性能基准测试和优化Hyperledger Fabric 区块链平台

（英文原文）

Parth Thakkar *印度Trichy国家技术学院pthakker@in.ibm.com

Senthil Nathan N印度IBM研究实验室snatara7@in.ibm.com印度

Balaji ViswanathanIBM研究实验室bviswana @ in.ibm.com区块

摘要 - 近期授权链平台的受欢迎程度显着提高。 Hyperledger Fabric是一个这样的授权区块链平台，也是Linux Foundation [13]托管的Hyperledger项目之一。 Fabric包含各种组件，如智能合约、背书者、提交者、验证者和排序器。鉴于区块链平台的性能是企业应用的主要关注点，本文中我们将进行全面的实证研究，以表征Hyperledger Fabric的性能，并确定潜在的性能瓶颈，以更好地理解系统。

我们采取一个两阶段的方法。在第一阶段，我们的目标是了解各种配置参数的影响，例如区块大小、背书策略、通道、资源分配、状态数据库选择对交易吞吐量和延迟的影响，以提供有关配置这些参数的各种指导原则。另外，我们还旨在确定性能瓶颈和热点。我们观察到（1）背书策略验证，（2）区块中交易的顺序策略验证，以及（3）状态验证和提交（使用CouchDB）是三大瓶颈。

在第二阶段，我们专注于基于我们的观察优化Hyperledger Fabric v1.0。我们介绍和研究了各种简单的优化，例如密码学组件中进行背书策略验证的积极缓存（3倍性能改进）和并行验证策略验证（7倍改进）。此外，我们在状态验证和提交阶段（2.5倍改进）期间增强并测量了对CouchDB的现有批量读/写优化的影响。通过结合所有三种优化，我们将整体吞吐量提高了16倍（即从140 tps提高到2250 tps）。

I. 引言

区块链技术最初获得了普及，因为它被看作是摆脱中介和分散系统的一种方式。此后，区块链从不同领域和用例中看到了越来越多的兴趣。区块链是共享的分布式账本，记录交易并由网络中的多个节点维护，其中节点彼此不信任。每个节点都保存着通常表示为一系列区块的账本的相同副本，每个区块都是逻辑交易序列。每个区块封装其前一个区块的散列，从而保证账本的不变性。

区块链常常被誉为新一代数据库系统，实质上是一个分布式交易处理系统，其中节点不可信并且系统需要实现拜占庭容错[18]。区块链提供序列化、不变性和加密可验证性不像数据库系统那样具有单一信任点;这些属性已经引发了各行各业的区块链采用。

区块链网络可以是无权限的，也可以是授权的。在诸如比特币[28]、以太坊[20]等无权限的网络或公共网络中，任何人都可以加入网络来执行交易。由于公共网络中有大量节点，因此使用工作量证明共识方法来排序交易并创建一个区块。在授权的网络中，每个参与者的身份都是已知的，并且通过加密方式进行身份验证，使得区块链可以存储谁执行了哪个交易。此外，这样的网络可以具有内置的广泛的访问控制机制，以限制谁可以：（a）读取和追加账本数据，（b）发布交易，（c）管理对区块链网络的参与。

授权的网络非常适合需要认证参与者的企业应用程序。授权网络中的每个节点都可以由不同的组织拥有。此外，企业应用程序需要复杂的数据模型和可表达性，这可以使用智能合约来支持[29]。企业能够整合多种系统而无需构建集中式解决方案，并在不信任的各方之间建立一定程度的信任或引入可信的第三方。贸易融资[26]和食品安全[21]就是区块链应用的例子，与现有的松散耦合集中系统相比，价值优势在于参与者看到其提供的可见性。关于授权区块链平台的性能以及它们在低延迟下处理大量交易的能力存在很多担忧。另一个值得关注的是描述交易的语言丰富程度。不同的区块链平台，例如Quorum [14]、Corda [25]使用派生于分布式系统领域的不同技术来解决这些问题。 Hyperledger Fabric [16]是一个企业级开源授权区块链平台，它通过信任模型和可插拔组件，具有模块化设计和高度可定制性。 Fabric目前被用于许多不同的

用例，如全球贸易数字化[33]、SecureKey [15]、Everledger [5]，并且是我们性能研究的重点。

Fabric由各种组件组成，如背书节点、排序服务和提交者。此外，它构成处理交易的各个阶段，如背书阶段、排序阶段、验证和提交阶段。由于众多组件和阶段，Fabric提供各种可配置参数，例如区块大小、背书策略、通道，状态数据库。因此，建立高效区块链网络的主要挑战之一是为这些参数找到正确的值集。例如，根据应用和需求，可能需要回答以下问题：

- 为了实现更低的延迟，区块大小应该是多少？
- 可以创建多少个通道，资源分配应该是多少？
- 什么类型的背书策略更有效率？
- 当它用作状态数据库时，GoLevelDB [7]和CocuhDB [2]之间的性能差异有多大？

为了回答上述问题并找出性能瓶颈，我们使用各种可配置参数对Fabric v1.0进行全面的实证研究。具体来说，我们的三大贡献如下所列。

1）我们通过进行1000多次实验，通过改变分配给五个主要参数的值，对Fabric平台进行了全面的实证研究。因此，我们提供六条关于配置这些参数以获得最佳性能的指导原则。

2）我们确定了三个主要的性能瓶颈：（i）加密操作，（ii）区块中交易的串行验证，以及（iii）对CouchDB的多个REST API调用。

3）引入并研究了三种简单的优化，以在单通道环境下将总体性能提高16倍（即从140 tps提高到2250 tps）。

本文的其余部分组织如下：§II介绍Hyperledger Fabric的体系结构。 §III简要描述了我们研究的目标，而§IV深入研究了实验装置和工作负载特性。 §V和§VI呈现我们的核心贡献，而§VII描述相关工作。最后，我们在§VIII中总结这篇论文。

II.背景：HYPERLEDGER Fabric结构和配置参数

Hyperledger Fabric是一个授权区块链系统的实现，它具有许多适用于企业级应用的独特属性。它可以运行以Go / JAVA / Nodejs语言实现的任意智能合约（又称链码s [1]）。它支持用于交易验证的应用程序信任模型和可插拔共识协议等等。 Fabric网络由不同组织的不同类型的实体、peer节点、排序服务节点和客户端组成。它们中的每一个都具有由会员服务提供商（MSP）[9]提供的网络身份，通常与组织相关联。网络中的所有实体都可以看到所有组织的身份，并可以对其进行验证。

A.结构关键组件

Peer。一个peer节点执行链码，实现用户智能合约，并将账本维护在一个文件

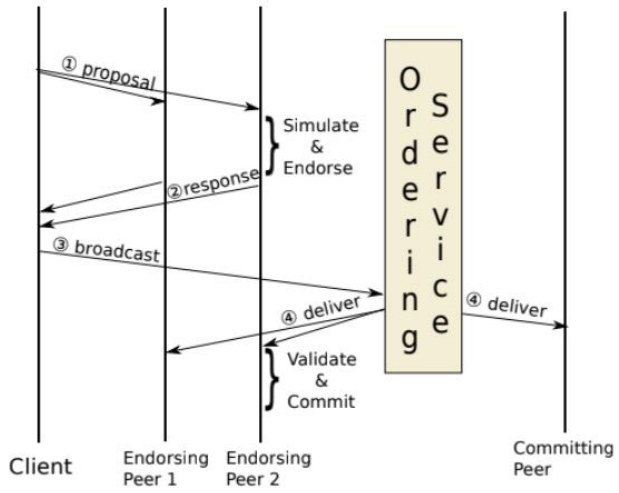


Fig. 1. Transaction flow.

系统。链码可以通过定义明确的账本API访问共享状态。一个Peer被进一步隔离为一个背书Peer（它具有链码逻辑并执行它来背书一个交易）或一个（不承载链码逻辑的）提交peer。无论这种差异，这两种类型的Peer都会维护账本。此外，两个Peer都将当前状态保存为键值存储中的StateDB，以便链码可以使用数据库查询语言查询或修改状态。

背书策略。 链码是用通用语言编写的，它们在网络中的不受信任的peer上执行。这带来了多个问题，一个是非确定性执行，另一个是信任任意给定peer的执行结果。背书策略通过指定背书策略的一部分来解决这两个问题，即指定需要模拟交易并背书或数字签署执行结果的Peer组。背书策略[3]被指定为网络主体标识上的布尔表达式。这里的主体(principal)是特定组织的成员。

系统链码。与用户链码不同，系统链码与普通用户链码具有相同的编程模型，并且内置于peer可执行文件中。 Fabric实现各种系统链码；生命周期系统链码（LSCC）- 安装/实例化/更新链码；背书系统链码（ESCC）- 通过数字签署回复来批准交易；验证系统链码（VSCC）- 根据背书策略确认交易的背书签名；配置系统链码（CSCC）- 管理通道配置。

通道。Fabric引入了一个称为通道的概念，作为两个或多个Peer之间通信的“私有”子网，以提供隔离级别。一个通道的交易只能由Peer成员和参与者看到。不可变的账本和链码基于每个通道。此外，共识适用于每个通道，也就是说，跨通道交易没有明确的排序器。

排序服务。排序服务节点（OSN）参与共识协议并将交易打包成区块然后

[3]AND（'Org1.member'，'Org2.member'，'Org3.member'）请求每一个的签名，而OR（'Org1.member'，AND（'Org2.member'，'Org3.member'））要求签署组织1或两个签名，即来自组织2和3的签名

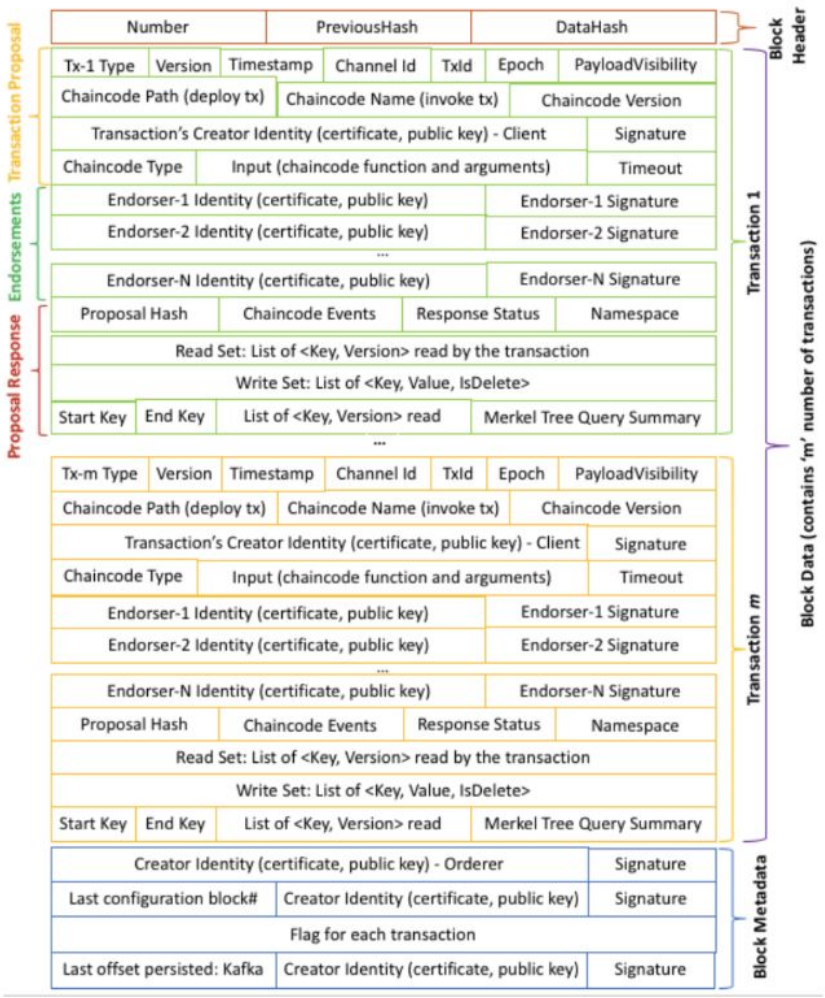

Block Header
Number
PreviousHash
DataHash
Block Data (contains 'm' number of transactions)
Transaction 1
Transaction Proposal
Tx-1 Type
Version
Timestamp
Channel Id
TxId
Epoch
PayloadVisibility
Chaincode Path (deploy tx)
Chaincode Name (invoke tx)
Chaincode Version
Transaction's Creator Identity (certificate, public key) - Client
Signature
Chaincode Type
Input (chaincode function and arguments)
Timeout
Endorsements
Endorser-1 Identity (certificate, public key)
Endorser-1 Signature
Endorser-2 Identity (certificate, public key)
Endorser-2 Signature
...
Endorser-N Identity (certificate, public key)
Endorser-N Signature
Proposal Response
Proposal Hash
Chaincode Events
Response Status
Namespace
Read Set: List of <Key, Version> read by the transaction
Write Set: List of <Key, Value, IsDelete>
Start Key
End Key
List of <Key, Version> read
Merkel Tree Query Summary
...
Transaction m
Tx-m Type
Version
Timestamp
Channel Id
TxId
Epoch
PayloadVisibility
Chaincode Path (deploy tx)
Chaincode Name (invoke tx)
Chaincode Version
Transaction's Creator Identity (certificate, public key) - Client
Signature
Chaincode Type
Input (chaincode function and arguments)
Timeout
Endorser-1 Identity (certificate, public key)
Endorser-1 Signature
Endorser-2 Identity (certificate, public key)
Endorser-2 Signature
...
Endorser-N Identity (certificate, public key)
Endorser-N Signature
Proposal Hash
Chaincode Events
Response Status
Namespace
Read Set: List of <Key, Version> read by the transaction
Write Set: List of <Key, Value, IsDelete>
Start Key
End Key
List of <Key, Version> read
Merkel Tree Query Summary
Block Metadata
Creator Identity (certificate, public key) - Orderer
Signature
Last configuration block#
Creator Identity (certificate, public key)
Signature
Flag for each transaction
Last offset persisted: Kafka
Creator Identity (certificate, public key)
Signature

图2.Hyperledger Fabric v1.0块结构

通过gossip通信协议传递给Peer。图2显示了Fabric v1.0中的区块结构。排序服务是模块化的，并且支持可插入的共识机制。默认情况下，使用基础Kafka/Zookeeper集群实现序列化排序（即共识）。 OSN将交易发布到kafka主题，并利用kafka主题中记录的有序和不可变性来在区块中生成唯一的有序交易序列。自上一个区块以来，新增交易达到设定的最大数量，或自从上一个区块时间超过配置的超时时间时，新区块被生成。当满足任何一个条件时，OSN会公布自上次切割消息偏移量以来的所有交易消息偏移量的时间标记并打包一个区块。然后该区块被传送到peer节点。

客户端。客户端应用程序负责将交易提议放在一起，如图2所示。客户端同时向一个或多个Peer提交交易提议，以便通过背书收集提议响应以满足背书策略。然后它将该交易广播给排序器以包含在一个区块中并传递给所有Peer进行验证和提交。在Fabric v1.0中，客户有责任确保交易的格式良好并符合背书策略。

B. Hyperledger Fabric中的交易处理流

与使用排序器执行[32]交易模型的其他区块链网络不同，Fabric使用模拟-排序器-验证和提交模型。图1描述了涉及3个步骤的交易流程，1）背书阶段 - 在选择的Peer上模拟交易并收集状态变化; 2）排序阶段 - 通过共识协议对交易进行排序;和3）验证阶段 - 验证，然后提交到账本。在Fabric上提交交易之前，网络需要由参与组织，他们的MSP和peer身份引导。首先，在排序器网络上与相应的组织MSP一起创建一个通道。其次，每个组织的Peer加入通道并初始化账本。最后，所需的链码被安装在通道上。

背书阶段。一个使用Fabric SDK [8]，[6]，[10]的客户端应用程序构造一个交易提议来调用一个链码函数，该函数反过来会对账本状态执行读和/或写操作。该提议是使用客户端的凭据签署的，客户端同时将其发送给1个或更多个Peer。链码的背书策略决定了客户端需要将提议发送给模拟的组织peer。

首先，每个背书节点验证提交者被授权调用该通道上的交易。其次，peer执行链码，可以访问peer的当前账本状态。交易结果包括响应值，读取集和写入集。所有的读操作都会读取当前的账本状态，但所有的写操作都会被截取并修改一个私有的交易工作区。第三，背书Peer调用一个称为ESCC的系统链码，该系统链码用Peer的身份签署该交易响应，并用提议响应回复给客户端。最后，客户端检查提议响应以验证它是否具有Peer的签名。客户收集来自不同Peer的足够的提议响应，验证背书是否相同。由于每个Peer都可能在区块链中的不同高度执行交易，因此提议响应可能会有所不同。在这种情况下，客户必须将提议重新提交给其他Peer，以获得足够的匹配响应。

排序阶段。客户端向排序服务广播格式良好的交易信息。该交易将包含读写集、背书peer签名和通道ID。排序服务不需要检查交易的内容来执行其操作。它接收来自不同客户端的各种通道的交易，并按每个通道进行排队。它为每个通道创建交易区块，使用其身份签署该区块，并使用gossip消息协议将它们交付给peer。

验证阶段。所有的Peer，包括通道上的背书peer和提交peer接收来自网络的区块。Peer首先验证该区块上的Orderer（排序器）签名。每个有效区块都被解码，并且在执行MVCC验证之前，区块中的所有交易都会首先通过VSCC验证。

VSCC验证。验证系统链码根据为链码指定的背书策略评估交易中的背书。如果背书策略不满足，那么该交易标记为无效。

MVCC验证。顾名思义，多版本并发控制[30]检查可以确保在背书阶段交易读取的密钥版本与提交时本地账本中的当前状态相同。这与为并发控制完成的读写冲突检查类似，并且在区块中的所有有效交易（由VSCC验证标记）上按顺序执行。如果读取设置的版本不匹配，则表示并发的前一个（在此之前或之前）交易修改了读取的数据，并且自成功提交（它是背书的）以来，交易被标记为无效。为确保不发生幻读，对于范围查询，重新执行查询并比较结果的散列（也作为背书中捕获的读集的一部分存储）。

账本更新阶段。作为交易处理的最后一步，通过将区块附加到本地账本来更新账本。保存所有密钥

的当前状态的StateDB用有效交易的写入集（由MVCC验证标记）更新。 StateDB的这些更新是针对一个交易区块自动执行的，然后将StateDB更新为区块中所有交易执行后的状态。

C.配置参数

我们的目标是研究Fabric在各种条件下的性能，以了解系统不同方面的选择如何影响性能。然而，参数空间很广泛，我们限制我们的选择来全面覆盖一些组件，并且广泛关注系统的其他方面，以便我们可以确定组件级别选择的相互影响。为此，我们选择主要从Peer的角度来理解和描述整体表现。更具体地说，我们保持Orderer，Gossip（物理网络）等静态，以便它不影响我们的实验和观察。接下来，我们描述本研究中考虑的五个参数及其意义。

1）区块大小。交易在排序服务处进行批量处理，并使用Gossip协议以区块的形式交付给Peer。每个Peer一次处理一个区块。像排序器签名验证一样的加密处理是按照每区块进行的，而不像交易背书签名验证（按交易）。变化的区块大小也会带来吞吐量与延迟之间的折衷，为了获得更好的图像，我们将结合交易到达率进行研究。

2）背书策略。背书策略规定在交易请求可以提交给排序器之前需要执行多少次交易和签署，以便交易可以通过Peer的VSCC验证阶段。 VSCC验证交易的背书需要评估背书策略表达与收集的背书情况并检查可满足性[24]，这是NP-Complete。此外，检查包括验证身份及其签名。背书策略的复杂性将影响资源以及收集和评估资源的时间。

3）通道。通道隔离交易。提交给不同通道的交易在排序、交付和处理上是互相独立的，尽管在相同的Peer中。通道为Fabric中交易处理的各个方面带来固有的并行性。尽管应用程序和参与者组合确定了要使用的通道数量以及要处理的通道数量，但它对平台性能和可伸缩性有重大影响。

4）资源分配。Peer运行CPU密集型的签名计算和作为系统链码一部分的路由验证。在交易模拟期间由Peer执行的用户链码增加到这种混合。我们改变对等节点上的CPU核心数量来研究其效果。虽然网络特性很重要，但我们假设这个研究的延迟非常低的数据中心或高带宽网络。

5）账本数据库。 Fabric支持键值存储的两种选择，CouchDB [2]和GoLevelDB [7]来维护当前状态。两者都是键值存储，而GoLevelDB是嵌入式数据库，CouchDB使用客户端-服务器模型（通过安全HTTP使用REST API访问）并支持文档/ JSON数据模型。

III. 问题描述

我们工作的两个主要目标是：

1）性能基准。对Fabric核心组件进行深入研究，并针对常见使用模式测试Fabric性能。我们的目标是在改变所列参数的配置时，研究系统的吞吐量和延迟特性，以便理解性能指标和参数之间的关系。基于我们的观察，我们的目标是推出并提出一些高级指南，这对开发人员和部署工程师非常有用。

2）优化。使用代码级别的工具识别瓶颈并提取操作项以提高Fabric的整体性能。在确定瓶颈时，我们的目标是引入和实施优化来缓解这些瓶颈。

IV. 实验方法

我们研究吞吐量和延迟作为Fabric的主要性能指标。**吞吐量(Throughput)**是交易提交到账本的速度。**延迟(Latency)**是从应用程序发送交易提议到交易提交所花费的时间，由以下延迟组成：

TABLE I
DEFAULT CONFIGURATION FOR ALL EXPERIMENTS UNLESS SPECIFIED OTHERWISE.

Parameters	Values
Number of Channels	1
Transaction Complexity	1 KV write (1-w) of size 20 bytes
StateDB Database	GoLevelDB
Peer Resources	32 vCPUs, 3 Gbps link
Endorsement Policy AND/OR	OR [AND(a ,b, c), AND(a, b, d), AND(b, c, d), AND(a, c, d)]
Block Size	30 transactions per block
Block Timeout	1 second

- **背书延迟** - 客户收集所有提议响应（为了背书）所需的时间。
- **广播延迟** - 客户端提交到排序器和排序器背书客户端之间的时间延迟。
- **提交延迟** - Peer验证和提交交易所花费的时间。
- **排序延迟** - 交易花费在排序服务上的时间。由于排序服务的性能没有在这项工作中研究，我们没有提出这种延迟。此外，我们在区块级别定义以下三个延迟：
- **VSCC验证延迟** - 根据背书策略，验证所有交易背书签名集合（在一个区块中）所用的时间。
- **MVCC验证延迟** - 通过采用§II-B中所述的多版本并发控制来验证块中的所有交易所花费的时间。
- **账本更新延迟** - 更新状态数据库所花费的时间，其中包含区块中所有有效交易的写入集。

由于这项工作的主要目标之一是确定性能瓶颈，因此我们的负载生成器4跨越多个客户端，每个客户端都通过不断生成交易而不是遵循分布模型（比如泊松）来压迫系统。每个客户端也同时发送提议请求并整理背书。这些交易是异步提交的，以达到指定的速率而无需等待提交。但是，基准框架使用名为fetch-block5的工具来跟踪通信，以计算吞吐量和延迟。此外，我们安装了Fabric源代码来收集细粒度的延迟，如MVCC延迟和其他延迟。对于多通道实验，所有组织和所有Peer加入该通道。虽然其他组合是可能的，但我们相信我们的方法会对系统进行压力测试。

A.设置和工作负载

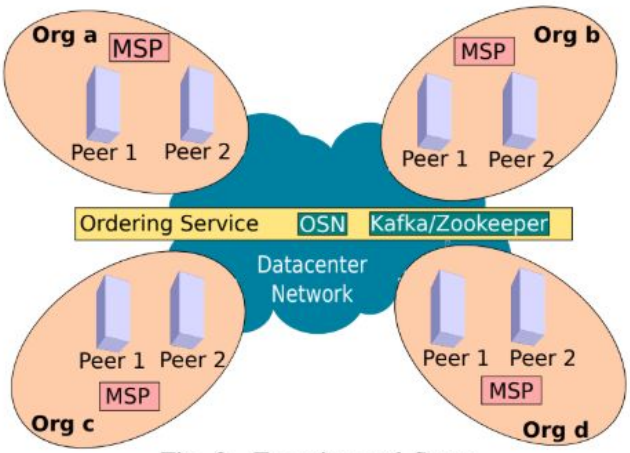


Fig. 3. Experimental Setup

我们的测试Fabric网络由4个组织组成，每个组织有2个对等节点，共有8个对等节点，如图3所示。有1个排序器节点带有支持它的kafka-zookeeper集群。所有节点和kafka-zookeeper都在IBM SoftLayer Datacenter中的x86 64虚拟机上运行。每个虚拟机分配32个vCPU的Intel（R）Xeon（R）CPU E5-2683 v3 @ 2.00GHz和32 GB内存。用于生成负载的三台功能强大的客户端分配了56个vCPU和128 GB内存。节点连接

到3 Gbps数据中心网络。

4https：//github.com/thakkarparth007/fabric-load-gen 5https：//github.com/cendhu/fetch-block

在区块链缺乏标准基准的情况下，我们通过调查大约12种基于Fabric架构的内部客户解决方案来构建我们自己的基准，以满足各种用例的需求。我们确定了全面的共同定义模式、数据模型和需求。循环模式之一是每个链码调用仅在一个资产或数据单元上进行操作，并将标识符传递给它。查询逻辑由更高级别的应用程序层完成，通常不查询区块链数据。这种模式被模拟为简单的只写交易（1w，3w和5w--表示写入的键的数量）。另一种常见模式是链码读取和写入一小组keys，如读取JSON文档，更新字段并将其写回。我们将它们建模为1、3和5键的读写。由于我们已经将我们的基准建模为模仿生产中的真实世界区块链应用，因此我们没有考虑其他宏基准。

V. 实验结果

在本节中，我们将研究§II-C中列出的各种可配置参数对Hyperledger Fabric性能的影响。本节介绍的吞吐量和交易延迟是多次运行的平均值。总的来说，我们进行了超过1000次的实验。

A.交易到达率和区块大小的影响

TABLE II
CONFIGURATION TO IDENTIFY THE IMPACT OF BLOCK SIZE AND TRANSACTION ARRIVAL RATE.

Parameters	Values
Tx. Arrival Rate	25, 50, 75, 100, 125, 150, 175 (tps)
Block Size	10, 30, 50, 100 (#tx)

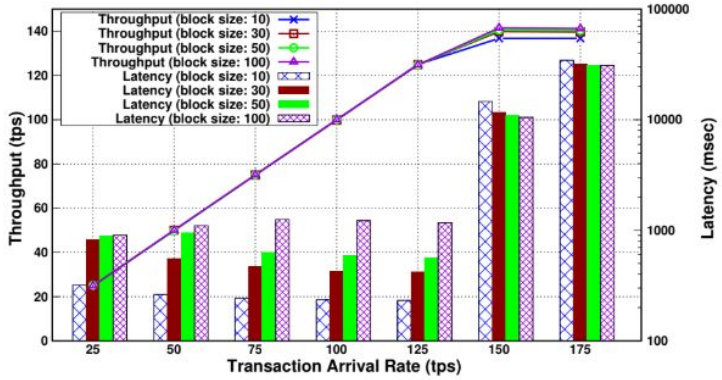
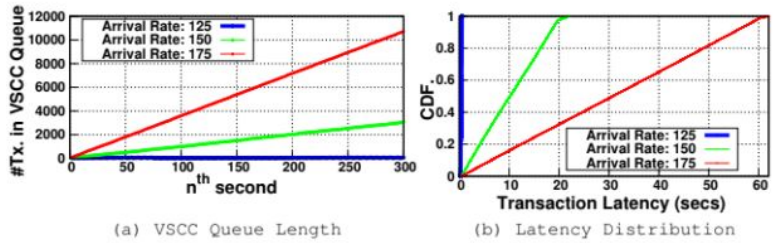


Fig. 4. Impact of the block size and transaction arrival rate on performance.

图4绘出了不同交易到达率下各种区块大小的平均吞吐量和延迟。表二列出了各种交易到达率和所使用的区块大小。对于其他参数，请参阅表I.

观察1：随着交易到达率的增加，吞吐量按预期线性增加，直到它平均在140 tps处饱和，如图4所示。当到达率接近达到或高于饱和点时，延迟显著增加（即，从100ms的数量级到10秒的数量级）。这是因为在验证阶段在VSCC队列中等待的排序交易数量迅速增长（参见图5），这影响了提交延迟。然而，随着进一步增加到达率，我们没有观察到对背书和广播延迟的影响，但影响到了提交延迟，这是因为VSCC只使用单个vCPU，因此新的交易提议利用Peer上的其他vCPU进行模拟和背书。结果，只有验证阶段成为瓶颈。在本次实验中，背书和广播延迟分别是大约12毫秒和9毫秒。



(a) VSCC Queue Length (b) Latency Distribution

Fig. 5. The length of VSCC queue, and latency distribution for various arrival rates (block size was set to 30).

观察2：对于低于饱和点的到达率，随着区块大小的增加，延迟增加。例如，当到达率为50 tps时，随着区块大小从10增加到100，交易延迟从242 ms增加到了1250 ms，增加了5倍。原因在于，随着区块大小的增加，排序器处的区块创建时间增加，因此平均而言，交易必须等待排序器一段时间。例如，当交易到达率为100 tps时，对于区块大小50和100，区块创建速率分别为2和1块每秒，导致延迟加倍。

观察3：对于大于饱和点的到达率，随着区块大小的增加，等待时间减少。例如，当到达率为150 tps时，随着区块大小从10增加到200，交易延迟从14秒减少到10秒。这是因为验证和提交大小为n的区块的时间总是少于验证和提交m个区块所花费的时间（每个尺寸为n/m）。结果，吞吐量增加了3.5%。注意，排序器节点处的区块创建速率总是大于验证者处理速率，而与区块大小和到达速率无关。

观察4：对于区块大小，随着到达速率增加到区块大小临界值以下，延迟增加。当到达速率增加到区块大小以上时，延迟减小。对于较小的区块大小和较高的到达率，创建区块的速度更快（而不是等待区块超时），从而减少了排序器节点处的交易等待时间。相反，例如，当区块大小为100，到达率从25增加到75 tps时，延迟从900 ms增加到1250 ms。原因在于，随着速率的增加，一个区块中的交易数量增加，验证和提交阶段花费的时间也增加。请注意，如果在一秒钟内没有达到区块大小限制，则由于区块超时而创建块。

观察5：即使在最高吞吐量时，资源利用率也非常低。随着到达率从25提高到175 tps，平均CPU利用率仅从1.4%上升到6.7%6。原因在于，在VSCC验证阶段执行的CPU密集型任务（即按背书策略验证每个交易的签名集）一次只处理一个交易。由于这个串行执行，只使用了一个vCPU。

准则1：当交易到达率预计低于饱和点时，为了实现应用程序的较低交易延迟，请始终使用较低的区块大小。在这种情况下，吞吐量将与到达率相匹配。

准则2：当交易到达率预计很高时，为了实现更高的吞吐量和更低的交易延迟，请始终使用更高的区块大小。

措施1：CPU资源利用率低下。一个潜在的优化是在VSCC验证阶段一次处理多个交易，如§VI-B所示。

B.背书策略的影响

TABLE III
CONFIGURATION TO IDENTIFY THE IMPACT OF ENDORSEMENT POLICIES.

Parameters	Values
Endorsement Policy (AND/OR)	1) OR [a, b, c, d] 2) OR [AND(a ,b), AND(a, c), AND(a, d), AND(b, c), AND(b, d), AND(C, D)] 3) OR [AND(a ,b, c), AND(a, b, d), AND(b, c, d), AND(a, c, d)] 4) AND [a, b, c, d]
Endorsement Policy (NOutOf)	1) 1-OutOf [a ,b ,c, d] 2) 2-OutOf [a ,b ,c, d] 3) 3-OutOf [a ,b ,c, d] 4) 4-OutOf [a ,b ,c, d]
Tx. Arrival Rate	125, 150, 175 (tps)

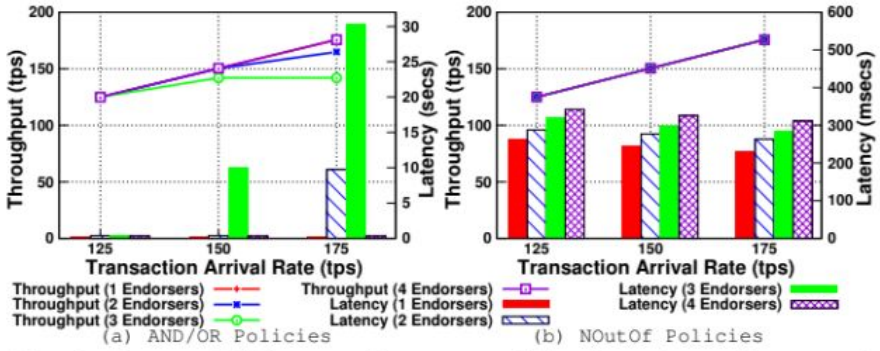


Fig. 6. Impact of different endorsement policies (AND/OR and NOutOf).

图6绘出了在不同交易到达率下使用AND/OR和NOutOf语法定义的各种背书策略的吞吐量和延迟时间。表三列出了本研究中使用的各种策略。请注意，'a'，'b'，'c'和'd'表示四个不同的组织。对于其他参数，请参阅表I.尽管用于定义四种背书策略的语法（AND/OR，NOutOf）是不同的，但在语义上它们是相同的。例如，用这两种语法列出的第三项策略表示任何三个组织的背书足以通过VSCC验证。

观察6：多个子策略和一些加密签名验证的组合影响了性能，如图6所示。由于少量签名验证，没有子策略的第一个和第四个AND / OR策略执行了与NOutOf策略相同的操作。

6除非另有指定，CPU利用率指定为32个vCPU的平均值。按绝对值计算，6.7％等于6.7×32 = 214％

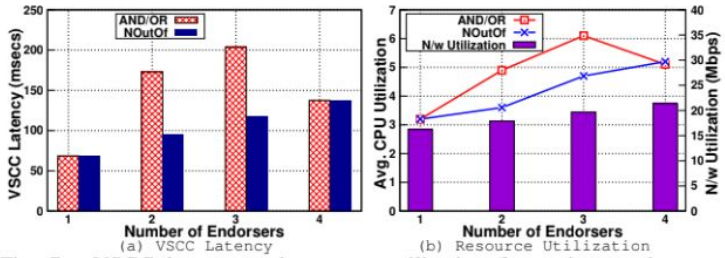


Fig. 7. VSCC latency, and resource utilization for various endorsement policies (arrival rate = 125).

TABLE IV
CONFIGURATION TO IDENTIFY THE IMPACT OF CHANNELS.

Parameters	Values
Number of Channels	1, 2, 4, 8, 16
Tx. Arrival Rate	125 to 2500 tps with a step of 25

使用子策略，子策略的数量（即搜索空间e）和签名的数量决定了性能。例如，第二和第三个AND / OR策略的吞吐量分别比其他策略低7％和20％。图7绘出了各种策略下peer节点的VSCC延迟时间和资源利用情况。随着签名验证数量的增加（专门针对NOutOf），VSCC延迟从68 ms线性增加到137 ms。当存在子策略时（与第二和第三AND / OR策略一样），VSCC等待时间显著增加（即，分别为172ms和203ms）。图7（b）显示了类似的资源利用趋势。请注意，由于每个交易中编码的x.509证书数量增加，所以区块字节数会随着签名数量的增加而增加。

在策略验证阶段有三个主要的CPU密集型操作（不包括满足性检查），下面列出了这些操作。

1）身份的反序列化（即x.509证书）。

2）对组织MSP的身份验证[9]。

3）验证交易数据的签名。

因此，随着子策略（即搜索空间）的增加，要验证的身份和签名的数量、CPU利用率和VSCC延迟增加。有趣的是，MSP标识符不与x.509证书一起发送。因此，策略评估人员必须使用多个组织MSP验证每个x.509证书，以确定正确的证书。以150 tps的到达速度运行5分钟，我们观察到220K这样的验证，其中96K验证失败，导致CPU和时间的浪费。

准则3：要实现高性能，请使用较少数量的子策略和签名来定义策略。

措施2：由于密码操作是CPU密集型的，我们可以通过维护一个缓存的反序列化身份及其MSP信息来避免某些程序操作，如第VI-A页所示。这不会带来安全风险，因为身份是长期存在的，并且保留单独的证书撤销列表（CRL）。

C.通道和资源分配的影响

我们将不同通道数量的到达率分为两类：当延迟范围为[0.4-1s]时为非过载，而在延迟范围为[3--40s]时为过载。

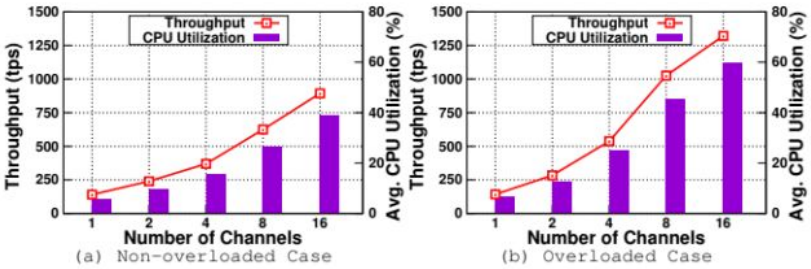


Fig. 8. Impact of the number of channels on performance.

TABLE V
CONFIGURATION TO IDENTIFY THE IMPACT OF RESOURCE ALLOCATION.

Parameters	Values
Number of Channels	4, 16
Resources (same for all peers)	(2, 4, 8, 16, 32) vCPUs, 3 Gbps
Tx. Arrival Rate	350 tps for 4 channels 850 tps for 16 channels

图8描绘了这两个类别的平均吞吐量和CPU利用率，表4列出了本研究中使用的各种通道和交易到达率，其他参数参见表1。 §IV。

观察7：随着通道数量的增加，吞吐量增加，等待时间减少，CPU等资源利用率也增加，如图8所示。例如，随着通道数量从1增加到在图16中，吞吐量从140tps增加到832tps（即，在非过载情况下为6×）再增加到1320tps（即在过载情况下为9.5×）。这是因为每个通道都独立于其他通道并保持自己的区块链。因此，并行执行的多个区块（每个通道一个）的验证阶段和最终账本更新提高了CPU利用率，从而提高了吞吐量。

图9（a）和（b）分别描绘了在同样peer节点下分配不同数量的vCPU的情况下，4个和16个通道的吞吐量、背书和提交延迟。图9（c）绘出了所有vCPU之间的绝对（而不是平均）CPU利用率。表V列出了分配的各种vCPU数量、使用的通道数量和交易到达率。有关其他参数，请参阅表I.

观察8：在中等负载下，当分配的vCPU数量少于通道数量时，性能会降低。例如，当16个通道分配的vCPU数量少于16个时，吞吐量从848tps显著降低到32tps（26 x） - 参见图9（b）。此外，由于CPU的竞争很激烈，平均背书和提交延迟都会发生爆炸（分别从37 ms到21 s和640 ms到49 s）。此外，在背书阶段有大量请求超时。一旦发生超时，我们将该交易标记为失败。在分配2个vCPU的情况下，与4个vCPU相比，更多的背书请求获得了超时。因此，如图9（b）所示，2个vCPU观察到的背书和提交延迟（成功交易）小于4个vCPU。

由于CPU竞争，VSCC延迟也增加，影响了提交延迟，如图9（c）所示。增加vCPU分配增加了CPU利用率，从而提高性能达到vCPU与通道数匹配的峰值。

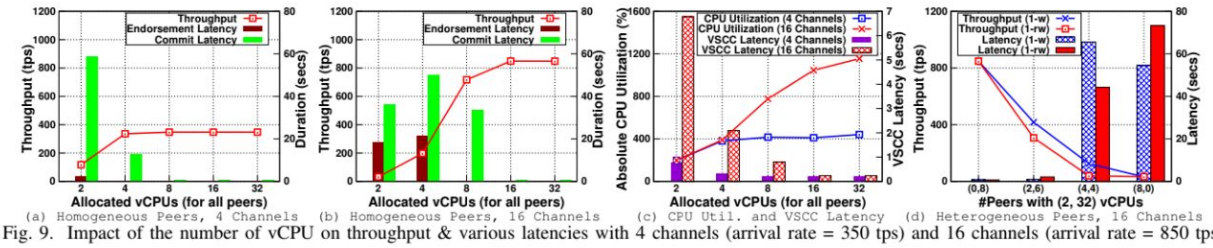


Fig. 9. Impact of the number of vCPU on throughput & various latencies with 4 channels (arrival rate = 350 tps) and 16 channels (arrival rate = 850 tps).

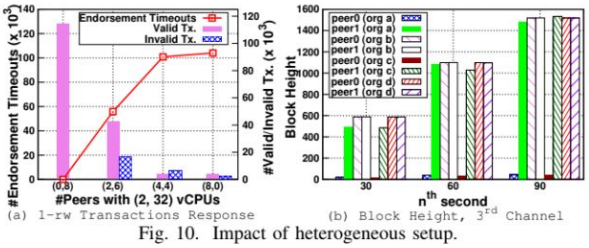


Fig. 10. Impact of heterogeneous setup.

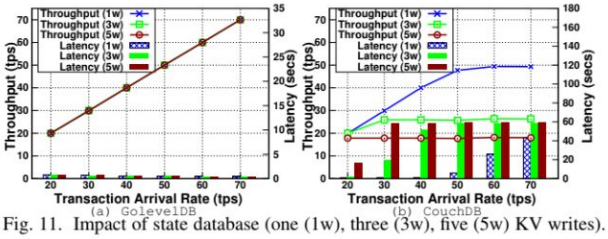


Fig. 11. Impact of state database (one (1w), three (3w), five (5w) KV writes).

TABLE VI
CONFIGURATION TO IDENTIFY THE IMPACT OF HETEROGENEOUS SETUP.

Parameters	Values
Number of Channels	16
Transaction Complexity	1 KV write (1-w) & 1 KV read/write (1-rw)
#Peers with (2, 32) vCPUs	(0, 8), (2, 4), (4, 4), (8, 0) peers
Tx. Arrival Rate	850 tps

除此之外，由于单线程顺序VSCC验证 ，额外分配的vCPU处于空闲状态- 参见图9（c）。

图9（d）绘制了异构安装中16个通道的吞吐量和延迟。表VI给出了分配给不同peer的各种数量的vCPU，以实现异构性，使用的通道数量、交易复杂度和交易到达率。对于其他参数，请参见表I.

观察9：在中等负载情况下，即使8个中分配给2个peer的vCPU数量少于通道数量，性能也会降低。例如，当只有2个vCPU分配给2个peer（其他32个vCPU）时，对于只写交易吞吐量从848 tps减少到417 tps（2×），对于读写交易减少到307 tps（2.7倍）。减少的原因是双重的，背书请求来自低配置peer的超时以及专门用于读/写事务的MVCC冲突。图10（a）绘制了由于读写事务的MVCC冲突导致的背书请求超时、有效交易和无效交易。随着低配peer的增加，背书请求超时增加。此外，由于MVCC冲突，失效交易的比例提高。这是因为与功能强大的peer相比，低配peer中的较低的区块提交率较低。由于peer的区块高度不同（参见图10（b）），所收集的读取集中的密钥版本不匹配。结果，MVCC冲突发生在状态验证期间，这使交易变得无效。

虽然我们还没有研究网络资源的影响，但我们认为这种影响会与CPU类似。这是因为，在网络带宽较低的情况下，区块和交易传递的延迟会增加。甚至异构网络资源，我们也认为这种影响与我们在CPU中观察到的影响相似。

准则4：要实现更高的吞吐量和更低的延迟，最好为每个通道分配至少一个vCPU。对于最优的vCPU分配，我们需要确定每个通道的预期负载并相应地分配足够的vCPU。

准则5：为了实现更高的吞吐量和更低的延迟，最好避免异构peer，因为性能会受低配peer的支配。

措施3：处理通道内和跨通道的交易可以得到改进，办法是分配更多的CPU能力，如§VI-B所示。

D.总帐数据库的影响

图11显示了不同交易复杂性下的GoLevelDB和CouchDB的多个交易到达率的平均吞吐量和延迟。表七列出了各种数据库、交易复杂性和使用的到达率。对于其他参数，请参阅表I.

观察10：使用GoLevelDB作为状态数据库的Fabric交易吞吐量比CouchDB大3倍。 GoLevelDB在单个通道上实现的最大吞吐量为140 tps（参见图4），而使用CouchDB时，吞吐量仅为50 tps（请参见图11（b））。此外，随着交易复杂性的增加，即对于多次写入，CouchDB的吞吐量从50 tps下降到18 tps，而GoLevelDB没

有这样的影响（参见图11（a））。 CouchDB和GoLevelDB之间显着性能差异的原因是后者对于peer进程是嵌入式数据库，而前者是通过安全HTTP使用REST API访问的。因此，与GoLevelDB相比，CouchDB的背书延迟、VSCC延迟、MVCC延迟和账本更新延迟更高，如图12（a）和（b）所示。使用CouchDB，与GoLevelDB相比，VSCC延迟增加了，因为交易模拟时，为了检索链码的背书策略，peer使用REST API调用访问状态数据库。同样地，使用CouchDB时MVCC延迟也增加了。

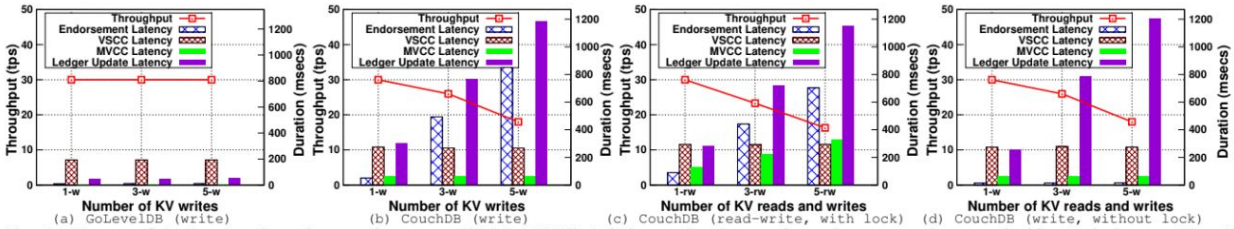


Fig. 12. Impact of database on throughput, endorsement, VSCC, MVCC, & ledger update latency for various transaction complexities (arrival rate = 30 tps)

TABLE VII
CONFIGURATION TO IDENTIFY THE IMPACT OF STATE DATABASE.

Parameters	Values
Database	GoLevelDB, CouchDB
Transaction Complexity	• (1, 3, 5) KV writes • (1, 3, 5) KV read/writes
Tx. Arrival Rate	20, 30, 40, 50, 60 (tps)

观察11：使用CouchDB，背书等待时间和账本更新延迟随着每笔交易写入次数的增加而增加，即从40ms和240ms分别增加，即一次写入800ms和三次写入1200ms，如图所示在图12（b）中，即使在背书阶段只写交易不访问数据库。这是因为背书阶段在整个数据库上获得了一个共享读锁，以向链码提供一致的数据视图（即可重复读隔离级别[17]）。同样，最终账本更新阶段在整个数据库上获得了独占写锁。因此，背书阶段和最终账本更新均争用此资源。特别是，使用CouchDB时最终账本更新更昂贵，因为对于交易写集合中的每个键值写，它必须执行下面的三个任务：

1. 通过发出GET请求检索key（如果它存在于数据库中）的先前修订号（用于CouchDB内的并发控制）。
2. 为该值构建一个文档（可能是JSON文档或二进制附件）。
3. 通过提交PUT请求来更新数据库。

因此，随着每笔交易写入次数的增加，账本更新延迟增加（参见图12（b））。由于上述三个耗时的串行操作，我们推测提交者（committer）锁定数据库很长时间，从而增加了背书延迟。为了验证我们的假设，我们通过在背书阶段和最终账本更新期间禁用整个数据库上的锁获取来进行实验。这种做法的副作用是仅在该链码处提供不可重复的读取隔离级别。由于我们的交易只是写密钥，所以这种副作用不会影响数据库的一致性。图12（d）显示了背书阶段的改进。平均背书延迟从800毫秒减少到40毫秒，验证了我们的假设。
观察12：仅增加每个交易中的读取次数，MVCC延迟增加，如图12（c）所示。这是因为随着读集合中条目数量的增加，在MVCC验证阶段增加了对CouchDB的GET REST API调用次数。随着写入次数的增加，MVCC延迟并没有增加，如图12（b）所示，因为它只检查是否有任何读取密钥已被修改。

准则6：GoLevelDB是状态数据库的一个更好的高性能选项。如果对只读交易的富文本查询支持很重要，那么CouchDB是一个更好的选择。在使用CouchDB时，设计应用程序和交易以读取/写入更少数量的键来完成任务。

措施4：CouchDB支持批量读/写操作[3]，无需额外的事务语义。运用批量操作将减少锁定持续时间并提高性能，如§VI-C所示。

措施5：使用数据库，如GoLevelDB和CouchDB，不支持快照隔离级别，会导致在背书和账本更新阶段整个数据库锁定。因此，我们未来的工作[11]是研究如何移除锁和/或使用支持快照隔离级别的PostgreSQL [12]等

数据库。

E. 伸缩性和容错

在Fabric中，伸缩性可以通过通道数量、加入通道的组织数量和每个组织的peer数量来衡量。从资源消耗的角度来看，背书策略的复杂性控制着网络的伸缩性。即使有大量组织或peer，如果背书策略只需要少数组织签名，那么表现就不会受到影响。这是因为，需要在网络中较少的节点处模拟交易以收集背书。伸缩性还可以根据地理上分布的节点数量和它们之间的区块传播的延迟来定义。排序服务节点的数量和它们之间使用的共识协议的选择也将影响伸缩性。虽然这些超出了本研究的范围，是网络伸缩性的重要方面。

节点故障在分布式系统中很常见，因此研究Fabric的容错能力非常重要。在我们的初始和早期研究中，我们观察到节点故障不会影响性能（在非过载情况下），因为客户端可以收集来自其他可用节点的背书。在负载较高的情况下，节点在故障后重新加入并由于丢失区块而同步账本被观察到具有大的延迟。这是因为虽然重新加入的节点处的区块处理速率处于峰值，但是其他节点继续以相同的峰值处理速率添加新区块。

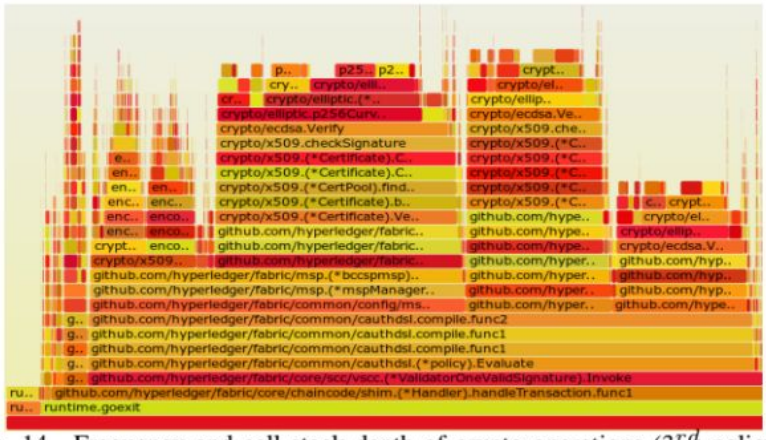


Fig. 14. Frequency and call stack depth of crypto operations (3^{rd} policy in AND/OR) – without the MSP cache

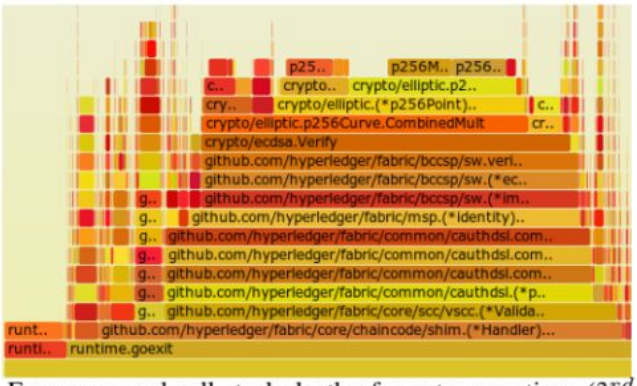


Fig. 15. Frequency and call stack depth of crypto operations (3^{rd} policy in AND/OR) – with the MSP cache.

VI. 研究优化

在本节中，我们介绍三种简单的优化方法，这些优化方法是基于以下列表中的措施：§VI-A中的§V-（1）MSP缓存，（2）§VI-B中区块的并行VSCC验证，以及（3）§VI-C中对CouchDB的批量读/写7在MVCC验证和提交。对于这些优化中的每一个，首先，我们分别研究性能改进。然后，我们结合所有三种优化来研究改进。

A. MSP缓存

Parameters	Values
Endorsement Policy (AND/OR)	all four from Table III
Tx. Arrival Rate	400, 500, 600 (tps)

TABLE VIII
CONFIGURATION TO IDENTIFY THE EFFICIENCY OF MSP CACHE.

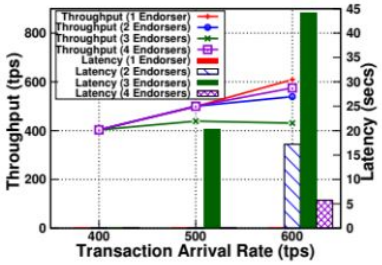


Fig. 13. Impact of MSP cache.

加密操作是非常CPU密集型的，在本节中，我们研究在加密模块中的下列两个操作中使用缓存的功效：

1）身份的反序列化（即x.509证书）。

2）验证组织的MSP的身份。

为了避免每次对序列化身份的反序列化，我们一个哈希映射来缓存反序列化身份，使用序列化身份充当映射的key。同样，为了避免每次验证多个MSP的身份，我们使用了一个哈希映射，身份做key，身份对应的

7A Fabric社区提议并且批量操作API的实现可用，但未与MVCC集成并且最终提交

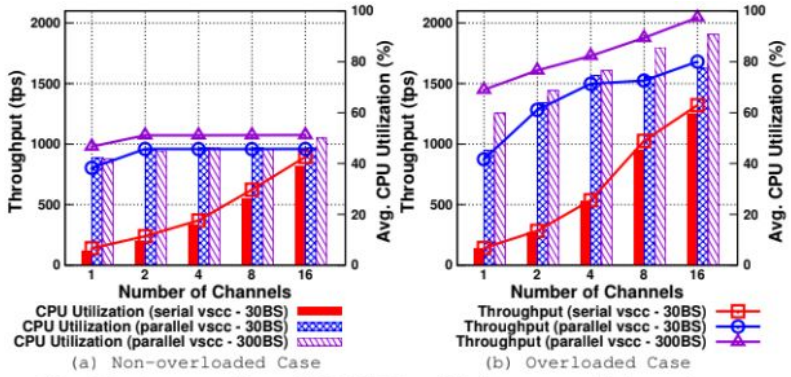


Fig. 16. Impact of parallel VSCC validation on multichannel setup.

MSP做值。此外，我们采用ARC [27]算法进行缓存替换。在身份撤销期间，我们适当地使缓存条目无效。

图13绘出了MSP缓存对不同交易到达率的AND / OR背书策略的吞吐量和延迟的影响。表三列出了各种策略以及不同的交易到达率。我们将读者的注意力吸引到图6（b），以便与无缓存行为进行比较。平均而言，由于MSP缓存与吞吐量相比吞吐量增加了3倍。例如，当背书策略需要两名背书人（使用AND/OR语法定义）的签名时，没有MSP缓存的情况下达到的最大吞吐量为160tps，而缓存则增加到540tps。这是因为MSP缓存减少了某些重复性的CPU密集型操作。

图14和图15分别显示了vanilla peer和MSP高速缓存peer的密码操作频率和VSCC验证阶段的调用堆栈深度

的火焰图。可以看出，使用MSP缓存可以显着减少加密操作和调用堆栈深度的数量。

B. 区块的VSCC并行验证

VSCC验证阶段根据背书策略连续验证每个交易。由于这种方法未充分利用资源，我们研究了并行验证的效率，即并行验证多个交易的背书，以便利用空闲的CPU并提高整体性能。为此，我们在peer启动时为每个通道创建了一个可配置数量的工作线程。每个工作线程根据其背书策略验证一笔交易的背书签字。

图16绘出了并行VSCC对性能和资源利用率的影响。我们将不同通道数的到达率分为两类;当延迟落在[0.1-1s]时为非过载情况，当延迟落在[30-40s]时为过载情况。对于每个通道，我们分配了等于区块大小的工作线程。在非过载情况下，一个通道的吞吐量和资源利用率从130 tps到800 tps（对于30的区块大小改进了6.3倍），对于300的区块大小，达到了980 tps（7.5×）。这是由于并行验证并因此减少了VSCC等待时间（从300ms减少到30ms，即对于30的区块大小减少了10倍）。吞吐量

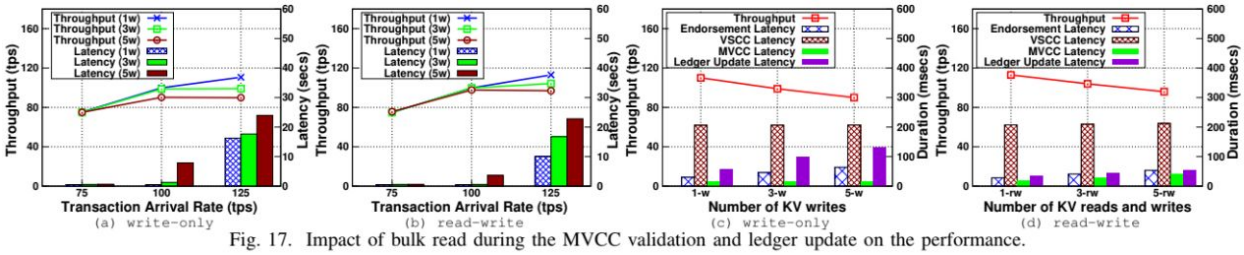


Fig. 17. Impact of bulk read during the MVCC validation and ledger update on the performance.

在950 TPS（对于30块大小）和1075 TPS（对于300块大小）达到饱和 -参见图16的（a）。

同样地，在过载的情况下，吞吐量和资源利用率在16个通道中增加到1.5倍，而在1个通道中增加到10倍 - 参见图16（b）。这表明对一个区块的并行VSCC验证显着提高了单个通道的性能。随着通道数量的增加，改善比例下降。这是因为默认情况下多个通道会导致区块的并行验证（而不是交易），因此少量的空闲vCPU可用于并行VSCC。

C. 在MVCC验证和提交时的批量读写

在MVCC验证期间，使用CouchDB作为状态数据库，对于区块中的每个交易，对于交易的读取集合中的每个key，通过对数据库的GET REST API调用安全的HTTPS检索最近提交的版本号。在提交阶段，对于区块中的每个有效交易（在MVCC验证后记录），对于写交易集中的每个key，GET REST API调用检索修订号[4]。最后，对于写集中的每个条目，PUT REST API调用都会提交该文档。由于这些多个REST API调用，性能显着降低，如§VD所示。

为了减少REST API调用的次数，CouchDB建议使用批量操作。因此，我们使用Fabric中现有的BatchRetrieval API，通过每个区块的单个GET REST API调用将多个key的版本号和修订号批量加载到缓存中。为了增强账本更新过程，我们使用Fabric中的BatchUpdate API来为每个区块使用单个PUT REST API调用提交一批文档。此外，我们在VSCC中引入了一个缓存来减少对CouchDB的调用，以获得每个交易的链码的背书策略。在本节中，我们将展示这些增强对整体性能的有效性。图17显示了使用大容量读/写优化将CouchDB作为状态数据库运行时的吞吐量和延迟。与非批量读/写比较，请参见图11（b）和图12.对于单次写入交易，性能从50 tps显着提高至115 tps（即2.3倍）。对于多次写入（3-w & 5-w），吞吐量从26 tps增加到100 tps（即3.8倍，3 w），18 tps到90 tps（即5 w 5 w）。我们注意到对读写交易的类似改进。

由于批量读/写优化，与图12相比，MVCC延迟、账本更新延迟和背书延迟减少，如图17（c）和（d）所示。

TABLE IX
CONFIGURATION TO IDENTIFY THE IMPACT OF ALL THREE OPTIMIZATIONS COMBINED.

Parameters	Values
Number of Channels	1, 8, 16
Transaction Complexity	1 KV write (1-w) of 20 bytes
StateDB Database	GoLevelDB, CouchDB
Peer Resources	32 vCPUs, 3 Gbps link
Endorsement Policy	1^{st} and 3^{rd} AND/OR policies
Block Size	100, 300, 500 (#tx)
#VSCC Workers per Channel	Equal to the block size

背书延迟减少（至少3倍），这是因为锁定持续时间缩短了至少8倍。由于批量读取块中所有交易的读取集中的所有key，因此读写交易的MVCC延迟减少（至少6倍）。请注意，MVCC延迟随着批量读取中读取的key数量的增加而增加。包含更多只写交易的区块的账本更新延迟更高。这是因为，在读写交易中，MVCC验证阶段本身将所需修订号加载到缓存中（因为交易在修改之前读取了这些key），而只写交易并非如此。

D.优化组合

图18显示了所有三种优化结合后的性能改进。表IX显示了每个通道的VSCC工作线程数，区块大小以及用于本研究的其他相关参数。

使用GoLevelDB作为状态数据库，由于全部三种优化，单通道吞吐量从140 tps提高到2250 tps（即16倍提升）- 参见图18（a）和图4。同样，以CouchDB作为状态数据库，单通道吞吐量从50 tps增加到700 tps（即14倍的改进）- 参见图18（b）和图11。随着区块大小的增加，当CouchDB是状态数据库时，我们观察到较低由于对CouchDB的大量REST API调用的数量减少（即对于500个交易，只有2个批量REST API调用，在MVCC阶段进行一次读取调用，并且在提交阶段进行一次写入），导致总延迟时间为区块大小为500，而区块大小为100的10个批量REST API调用相比）。因此，我们的准则1和2不适用于具有批量读取/写入优化的CouchDB。

此外，对于8和16通道，吞吐量从1025 tps和1321 tps分别增加到2700 tps，如图19（a）所示。通过更简单的背书策略，即第1个

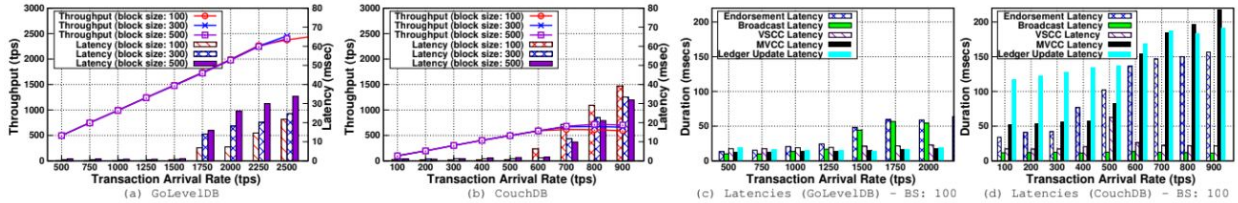
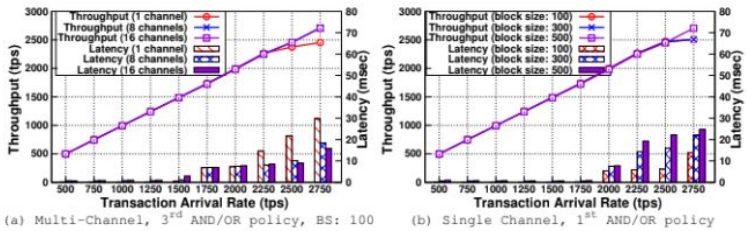


Fig. 18. Impact of all the three optimizations on the performance with different block sizes.



AND/OR策略，单通道吞吐量也增加到2700 tps（使用GoLevelDB），如图19（b）所示。

即使吞吐量为2700 tps，peer的平均CPU利用率也只有60%，网络利用率的一个节点是1680 Mbps（发送）和240 Mbps（接收）。这是因为MVCC延迟和账本更新延迟（较少的CPU密集型任务）的总和几乎等于或高于VSCC延迟（如图18所示c）和（d））。由于这些连续阶段，vCPU没有得到充分利用。一个潜在的优化是将VSCC和MVCC验证阶段管道化。

VII. 相关工作

对公共区块链网络的伸缩性和性能特征以及共识协议及其安全影响的限制因素已有相当大的兴趣[31]，[23]。

对于公共区块链，[19]也研究了基于公开可用数据量化比特币网络的吞吐量、延迟、引导时间和每次交易成本。

BlockBench [22]是第一批查看权限区块链的人之一。他们提出了一个框架，使用一套微观和宏观基准来比较不同区块链平台的性能，即以太坊、Parity和Hyperledger Fabric。与[19]类似，他们将共识、数据、执行和应用概括为4层区块链，并使用基准来运用它们。他们根据吞吐量、平台的延迟和伸缩性来衡量整体性能，并在3个平台上得出结论。然而，他们研究了Fabric v0.6的性能，v1.0版本引入了一个完整的重新设计，他们的观察不具有相关性，需要重新研究。

[16]介绍了Fabric的设计和新架构，深入研究了其设计考虑因素和模块性。它介绍了像Fabric这样的加密货币应用的一个比特币的性能，称为Fabcoin，它使用定制的VSCC来验证Fabcoin特定的交易并避免复杂的背书和通道。此外，它们用来CLI命令仿真客户端而不是使用SDK [6]，[8]，[10]，这是不现实的。我们的工作与他们的不同之处在于，我们针对不同的工作负载进行了全面的研究，使Fabric的模块化和应用在多个领域得到了关注。给读者的一个注释，[16]使用了Fabric v1.1-preview发行版，它包含了我们在v1.0上的所有优化和其他附加功能。但是，作为次要版本更新核心功能的大部分仍然是相同的，我们的观察结果适用于基于Fabric新架构的v1.1和未来版本。

VIII. 结论和未来的工作

在本文中，我们进行了一项全面的实证研究，以了解Hyperledger Fabric的性能，一个许可的区块链平台，通过改变分配给可配置参数的值，例如区块大小、背书策略、通道、资源分配和状态数据库选择。作为我们研究的结果，我们提供了六个有关配置这些参数的宝贵指南，并确定了三个主要的性能瓶颈。因此，我们在MVCC验证和提交阶段引入并研究了三个简单的优化，例如MSP缓存、并行VSCC验证和批量读/写，以将单个信道性能提高16倍。此外，这三项优化已成功应用于Fabric v1.1。

作为未来工作的一部分，我们将通过使用不同的区块链拓扑来研究Fabric的伸缩性和容错能力，例如不同组织数量和每个组织不同数量的节点。此外，我们计划量化各种共识算法和排序服务中的节点数量对不同工作负载性能的影响。在我们的研究中，我们假设网络不是瓶颈。但是，在现实世界部署中，节点可以在地理上分布，因此网络可能起作用。此外，现实世界生产系统的到达率将遵循某些分布。因此，我们将研究具有不同到达率分布的Fabric在广域网（WAN）中的性能。

IX. 致谢

我们计划量化各种共识算法和排序服务中的节点数量对不同工作负载性能的影响。在我们的研究中，我们假设网络不是瓶颈。但是，在现实世界部署中，节点可以在地理上分布，因此网络可能起作用。此外，现实世界生产系统的到达率将遵循某些分布。因此，我们将研究具有不同到达率分布的Fabric在广域网（WAN）中的性能。我们希望感谢以下同事对我们工作的宝贵帮助。
Angelo De Carlo（MSP Cache），Alessandro Sorniotti（并行验证）。感谢David Enyeart，Chris Elder，

Manish Sethi提出的关于CouchDB批量阅读的提案。我们要感谢Yacov Manevich的一贯帮助。

策略

参考文献

[1] Chaincodes. http://hyperledger-fabric.readthedocs.io/en/release-1.1/chaincode4noah.html. [Online; accessed 1-May-2018].

[2] CouchDB. http://couchdb.apache.org/. [Online; accessed 1-May-2018].

[3] CouchDB: Bulk API. http://docs.couchdb.org/en/2.0.0/api/database/bulk-api.html. [Online; accessed 1-May-2018].

[4] CouchDB: Document Revision Number. http://docs.couchdb.org/en/2.0.0/intro/api.html?highlight=revision#revisions. [Online; accessed 1-May-2018].

[5] Everledger — A Digital Global Ledger. https://www.everledger.io/. [Online; accessed 1-May-2018].

[6] Go SDK for Fabric Client/Application. https://github.com/hyperledger/fabric-sdk-go. [Online; accessed 1-May-2018].

[7] GoLevelDB. https://github.com/syndtr/goleveldb. [Online; accessed 1-May-2018].

[8] Java SDK for Fabric Client/Application. https://github.com/hyperledger/fabric-sdk-java. [Online; accessed 1-May-2018].

[9] Membership Service Providers (MSP). http://hyperledger-fabric.readthedocs.io/en/release-1.1/msp.html. [Online; accessed 1-May-2018].

[10] Node SDK for Fabric Client/Application. https://github.com/hyperledger/fabric-sdk-node. [Online; accessed 1-May-2018].

[11] PostgreSQL as the State Database for Hyperledger Fabric. https://jira.hyperledger.org/browse/FAB-8031. [Online; accessed 1-May-2018].

[12] PostgreSQL Database Management System. https://github.com/postgres/postgres. [Online; accessed 1-May-2018].

[13] The Linux Foundation Helps Hyperledger Build the Most Vibrant Open Source Ecosystem for Blockchain. http://www.linuxfoundation.org/. [Online; accessed 1-May-2018].

[14] Quorum: an Ethereum-forked variant Blockchain. https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf, 2016. [Online; accessed 1-May-2018].

[15] SecureKey: Building Trusted Identity Networks. https://securekey.com/, 2017. [Online; accessed 1-May-2018].

[16] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. Weed Cocco, and´ J. Yellick. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. ArXiv e-prints, Jan. 2018.

[17] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ansi sql isolation levels. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, SIGMOD '95, pages 1–10, New York, NY, USA, 1995. ACM.

[18] M. Castro and B. Liskov. Practical byzantine fault tolerance. In Proceedings of the Third Symposium on Operating Systems Design and Implementation, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

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

[20] C. Dannen. Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners. Apress, Berkely, CA, USA, 1st edition, 2017.

[21] D. D. Detwiler. One nations move to increase food safety with blockchain. https://www.ibm.com/blogs/blockchain/2018/02/one-nations-move-to-increase-food-safety-with-blockchain/, 2018. [Online; accessed 1-May-2018].

[22] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan. Blockbench: A framework for analyzing private blockchains. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, pages 1085–1100, New York, NY, USA, 2017. ACM.

[23] A. Gervais, G. O. Karame, K. Wust, V. Glykantzis, H. Ritzdorf,¨ and S. Capkun. On the security and performance of proof of work

blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pages 3–16. ACM, 2016.
[24] J. Gottlieb, E. Marchiori, and C. Rossi. Evolutionary algorithms for the satisfiability problem. Evol. Comput., 10(1):35–50, Mar. 2002.
[25] M. Hearn. Corda: A distributed ledger. https://docs.corda.net/ static/ corda-technical-whitepaper.pdf, 2016. [Online; accessed 1-May-2018].
[26] F. Keller. New collaboration on trade finance platform built on blockchain. https://www.ibm.com/blogs/blockchain/2017/10/ new-collaboration-on-trade-finance-platform-built-on-blockchain/, 2017. [Online; accessed 1-May-2018].
[27] N. Megiddo and D. S. Modha. Arc: A self-tuning, low overhead replacement cache. In Proceedings of the 2Nd USENIX Conference on File and Storage Technologies, FAST '03, pages 115–130, Berkeley, CA, USA, 2003. USENIX Association.
[28] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf.

[29] S. Omohundro. Cryptocurrencies, smart contracts, and artificial intelligence. AI Matters, 1(2):19–21, Dec. 2014.

[30] C. H. Papadimitriou and P. C. Kanellakis. On concurrency control by multiple versions. ACM Trans. Database Syst., 9(1):89–99, Mar. 1984.
[31] M. Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs.´ bft replication. In International Workshop on Open Problems in Network Security, pages 112–125. Springer, 2015.
[32] M. Vukolic. Rethinking permissioned blockchains. In´ Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, BCC '17, pages 3–7, New York, NY, USA, 2017. ACM.
[33] M. White. Digitizing Global Trade with Maersk and IBM. https://www.ibm.com/blogs/blockchain/2018/01/ digitizing-global-trade-maersk-ibm/, 2018. [Online; accessed 1-May-2018].