МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

АЛГОРИТМЫ КОНСЕНСУСА. RAFT

КУРСОВАЯ РАБОТА

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Ахмановой Элины Дамировны

Научный руководитель
к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развитие человека как вида непосредственно связано с увеличением размера социальных групп. Многие люди не могут представить себя как отшельника или участника закрытой общины на 50 человек. Эволюция компьютерных систем имеет похожую историю: монолитные однопроцессорные системы уходят в прошлое, уступив место гибким масштабируемым распределённым системам.

Особенно остро изменения в многопоточных и распределённых системах ощущаются с распространением микропроцессорной технологии (как итог компьютеры стали более мощными и дешёвыми), ростом использования сети Интернет и увеличением скорости обмена информацией. Это повлияло на нарастание привлекательности горизонтального масштабирования распределённых систем (увеличения числа узлов) вместо вертикального (замена устройств на более мощные).

Несмотря на общую дешевизну и простоту увеличения числа узлов, горизонтельное масштабирование нуждается в строгой системе управления распределёнными вычислениями между большим числом узлов. Узлы в управляемой системе должны либо состоять в строгой иерархии, либо поддерживать консенсус равноправных элементов.

Алгоритмы консенсуса являются одним из способов организации сообщения узлов в децентрализованных распределённых системах. Эти алгоритмы лежат в основе разработки банковских систем [Distributed Online Banking Mahmood Akhtar], баз данных [Big table], болкчейн систем [mastering blockchain] и пр.

В данной работе основное внимание будет посвящено именно алгоритмам консенсуса, и, в частности, алгоритму Raft. Цель работы: изучить алгоритмы консенсуса и реализовать распределённую систему с помощью алгоритма Raft.

Задачи работы:

— изучить модель логических часов;
— выделить особенности основных алгоритмов консенсуса;
— реализовать распределённую, реплицированную, устойчивую к непредвиденным сбоям систему с помощью алгоритма Raft.

В первой части будут рассмотрены теоретические понятия логических часов, консенсуса; также будет рассмотрены модель логических часов Лампорта

и особенности алгоритмов консенсуса. Мы постараемся объяснить, чем был обусловлен выбор алгоритма Raft для последующей реализации.

Во второй части будет реализована распределённая система и предложена базовая реализация для принятия решения несколькими узлами с учётом возможных сбоев сети (уменьшение скорости передачи данных на некоторых узлах, разделение сети на изолированные подсети, отключение некоторых узлов от сети) и выхода из строя некоторых узлов.

Работа состоит из содержания, введения, двух глав, заключения и списка использованных источников.

1 Алгоритмы консенсуса в распределённых системах

1.1 Распределённые системы

Когда мы говорим о распределённых алгоритмов, мы подразумеваем, что это алгоритмы для распределённых систем.

Распределённые системы - это парадигма, согласно которой два или более компьютера (узла) сообщаются между собой с целью достичь общего соглашения. Конечный пользователь видит соглашение как единую логическую платформу [13].

Каждый узел может быть определён как индивидуальный игрок в распределённой системе. Все узлы способны отправлять и получать сообщения друг от друга.

Существует несколько состояний надёжности передаваемой информации между узлами: узлы могут быть точны в отправке информации, неисправны, а также отправлять заведомо неверную информацию. Кроме поведения, узлы наделены собственной памятью и процессором.

Узел, который был замечен в выставлении произвольной или заведомо неверной информации, называется **византийским узлом**, а проблема организации взаимодействия между несколькими узлами, некоторые из которых могут быть неисправны, называется задачей византийских генералов (англ. Byzantine generals problem) или византийская задача отказоустойчивости (англ. Byzantine fault tolerance) [14]. Такое поведение узла может быть не только признаком неисправности, но также и признаком злоумышленного поведения, которое может привести к неполадкам во всей распределённой сети. Таким образом, непредвиденное поведение узла в распределённой сети можно назвать **византийским**.

1.2 Синхронизация узлов распределённой системы. Понятие логических часов. Часы Лампорта

Одно из фундаментальных понятий распределённых систем - это часы Лампорта. Распределённые системы состоят из множества отдельных процессов, которые упорядочены и разделены в пространстве, и которые могут сообщаться между собой с помощью обмена сообщениями.

Концепция времени фундаментальна для человеческого способа мышления. Она исходит от более базовой концепции упорядоченности событий.

Мы говорим, что событие a произошло раньше события b в том случае, если на временной прямой, идущей слева направо событие a находится строго левее события b.

К сожалению, в распределённых системах невозможно определить, какое из двух событий произошло первым, если время задержки между сообщениями может быть больше, чем время между двумя последовательно возникающими событиями. Такое отношение как "произошло прежде"поэтому может лишь частично упорядочивать события в системе.

Предположим, что система состоит из конечного набора процессов. Каждый процесс содержит последовательность событий.

PICTURE 1

Предположим, что события в процессе такие, что событие a произошло прежде события b. Также предположим, что отправка или получение сообщения - это события в процессе. Таким образом, мы можем определить отношение "произошло прежде"($\Rightarrow$) как множество событий системы, удовлетворяющее условиям [1]:

1) Если a и b - события одного процесса, и a произошло раньше b, тогда $a \Rightarrow b$.

2) Если a - это отправка сообщения от одного процесса к другому, и b - это получение этого же сообщения другим процессом, тогда $a \Rightarrow b$.

3) Если $a \Rightarrow b$ и $b \Rightarrow c$, тогда $a \Rightarrow c$. Два процесса a и b называют параллельными, если $a \not\Rightarrow b$ и $b \not\Rightarrow a$.

На рисунке 1 вертикаль моделирует время: более позднее событие находится выше более раннего. Точки определяют события, вертикальные прямые - процессы.

Теперь добавим в систему логические часы (часы). Определим часы C_i для каждого процесса P_i как функцию с числами $C_i(a)$ для каждого события a в этом процессе. Система часов представляет из себя функцию C, которая определяет для каждого события b число $C(b)$, где $C(b) = C_j(b)$ в том случае, если b - событие процесса P_j [1].

Условие для часов: Для любого события a, b: если $a \Rightarrow b$, тогда $C(a) < C(b)$.

Условие 1. Если a и b - события процесса P_i, и a произошло прежде события b, тогда $C_i(a) < C_i(b)$.

Условие 2. Если a - это отправка сообщения от процесса P_i, и b - это получение того же сообщения процессом P_j, тогда $C_i(a) < C_j(b)$.

PICTURE 2

Рассмотрим рисунок 2. На нём часы представлены в терминах пространственно-временной диаграммы. Представим, что у каждого процесса есть атомарные тики, которые происходят между событиями процесса P_i с $C_i(a) = 4$ и $C_i(b) = 7$. Тогда тики 5, 6 и 7 на часах происходят между двумя событиями. На рисунке 2 каждый тик изображён как пунктирная линия.

Условие 1 означает, что здесь должен быть тик между двумя событиями в процессе.

Условие 2 означает, что каждая линия сообщений должна пересекать линию тика.

Для того, чтобы гарантировать, что часы в системе удовлетворяют условиям, описанным выше (Условие 1 и Условие 2).

Условие 1 можно считать достаточно простым: процессам необходимо лишь подчиняться правилам реализации [1]:

Правило реализации 1.

Каждый процесс P_i увеличивает C_i на 1 для каждого нового успешного события. Для описания следующего правила потребуем, чтобы каждое сообщение m содержало временную отметку T_m, которая эквивалентна времени, в которое сообщение было отправлено. Правило реализации 2.

а) Если событие a - это отправка сообщения m процессом P_i, тогда сообщение m содержит временную отметку $T_m = C_i(a)$.

б) Для любого процесса P_j временная отметка C_j получения сообщения m больше либо равна T_m.

Правило реализации 2.

а) Если событие a - это отправка сообщения m процессом P_i, тогда сообщение m содержит временную отметку $T_m = C_i(a)$.

б) Для любого процесса P_j временная отметка C_j получения сообщения m больше либо равна T_m.

С помощью условий и правил реализации логических часов мы смогли выяснить концепцию словосочетания "произошло прежде которое определяет инвариант порядка сообщений в распределённой системе с несколькими процессами. С точки зрения распределённых алгоритмов наиболее важным

является факт относительной упорядоченности событий.

1.3 Понятие консенсуса

В этой части мы рассмотрим проблему консенсуса [15]. Авторы работы "Распределённые системы: концепции и дизайн"[5] рассматривают консенсус как проблему *соглашения*.

Модель системы

Модель системы включает множество процессов $p_i (i = 1, 2, ..., N)$, сообщающихся благодаря отправке сообщений. Важное требование: соглашение должно быть достигнуто даже при возможных ошибках, выходе из строя одного или нескольки процессов. Предположим, что сообщение между процессами надёжное, но процессы всё ещё могут выйти из строя. Мы будем рассматривать Византийские сбои наравне с прочими.

Определение проблемы консенсуса Для того, чтобы достичь консенсуса, каждый процесс p_i в *не принявший решений* состоянии *предлагает* единственное значение v_i из набора значений $D (i = 1, 2, ..., N)$. Процессы общаются друг с другом с помощью обмена значений. Каждый процесс устанавливает значение в *переменную решения*, d_i. Это приводит к изменению состояния процесса на состояние *принявшего решение*. На рисунке 3 изображен описанный процесс. Два процесса предлагают принять значение, третье предлагает отменить и затем выходит из строя. Два процесса решают продолжить.

PICTURE 3 661

Условия для выполнения алгоритма консенсуса [5]:

Прерывание: Случай, когда для каждого процесса выставленно значение переменной решения.

Соглашение: Значение решения для всех корректно работающих процессов одно и то же: если p_i и p_j работают корректно и для них введены значения, то $d_i = d_j (i, j = 1, 2, ..., N)$.

Целостность: Все корректные процессы предлагают одинаковое значение. То есть любой корректный процесс находится в состоянии *принявшего решение* и выбрал одинаковое с другими значение.

В определении целостности могут встречаться вариации, согласно приложению условия. К примеру, более слабый тип целостности может предполагать, что *переменную решения* приняло какое-то число процессов - не обязательно даже всех. Так же условие целостности могут называть

условием *доказанности*.

Для того, чтобы сопоставить формулировку проблемы и алгоритм, предположим систему, которая не может давать сбоев. Это в тот же самый момент решает консенсус. К примеру, мы можем объединить процессы в группы и реализовать надёжную широковещательную рассылку сразу всем остальным процессам в группе. Каждый процесс ждёт, пока он соберёт все N значений со всех процессов (включая его самого). Это определяет функцию $большинства(v_1, v_2, ..., v_N)$, которая возвращает значение наиболее частого аргумента или специальное значение $\perp \notin D$, если большинство не достигнуто.

Прекращение гарантируется надёжностью рассылки. Соглашение и целостность гаратируются благодаря опредлелению большинства и и целостности значения при рассылки. Каждый процесс получает одинаковый набор значений, каждый процесс использует одинаковый функционал для этих значений. Таким образом, все они находятся в соглашении, и, если каждый процесс предлагает одинаковое значение, они все решают установить его.

Заметим, что функция большинства - единственный способ соглашения о значении между процессами.

Если процессы могут выходить из строя, это может принести проблемы, и процесс принятия соглашения будет прерван. Как известно, часто для запуска процессов используются узлы в виде компьютеров, которые могут выходить из строя полностью или частично, и даже стать византийскими по стечению обстоятельств.

Если процессы выходят из строя в случайном (возможно, византийском) порядке, то "поломанные"процессы могут сообщать случайные значения друг для друга. Это, конечно, выглядит неправдоподобно, что ошибка в системе может породить подобный алгоритм консенсуса, но с совершенно другими значениями. Несмотря на это, сбой может быть не случайными, но являться результатом злонамеренного умысла. Кто-то может выборочно изменить процессы, и заставить их отправлять новые данные на не вышедшие из строя узлы, в попытке расстроить достижение консенсуса. В случае неконсистентности, корректные процессы могут сравнить полученные значения с ожидаемыми.

Проблема византийских генералов

В неформальном описании проблема византийских генералов [15] звучит как: три или более генералов согласились атаковать или отступать. Один,

коммандир, отдаёт приказ. Остальные, лейтенанты коммандира, должны решить, нужно атаковать или оступать. Но один или более генералов может быть "предателем". Если коммандир предатель, он предложет атаковать одному генералу и отступать другим. Если же один из лейтенантов предатель, он может сообщить одним, что коммандир приказал атаковать, а другим - что отступать.

Проблема византийских генералов отличается от консенсуса в том, что отличающийся процесс подчиняется всем остальным, если они согласны на одном значении, вместо того чтобы каждому предлагать значение.

Прерывание Каждый случайный процесс создаёт его собственную переменную принятия решения.

Соглашение Переменная принятия решения одна для всех процессов: если p_i и p_j работают исправно, тогда состояние соглашения такое, что $d_i = d_j (i, j = 1, 2, ..., N)$.

Целостность Если командир исправен, тогда все корректные процессы приходят к соглашению с тем, что командир предложил.

Заметим, что для проблемы византийских генералов целостность предполагает соглашение, когда командир исправен, но командир не обязан работать исправно.

1.4 Поняте кворума

2 Алгоритмы консенсуса в распределённых системах

2.1 Алгоритмы консенсуса

В данном разделе мы рассмотрим алгоритмы консенсуса, обозначим плюсы и минусы каждого из них [20].

Доказательство работы (Proof of Work)

Этот алгоритм считается первым, и был представлен Сатоши Накамото [16] для создания распределённого консенсуса, который априори не доверяет узлам. Доказательство работы - не новая идея, но то, как Сатоши объединил уже сущетсвующие идеи (криптографические подписи, дерево хешей и одноранговые компьютерные сети peer-to-peer) в самодостаточную распределённую систему консенсуса, которая была достаточно инновационной для первого приложения криптовалют.

Это работает так, что участники блокчейна должны решить **сложную,**

но бесполезную вычислительную проблемы и добавить новую транзакцию как блок в блокчейн к другим. Аналогично с работниками шахт, которые получают ресурсы в процессе добычи (майнинг от англ. mining). Это сделано, чтобы удостовериться, что участники готовы тратить деньги и ресурсы (вычислитель мощности) на работу, то есть они не хотят нанести вред системе. Нанесение вреда ведёт к потере этих ресурсов, что должно нанести вред самим злоумышленника

Сложность проблемы может быть изменена в режиме реального времени, опираясь на постоянный промежуток времени для вычисления. Иногда может произойти ситуация, в которой один или более участников решают проблему в одно и то же время. В этом случае, участники должны выбрать цепочку действий, и самая долгая цепочка выигрывает. Так, предположим, что большая часть участников работает над одним вычислением. Тот участник, чьё вычисление вырастет быстрее и чья цепочка вычислений будет самой долгой, является доверительным. Вычисления безопасны до тех пор, пока более 50% участников честны.

Плюсы:

— Система активно тестируется с 2009 года и продолжает активно использоваться по сей день.

Минусы:

— Скорость работы (алгоритм медленный);

— Тратит много ресурсов, что скажется на скорости выхода из строя компонентов;

— Подвержен эффекту масштабирования.

Используется в: Bitcoin [17], Ethereum [18], Letecoin [19] и других.

Тип: консенсус с конкурированием.

Доказательство доли владения (Proof of Stake)

Этот алгоритм принадлежит к алгоритмам соревнующегося консенсуса. Он был создан как альтернатива алгоритма доказательства работы, чтобы решить возникающие в нём проблемы. Здесь вместо того, чтобы использовать майнинг, участники должны иметь некоторую ставку, долю (монеты) в системе. Итак, если у участника есть 10% монет, то вероятность майнинга в следующем блоке составит 10%.

Майнинг требует больших вычислительных мощностей для выполнения различных криптографических вычислений, благодаря которым можно открыть

новые вычислительные задачи.

Вычислительная мощность выражается тратами на электроэнергию, на сами устройства, необходимые для подтверждения работы. Если же здоумышленник атакует систему, то он теряет свою ставку. Одна из проблем, которая может возникнуть - это проблема "ничего на кону когда участникам вычислений нечего терять, они голосуют за множественные варианты цепочек блоков (вилки), тем самым предотвращая достижение консенсуса. Это изображено на рисунке 4.

PICTURE 4

В отличие от систем проверки работоспособности (где для расширения цепочки приходится выполнять много вычислений), работа с несколькими цепочками не требует больших затрат. Многие проекты пытались решить проблему "ничего на кону"по-разному, одним из решений является наказание тех, кто голосует за множественные вилки.

Плюсы:

— Энергетически эффективный алгоритм;
— Каждая новая атака стоит дороже для злоумышленника;
— Не подвержен эффекту масштабирования.

Минусы:

— Подвержен эффекту "ничего на кону".

Используется в: Ethereum (разрабатывается) [18], Peercoin [21] и других.

Тип: консенсус с конкурированием.

Отложенное доказательство работы (Delayed Proof-of-Work)

Отложенное подтверждение работы - это гибрид консенсуса за счёт безопасности и вторичного хеширования. Консенсус достигается с помощью группы подтверждающих узлов, которые добавляют данные с первой цепочки во вторую, что потребует атаки на обе цепочки, чтобы разрушить хотя бы одну из них.

Для работы сети вычислений можно использовать оба предыдущих алгоритма. Данный алгоритм присоединяется к любой желаемой цепочке алгоритма доказательство работы, как изображено на рисунке 5

PICTURE 5

В системе есть два типа узлов: подтверждающие (notary node) и обычные (common node, node). Подтверждающие узлы выбираются для уже согласованных

узлов цепочки в присоеденённую цепь блоков.

Чтобы предотвратить войны майнинга между подтверждающими узлами (которые могут снизить эффективность), для первого пользователя алгоритма, системы Komodo [22], был разработан метод майнинга с циклическим перебором, который работает в двух режимах. Режим "без подтверждения"позволяет всем сетевым узлам добывать блоки, аналогично традиционному алгоритму доказательство работы. Однако в режиме "активное подтверждение"сетевые подтверждающие узлы будут добывать со значительно меньшей степенью сложности. В рамках этой схемы главному подтверждающему узлу разрешено добывать один блок со своей текущей степенью сложности, в то время как другие подтверждающие узлы должны добывать блок со сложностью в 10 раз выше, а все обычные узлы - со сложностью в 100 раз выше.

Это приводит к новой проблеме: большие различия между сложностью хешей подтверждающих узлов и обычных.

Плюсы:

— Энергетически эффективный алгоритм;
— Повышенная безопасность;
— Повышение ценности некоторых блоков цепочки.

Минусы:

— Ограничение на тип алгоритмов, на которых может базироваться (только доказательство работы и доказательство доли владения).

Используется в: Komodo [22], Blockchain [17].

Тип: консенсус с коллаборацией.

2.2 Raft

Raft - это консенсусный алгоритм, разработанный как альтернатива Paxos . Он должен был быть более понятным, чем Paxos, посредством разделения логики, но он также формально доказал свою безопасность и предлагает некоторые дополнительные функции. Raft предлагает общий способ распределения конечного автомата по кластеру вычислительных систем, гарантируя, что каждый узел в кластере согласовывает одну и ту же серию переходов состояний. Он имеет ряд реализаций ссылок с открытым исходным кодом, с реализациями полной спецификации на Go , C ++ , Java и Scala .

Рафт достигает консенсуса через избранного лидера. Сервер в кластере является либо лидером, либо последователем , и может быть кандидатом в

конкретном случае выборов (лидер недоступен). Лидер отвечает за репликацию логов подписчикам. Он регулярно информирует последователей о своем существован отправляя сообщение сердцебиения. У каждого последователя есть тайм-аут (обычно от 150 до 300 мс), в течение которого он ожидает сердцебиение от лидера. Таймаут сбрасывается при получении пульса. Если пульс не получен, последователь меняет свой статус на кандидата и начинает выборы лидера.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Moscow Institute of Physics and Technology Специализация: Машинное обучение и анализ данных. Курс: Обучение на размеченных данных [Электронный ресурс]. — URL: https://ru.coursera.org/lecture/supervised-learning/rieshaiushchiie-dieriev-ia-HZxD1 (Дата обращения 20.05.2018). Загл. с экр. Яз. рус.

2 *Quinlan, J. R.* C4.5: Programs for Machine learning / J. R. Quinlan. — San Mateo: Morgan Kaufmann Publishers, 1993. — 302 pp.

3 *Quinlan, J. R.* Learning logical definitions from relations. Machine Learning / J. R. Quinlan. — 1990. — 239-266 pp.

4 *Н. Соловьев А. Семенов, А. Ц. Е. Ч.* Интеллектуальные системы / А. Ц. Е. Ч. Н. Соловьев, А. Семенов. — Оренбург, 2013. — 236 с.

5 Лекции Ульяновского государственного университета [Электронный ресурс]. — URL: https://studfiles.net/preview/6172591/page:10 (Дата обращения 20.05.2018). Загл. с экр. Яз. рус.

6 *Чубуков, И. Ф.* Data Mining. Учебный курс [Электронный ресурс] / И. Ф. Чубуков. — URL: http://www.intuit.ru/department/database/datamining/ (Дата обращения 20.05.2018). Загл. с экр. Яз. рус.

7 *Breiman, L.* Classification and regression trees / L. Breiman, J. H. Friedman, C. J. Olshen, R. A. ands Stone. — Belmont, CA: Wadsworth International Group, 1984. — 354 pp.

8 *Сенько, О. В.* Курс «Математические основы теории прогнозирования» Лекция 8 Решающие деревья / О. В. Сенько. — МОТП, 2017. — 15 с.

9 *Сирота, А. А.* Методы и алгоритмы анализа данных и их моделирование в MATLAB / А. А. Сирота. — Петербург: БХВ-Петербург, 2017. — 384 с.

10 *Sen, A.* 16 diagrams On Economic Inequality / A. Sen. — Oxford: Oxford University Press, 1997. — 280 pp.

11 *Shannon, C. E.* A mathematical theory of communication / C. E. Shannon // *Bell System Technical Journal*. — 1948. — Pp. 379–423.

12 *Bies, R. R.* A genetic algorithm-based, hybrid machine learning approach to model selection / R. R. Bies, M. F. Muldoon, B. G. Pollock, S. Manuck, G. Smith, M. E. Sale // *Journal of Pharmacokinetics and Pharmacodynamics.* — 2006. — Pp. 196–221.

13 *Berger, R. L.* Statistical Inference / R. L. Berger, G. Casella. — Duxbury Press, 2001. — 374 pp.

14 *Hastie, T.* The elements of statistical learning / T. Hastie, R. Tibshirani, J. Friedman // *Springer.* — 2001. — Pp. 269–272.

15 *Quinlan, J. R.* Machine Learning: an artificial intelligence approach. Learning efficient classification procedures / J. R. Quinlan. — Carbonell Mitchell, 1983. — Pp. 463–482.

16 *Quinlan, J. R.* Improved use of continuous attributes in c4.5 / J. R. Quinlan // *Journal of Artificial Intelligence Research.* — 1996. — Pp. 77–90.

17 *Мартин, Н.* Математическая теория энтропии / Н. Мартин, Д. Ингленд. — Москва: Мир, 1988. — 350 pp.

18 *Шеннон, К.* Работы по теории информации и кибернетике / К. Шеннон. — Москва, 1963. — С. 243–332.

19 *Уикем, Х.* Язык R в задачах науки о данных: импорт, подготовка, обработка, визуализация и моделирование данных / Х. Уикем, Г. Гроулмунд. — Вильямс, 2017. — 592 с.

20 *Коэльё, Л. П.* Построение систем машинного обучения на языке Python / Л. П. Коэльё, В. Ричерт. — Москва: ДМК Пресс, 2016. — 202 с.

21 *Жерон, О.* Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем / О. Жерон. — Вильямс, 2018. — 688 с.

ПРИЛОЖЕНИЕ А
Листинг программы