/t2.small)。

從圖 6.6、圖 6.7 中可以看到我們的共識演算法在節點數增加時，吞吐量會隨之降低；區塊大小增加時，吞吐量會隨之增加。從圖 6.8 中能夠觀察到 t2.large 相較 t2.small 皆具有較高的吞吐量。圖 6.9、圖 6.10 可以看到我們的共識演算法在節點數增加時延遲也會隨之增加。當區塊大小控制在 500 筆交易時，延遲皆低於 3.3 秒。

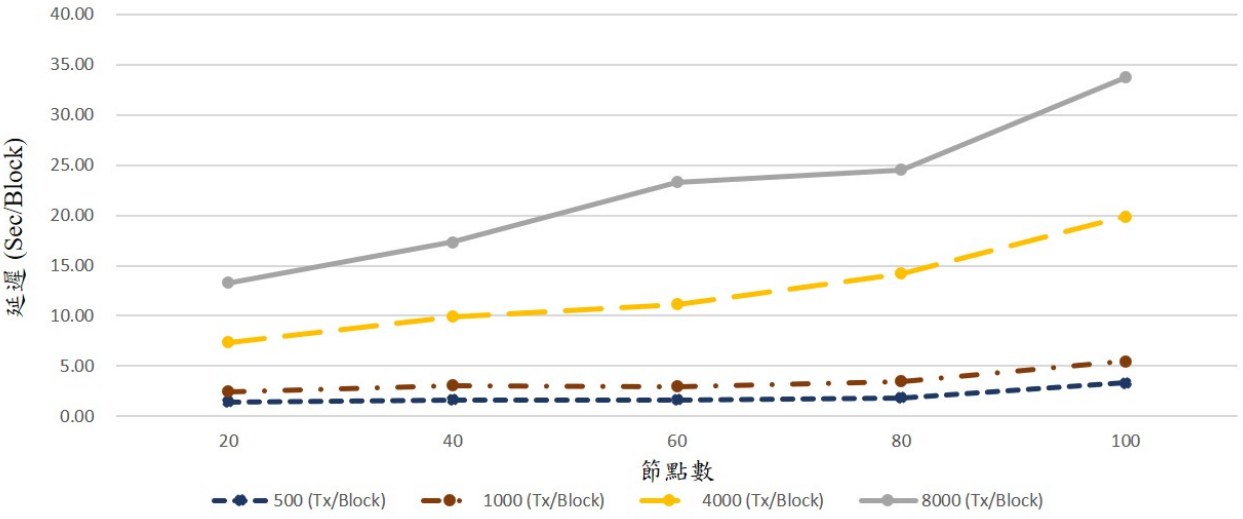

圖 6.9: 跨資料中心的延遲性表現 (t2.small)。

DOI:10.6814/NCCU201901225

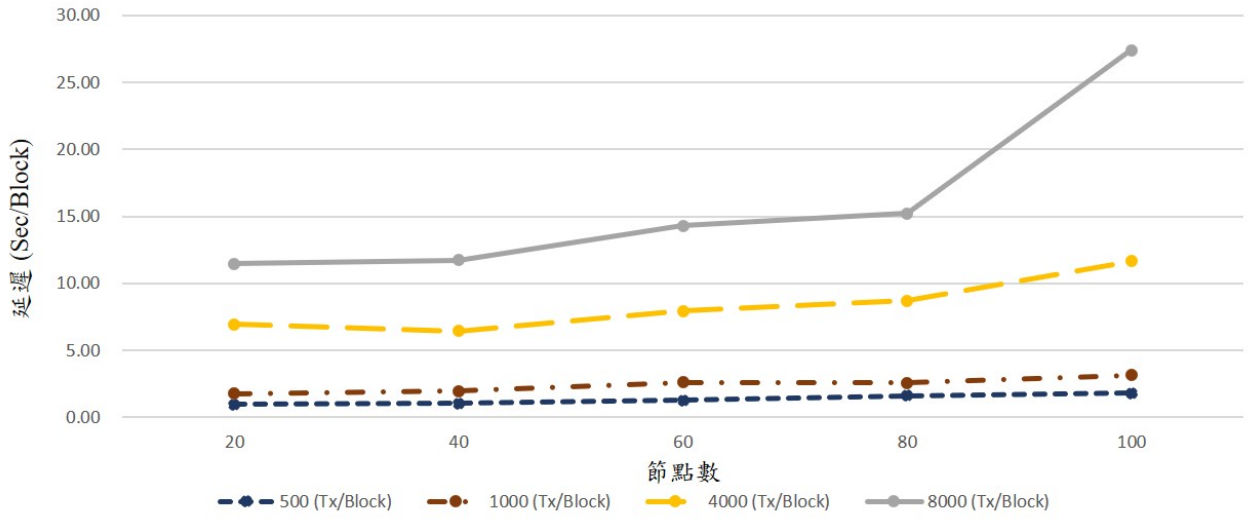

圖 6.10: 跨資料中心的延遲性表現 (t2.large)。

圖 6.11 與圖 6.12 將同資料中心實驗與跨資料中心實驗作比較，圖 6.11 探討雲端機器為 t2.small 時的兩組實驗；圖 6.12 探討雲端機器為 t2.large 時的兩組實驗。從圖 6.11、圖 6.12 中能觀察到在跨資料中心的實驗吞吐量幾乎皆小於同資料中心的實驗，從上述實驗中能得知資料中心的設置與虛擬機器的規格確實會影響共識效率。

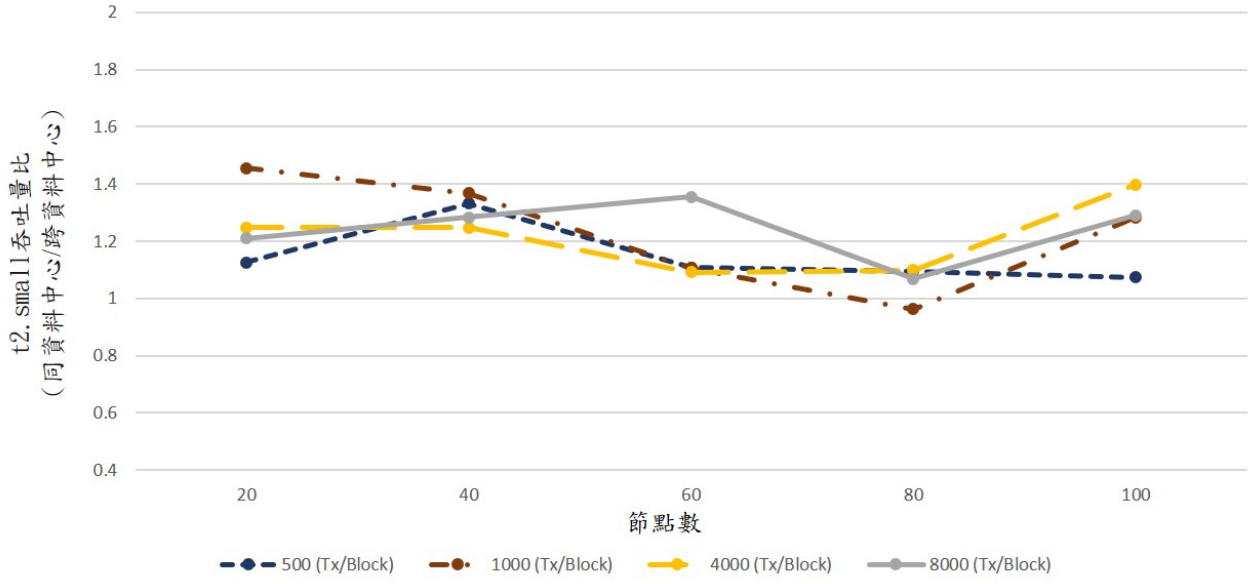

圖 6.11: 同資料中心與跨資料中心吞吐量比 (t2.small)。

DOI:10.6814/NCCU201901225

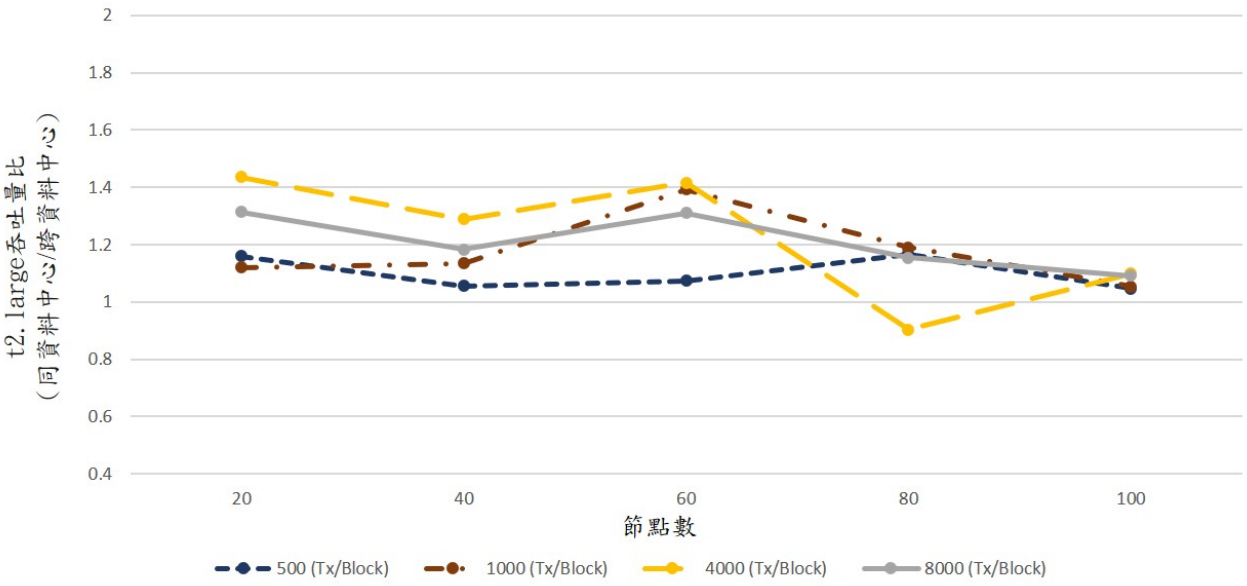

圖 6.12: 同資料中心與跨資料中心吞吐量比 (t2.large)。

DOI:10.6814/NCCU201901225

第七章　相關研究

7.1　公有鏈上共識演算法

7.1.1　Proof-of-Work

比特幣是目前最著名的區塊鏈應用，使用 PoW（Proof-of-Work）作為比特幣的共識系統。在該系統裡節點需要花費時間與電腦運算資源，嘗試解出一組數學公式的答案，用來獲得廣播區塊的權力。該答案也稱之為 Nonce。將 Nonce 值附於區塊內，其他節點就能透過簡單數學公式，驗證該答案是否有效。除非能控制超過 51% 以上的系統運算能力，才能進行攻擊。在私有鏈上因為參與共識的節點較少，因此系統內計算能力總和非常小。攻擊者可以輕易地以較高的計算能力，推翻過去的共識結果，因此私有鏈不傾向選擇使用 PoW。目前比特幣約 7TPS [8] 著名的以太坊 [4]、萊特幣 [15] 也都使用 PoW 作為共識機制。以太坊除了能夠儲存交易外，還能執行程式碼，運行智能合約。

7.1.2　Proof-of-Stack

PoS（Proof-of-Stack）是把持有資產數量作為參考。資產數量越多者，越有機會擔任下個區塊的廣播者。PoS 認為持有資產量高的人會越傾向維護貨幣價值，因此發動攻擊的可能性越低。不過 PoS 缺點也非常明顯，PoS 很可能造成貨幣不流通。如果貨幣不流通，該貨幣也失去其價值。PPCoin [20] 是目前少數運行 PoS 的區塊鏈。

DOI:10.6814/NCCU201901225

7.1.3 混合型共識演算法

混合型共識演算法概念是從公有鏈節點中，隨機挑選一群人運行拜占庭容錯共識演算法。這類的方法需要具有非常公平的抽籤演算法。Algorand [12] 作為一個混合型共識演算法，最重要的一個機制便是引入了 VRF（Verifiable Random Function），中文稱作可驗證隨機函數。透過該函數，區塊鏈上節點能夠自行驗證是否成為該回合的提議者，並且能夠提出證明供其他節點進行驗證。Algorand 的處理能力超過 1000TPS 且延遲低於五秒。相似的混合型演算法還包含 Bitcoin-NG [10] 與 Dexon [9] 等。

7.2 私有鏈上共識演算法

7.2.1 Proof-of-Authority

PoS（Proof-of-Authority）是由一群授權的節點來負責驗證區塊與廣播區塊。不同於 PoW 驗證節點不需強大的運算能力，也不需要像 PoS 得擁有很多資產才能廣播區塊。但此節點必須是大家公認的已知節點，且通過一定程度身分驗證。一旦節點勾結其他節點進行攻擊，那鏈上的其他管理者可以移除或替換這些惡意節點。目前以太坊在原始碼中也提供 PoA 共識演算法名為 Clique [5] 共識演算法。

7.2.2 三步驟拜占庭共識演算法

BFT 是拜占庭容錯（Byzantine-Fault-Tolerant）的縮寫。在拜占庭容錯的私有區塊鏈上，即使系統中部分節點當機或存在惡意節點情況下，區塊鏈依舊具備安全性（Safety）與活性（Liveness）等性質。一般來說，一個回合需要執行三個步驟。第一個步驟為播其提議；第二個步驟透過交互投票，檢查該提議者是否只有廣播一份提議；第三個步驟節點便開始投票，若一個節點蒐集到足夠多的同意票，便可以確認共識結果。知名的 BFT 演算法 PBFT [6] 便是根據上述三個步驟設計。而許多私有鏈區外鏈系統，都是根據上述想法而延伸出來。例如:Tendermint[3] 是一

DOI:10.6814/NCCU201901225

種區塊鏈共識機制主要以 Go 語言撰寫，與 PBFT 相似，每個回合都是一次廣播與兩次投票來產生共識，不同於 PBFT 之處在於 Tendermint 引入 Lock 概念，藉此維護系統的 Safety 和 Liveness。在官方文件裡提及理想狀況下 64 個節點能有約 4000 TPS。MSIG-BFT 是一個三步驟的共識演算法，與一般 BFT 演算法不同的是透過收集 f+1 個數位簽章來確保共識提議唯一性，讓共識演算法保證安全性及唯一性。目前 MSIG-BFT 實做在 Go-ethereum 上。LibraBFT [2] 是 Libra 區塊鏈系統所使用的共識系統，在 LibraBFT 技術白皮書提到，選擇 Hotstuff [21] 作為 LibraBFT 的演算法基礎。Hotstuff 採用了聚集簽名，讓系統內的訊息傳輸量從 $O(n^2)$ 變成 $O(n)$，在 libra 白皮書裡提到，理想狀況下 100 個節點仍有約 1000 TPS。

7.2.3 兩步驟拜占庭共識演算法

過去也有少部分的 BFT 演算法在一個回合中只需要兩個步驟，例子包含 FaB [3]、Zyzzyva [14]、SBFT [16]、Hydrachain [19]。然而，FaB 與 Zyzzyva 已被指出無法保證活性，換句話說，共識演算法可能永遠無法達成共識。另一方面，Hydrachain 也被指出無法保證安全性，也就是說，不同的正常節點可能會有不同的共識節果，這在區塊鏈上及代表分岔。雖然 SBFT 能夠保證安全性與活性，但是在有拜占庭攻擊時，會改用類似 PBFT 的三步驟設計。因此，在系統狀況良好時，SBFT 每回合只需要執行兩個步驟。但是在系統狀況不良時，SBFT 每回合仍需要執行三個步驟。在 100 個節點情況下吞吐量約 50 TPS。

DOI:10.6814/NCCU201901225

第八章　討論與結論

8.1 討論

下面我們探討影響共識效率可能因素如:

- 節點數影響共識效率: 節點數量上升會導致傳輸時間拉長，所以完成每個步驟時間都與節點數有關。在演算法設計裡，初始 Timeout 長度是固定的，如果初始的 Timeout 設定過低且未考慮傳輸時間，演算法會頻繁的 Timeout 更換新回合。進而拖慢整體共識效率。
- 區塊大小影響共識效率: 從實驗裡發現隨著區塊大小變大，共識吞吐量也會隨之增加，但增長幅度有限會趨於平緩；然而共識延遲雖也會隨之增加，但增長幅度卻急遽上升。因此，我們推斷隨著區塊大小變大，吞吐量將隨之增加，但會趨於平緩甚至下降。然而延遲卻會不斷上升且越來越急遽。因此，我們無法透過無限制增加區塊大小，來增加我們共識效率。

8.2 結論

TwoStepBFT 是一種新的兩步驟拜占庭共識演算法，可以保證部分同步假設下的安全性和活躍性。該演算法在任何情況下都只需要兩個步驟，不須額外複雜的方法來達成共識。通過實驗結果，我們知道共識吞吐量與延遲都與節點數息息相關。在 AWS 上測試 TwoStepBFT 效率皆能達到數百 TPS。並且透過將節點分散搭建在世界各地，讓實驗更加符合真實應用場景，且吞吐量依舊能維持約 300 TPS。

DOI:10.6814/NCCU201901225

透過學習其他共識演算法，TwoStepBFT 未來也能持續改進，例如:(1)Hotstuff 透過聚集簽名，來降低系統傳輸複雜度。(2) 由於 BFT 類的演算法透過輪流廣播訊息來達成共識，如果共識期間存在錯誤節點將會拖慢整體共識效率，因此如果能將錯誤節點排出共識系統，將能增加安全性並且提高共識效率。

DOI:10.6814/NCCU201901225

參考文獻

[1] R3 corda, 2016. https://www.r3.com.

[2] Librawhitepaper, 2019. https://libra.org/en-US/wp-content/uploads/sites/23/2019/06/LibraWhitePaper_en_US.pdf.

[3] I. Abraham, G. Gueta, D. Malkhi, and J.-P. Martin. Revisiting fast practical byzantine fault tolerance: Thelma, velma, and zelma. *arXiv preprint arXiv:1801.10022*, 2018.

[4] V. Buterin. A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2014.

[5] V. Buterin. clique for go ethereum. https://github.com/ethereum/go-ethereum/tree/master/consensus/clique, 2016.

[6] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[7] C.-W. Chen, J.-W. Su, T.-W. Kuo, and K. Chen. Msig-bft: A witness-based consensus algorithm for private blockchains. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 992–997. IEEE, 2018.

[8] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains. pages 106–125, 2016.

DOI:10.6814/NCCU201901225

[9] dexon org. dexon whitepaper. https://dexon.org/whitepaper, 2018.

[10] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[11] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Technical report, Massachusetts Inst of Tech Cambridge lab for Computer Science, 1982.

[12] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[13] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[14] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 45–58. ACM, 2007.

[15] C. Lee. Litecoin, 2011.

[16] J.-P. Martin and L. Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[17] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.

[18] Y.-J. Shiu. Nccu bft for go ethereum. https://github.com/ethereum/EIPs/issues/650, 2017.

DOI:10.6814/NCCU201901225

[19] W. T. Tsai, L. Yu, C. J. Hu, Y. F. Yao, and G. N. Li. Hydrachain: Design of a private blockchain. https://github.com/HydraChain/hydrachain/blob/develop/hc_consensus_explained.md, 2016.

[20] P. Vasin. Blackcoin's proof-of-stake protocol v2. *URL: https://blackcoin.co/blackcoin-pos-protocol-v2-whitepaper. pdf*, 71, 2014.

[21] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069*, 2018.

DOI:10.6814/NCCU201901225