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

АЛГОРИТМ ДЛЯ РЕШЕНИЯ ЗАДАЧ КОНСЕНСУСА В СЕТИ НЕНАДЁЖНЫХ ВЫЧИСЛЕНИЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Соколкова Павла Вячеславовича

Научный руководитель
доцент, к.ф.-м.н. _____ Г. Г. Наркайтис

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

Саратов 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Практически в любых современных организациях много внимания уделяется информатизации, построению надёжной сетевой инфраструктуры. Постоянно увеличиваются объёмы данных, которые нужно обрабатывать в режиме реального времени. Порой объём поступающей информации превышает скорость её обработки. Всё это приводит к тому, что привычные централизованные или однопроцессорные системы заменяются распределённой системой. Такая система состоит из множества компьютеров, соединённых высокоскоростной сетью. Области применения подобных систем разнообразны. Их можно использовать как для ускорения расчетов, применяя методы параллельных вычислений [1], так и для хранения данных. Но в области распределённых систем существует важная проблема консенсуса — нахождение способа разрешения конфликтов между узлами сети. Данная работа актуальна в контексте решения данного вопроса.

Для хранения информации обычно используются базы данных. Однако часто к данным осуществляется большое количество одинаковых запросов и каждый раз заново извлекать информацию оказывается слишком затратно и неэффективно. Для решений этой проблемы используются специальные программные решения, которые позволяют кэшировать некоторые результаты выборки из базы в оперативной памяти и быстро получать информацию по одним и тем же запросам без постоянного обращения к базе.

Цели данной дипломной работы:

1. Реализовать алгоритм Raft [2] для решения задачи консенсуса в сети ненадёжных вычислений;
2. Провести сравнение Raft и Paxos;
3. Разработать приложение, осуществляющее кэширование данных в виде пар «ключ-значение»;
4. Разработать распределённое «ключ-значение» хранилище с использованием алгоритма Raft.

Методологические основы для решения задачи консенсуса и построения хранилищ ключ-значение представлены в работах Д. Онгаро [2], В.В. Воеводина [1], Л. Лэмпорта [3], Э. Таненбаума [4], Т. Кормена [5].

Бакалаврская работа состоит из введения, 3 разделов, заключения, списка использованных источников и 2 приложений. Общий объем работы — 49

страниц, из них 37 страниц – основное содержание, включая 9 рисунков, цифровой носитель в качестве приложения, список использованных источников информации – 25 наименований.

1 Общие понятия задачи консенсуса и сравнение с Paxos

1.1 Распределённые системы

Распределённая система — это набор независимых компьютеров, представляющийся их пользователям единой объединённой системой.

В этом определении оговариваются два момента:

1. Все машины автономны;
2. Пользователи думают, что работают с единой системой.

Распределённые системы должны относительно легко поддаваться расширению или масштабированию. Такие системы обычно существуют постоянно, но некоторые их части могут выходить из строя. При этом на внешнем уровне (пользователи) не должно быть известно, что произошла какая-либо замена части, выход из строя или включение новых ресурсов.

1.2 Необходимость создания Raft

Алгоритмы консенсуса позволяют набору машин работать как связная группа, которая может пережить отключения некоторых своих членов. Из-за этого они играют важную роль в построении надёжных масштабируемых программных систем. Алгоритм Paxos доминирует среди других алгоритмов консенсуса: большинство реализации основаны на нём, а также именно он используется в большинстве случаев для обучения студентов консенсусу.

К сожалению, Paxos — довольно сложный для понимания алгоритм несмотря на огромное количество попыток сделать его более применимым и простым. Кроме того, его архитектура требует серьёзных изменений для использования в реальных системах.

Главной целью создания алгоритма Raft была «понятность». Было необходимо придумать алгоритм, подходящий для решения практических задач, но в то же время значительно более простой в понимании, чем Paxos. В дизайне Raft используются некоторые техники, направленные на улучшение его понимаемости: декомпозиция (в Raft разделяются стадии выбора лидера, репликации журнала и безопасность) и сокращение пространства состояний (в Raft уменьшено количество недетерминизма, а также вариации, каким образом серверы могут быть рассогласованы).

Raft похож на многие существующие алгоритмы консенсуса, но имеет ряд новшеств:

1. Сильный лидер: в Raft используется более сильная форма лидерства, чем в других алгоритмах. Например, записи в журнале идут только от лидера к другим серверам. Это упрощает понимание и управление распределённым журналом;
2. Выборы лидера: в Raft используются случайные задержки для выбора лидера;
3. Изменение состава: механизм Raft для изменения состава серверов в кластере использует новый подход – стыковочный консенсус, в котором большинство двух различных конфигураций пересекаются во время перехода. Это позволяет кластеру нормально функционировать во время применения изменений.

Работа алгоритма консенсуса заключается в поддержании распределённого журнала согласованным. Модуль на сервере, отвечающий за консенсус, получает команды от клиента и добавляется их в журнал. Он общается с такими же модулями на других серверах, чтобы обеспечить, что в конечном итоге все журналы содержат все записи в нужном порядке, даже если какие-либо серверы вышли из строя.

Алгоритмы консенсуса на практике обычно имеют следующие свойства:

1. Они обеспечивают надёжность (никогда не возвращает неверный результат) при любых невизантийский условиях, в том числе, задержки сети, потеря пакетов, дупликация, переупорядочивание;
2. Они полностью доступны до тех пор, пока большая часть серверов в рабочем состоянии и могут взаимодействовать с другими серверами и клиентами. Следовательно, кластер с пятью серверами может продолжить работу при отключении любых двух серверов. Считается, что сервер выходит из строя при остановке, позже они могут восстановиться в нормальное рабочее состояние и вернуться в кластер;
3. Они не зависят от времени, чтобы обеспечить согласованность журналов. Неисправные часы и экстремальные задержки могут вызвать проблемы с доступностью;
4. В общем случае, команда завершена тогда, когда большая часть кластера ответила на запрос. Медленные серверы не должны влиять на общую производительность системы.

2 Алгоритм Raft

Raft — это алгоритм для управления распределённым журналом транзакций. Raft уже используется в реальных системах, например, Facebook использует его в HydraBase, HashiCorp — в продукте Consul, а также он используется в InfluxDB.

Все серверы:

- Если $commit\_index > last\_applied$, то увеличить last_applied, применить команду log_arr[last_applied];
- Если RPC запрос или ответ содержит этап $T > current\_term$, то установить $current\_term = T$, перейти в состояние подписчика;

Подписчики:

- Отвечать на RPC от кандидатов и лидера;
- Если истекло время ожидания выборов без получения AppendEntries от лидера или без отдачи голоса за кандидата, то перейти в состояние кандидата;

Кандидаты:

- При переходе в состояние кандидата начать выборы:
  - Увеличить current_term;
  - Проголосовать за себя;
  - Обновить таймер выборов;
  - Отправить RequestVote всем остальным серверам;
- Если получено большинство голосов, то перейти в состояние лидера;
- Если получено AppendEntries от нового лидера, то перейти в состояние подписчика;
- Если истекло время выборов, то начать новые выборы.

Лидер:

- После избрания отправить пустое AppendEntries («сердцебиение») всем серверам; повторять во время простоя для предотвращения новых выборов;
- Если получена команда от клиента, то добавить её к локальному журналу, ответить клиенту, когда команда будет зафиксирована;
- Если $last\_log\_index >= next\_index$ для некоторого подписчика, то отправить AppendEntries с записями, начинающимися с next_index:
  - Если выполнено успешно, то обновить next_index и match_index для

данного подписчика;

- Если завершилось неудачей, то уменьшить next_index и повторить;

— Если существует N, такое что $N > commit\_index$, большинство $match\_index[i] >= N$ и $log\_arr[N].term == current\_term$, то установить $commit\_index = N$;

Консенсус в Raft достигается в первую очередь за счет выбора лидера, который затем отвечает за управление распределенным журналом записей. Лидер принимает транзакции от клиентов, дублирует их на другие серверы и сообщает им, когда можно по полученным данным осуществить переход в новое состояние конечного автомата. Управление журналом сильно упрощается за счет выделения лидера. Например, лидер может определить, где будут находиться новые записи в журнале без общения с другими серверами. Информация передаётся несложным образом — от лидера к другим серверам. Однако лидер может выйти из строя. Это инициирует выборы нового лидера.

Используя такой подход, Raft разделяет задачу консенсуса на три независимые подзадачи:

1. Выбор лидера: если нынешний лидер вышел из строя, то должен быть избран новый лидер;
2. Дупликация журнала: лидер должен принимать транзакции клиентов и распространять их по всему кластеру, обеспечивая соответствие всех журналов его собственному;
3. Безопасность: главное свойство, обеспечивающее безопасность в Raft — это свойство безопасности конечного автомата. Если какой-либо сервер применил данное правило, то никакой другой сервер не может применить другое правило для той же позиции в журнале активности.

После описания алгоритма консенсуса в данной главе обсуждаются вопросы доступности и правильной временной синхронизации системы.

2.1 Основы Raft

В кластере Raft находится несколько серверов, обычно это число равно 5, что позволяет продолжить работу минимум при 3 активных серверах. В любой момент времени каждый сервер находится в одном из трех состояний: лидер, подписчик, кандидат. Состояние сервера хранится в специальной переменной *State*. При нормальном функционировании системы всегда присутствует один лидер, а все остальные являются подписчиками. Подписчики пассивны: они не

инициируют запросы, а просто отвечают на запросы от лидера или кандидатов. Лидер отвечает за все запросы клиента (если запрос от клиента приходит к подписчику, то он перенаправляется лидеру).

Raft делит время на этапы произвольной длины. Номер текущего этапа для данного сервера хранится в переменной *current_term*. Этапы нумеруются последовательно целыми числами. Каждый этап начинается с избрания, в котором один или несколько кандидатов пытаются стать лидером. Если кандидат побеждает, то он становится лидером до конца этого этапа. Иногда выборы могут кончиться с разделением голосов. В таком случае данный этап закончится без лидера, и начнется следующий (с новым избранием). Raft отвечает за то, чтобы на каждом этапе был ровно один лидер.

Для всех серверов очередной этап может начаться в разное время, а некоторые серверы могут пропустить стадию избрания или даже весь этап. Этапы выступают в качестве логических часов и позволяют обнаруживать устаревшую информацию, например, лидеров с устаревшими данными. Каждый сервер хранит номер текущего этапа, который монотонно увеличивается. Серверы обмениваются номерами своих текущих этапов. Если сервер получается номер этапа больший, чем его собственный, то он увеличивает счетчик до наибольшего значения. Если кандидат или лидер обнаруживают, что их счетчик отстаёт, то он немедленно переходит в состояние подписчика. Запросы с устаревшими номерами отклоняются.

В оригинальном алгоритме сервера общаются посредством удалённых вызовов процедур (RPC). В базовой версии Raft использует только два типа RPC. RequestVote RPC инициируются кандидатами на стадии избрания, AppendEntries RPC используются лидерами для репликация данных. Серверы повторяют RPC, если не получают ответ в течение некоторого времени, кроме того, в целях быстродействия RPC запускаются параллельно.

3 Распределённое хранилище типа «ключ-значение»

Перед нами стояла задача разработать распределённое хранилище типа «ключ-значение». Для достижения данной цели необходимо разработать непосредственно структуры данных для хранения такого типа информации и построить надёжную сеть распределённых серверов, которые бы информацию хранили. Для непротиворечивости данных, хранимых на серверах, необходимо разработать алгоритм для решения задачи консенсуса. Рассмотрим подробнее структуры данных, которые использовались для построения хранилища.

В качестве такой структуры был выбран массив красно-черных деревьев.

3.1 Собственная реализация RPC

В оригинальном алгоритме все межсерверные операции осуществляются посредством удалённых вызовов процедур (RPC — Remote Procedure Call). Данная технология призвана упростить работу программистов по осуществлению взаимодействия распределённых систем, чтобы вызов функции на удалённом сервере был не сложнее, чем вызов локальной функции. Идея состоит в том, чтобы абстрагировать детали выполнения операций от разработчика. В исполняемой программе происходит вызов процедуры, она выполняется на удалённой машине и возвращает результат. Все детали межсетевого взаимодействия скрыты от программиста.

Такой подход имеет и другие проблемы. В данной работе было решено отказаться от использования RPC, чтобы получить понимание работы алгоритма и полный контроль над исполняемыми операциями. А RPC-системы требуют наличия специальных сторонних приложений, а также сгенерированного кода.

Однако в работе осталось присутствие термина RPC и всех остальных операций, требующих таких вызовов (AppendEntries RPC, RequestVote RPC), чтобы сохранить понятность алгоритма. Соответственно, термин RPC будет использован далее для обозначения межсетевого взаимодействия и некоторых команд.

ЗАКЛЮЧЕНИЕ

В современных организациях большое внимание уделяется построению надёжной сетевой инфраструктуры. Увеличиваются объёмы данных, которые нужно обрабатывать в режиме реального времени. Привычные централизованные или однопроцессорные системы заменяются распределённой системой. Области применения подобных систем разнообразны. Но существует важная проблема консенсуса — нахождение способа разрешения конфликтов между узлами сети.

В настоящей работе было разработано распределённое клиент-серверное приложение для хранения пар ключ-значение. Для хранения таких пар в оперативной памяти устройства разработана структура данных красно-черное дерево. Для поддержания актуальности информации в распределённой системе и решения задачи консенсуса был реализован и протестирован алгоритм Raft.

Таким образом, в результате данной работы были решены следующие задачи:

1. Изучен алгоритм Raft для решения задачи консенсуса в сети ненадёжных вычислений;
2. Разработано распределённое «ключ-значение» хранилище с использованием алгоритма Raft;
3. Разработано приложение, осуществляющее кэширование данных в виде пар «ключ-значение» на основе красно-черного дерева.

Полученное приложение не является портируемым или кроссплатформенным, поскольку для сетевого взаимодействия использовались различные платформозависимые флаги. Приложение было протестировано на операционной системе Ubuntu версии 18.04, на процессоре Intel Core i5 5200U 2200 МГц с объёмом кэш-памяти 3 Мбайта, 2 ядрами и 4 логическими процессорами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Воеводин, В. В.* Параллельные вычисления / В. В. Воеводин, В. В. Воеводин. — СПб: БХВ-Петербург, 2002. — С. 608.

2 *Ongaro, D.* In search of an understandable consensus algorithm (extended version) / D. Ongaro, J. Ousterhout. — 2013.

3 *Lamport, L.* The byzantine generals problem / L. Lamport, R. Shostak, M. Pease // *ACM Transactions on Programming Languages and Systems*. — 1982. — Vol. 4, no. 3.

4 *Таненбаум, Э.* Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. — Питер, 2003. — С. 877.

5 *Кормен, Т. Х.* Алгоритмы. Построение и анализ. Третье издание / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. — Москва: Вильямс, 2015. — С. 1323.