Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по научно-исследовательской работе студента

на тему Анализ и формальная верификация алгоритма консенсуса Raft

ФИО студента: Железцов Никита Владимирович

Группа: ИУ8-94

Специальность: 10.05.01 «Компьютерная безопасность»

Специализация: 10.05.01_01 «Математические методы защиты информации»

Научный руководитель НИРС:

Работа выполнена:	_____	_____	Железцов Н.В.
	Дата	Подпись	(И. О. Фамилия)
Допуск к защите:	_____	_____	Колесников А.В.
	Дата	Подпись	(И. О. Фамилия)

Дата защиты НИРС: _____

Результаты защиты: _____

Москва, 2025 г.

РЕФЕРАТ

Отчёт содержит 23 стр., 1 рис., 2 прил.

Ключевые слова: распределенные системы, задача двух генералов, невозможность Фишера-Линча-Патерсона, консенсус, Raft, Paxos, язык спецификаций TLA+.

Основная цель работы — провести анализ алгоритма консенсуса Raft, описать принципы его работы, сопоставить его с другими невизантийскими алгоритмами и проверить его корректность с использованием языка спецификаций TLA+.

В процессе работы было проведено исследование отечественной и зарубежной литературы по заданной теме, изучены подходы к достижению консенсуса в распределенных системах. Была написана TLA+ спецификация одной из имплементаций алгоритма Raft, в частности Raft с реконфигурацией реплик.

В результате данного исследования было установлено, что Raft проще для понимания и реализации, но принципиального отличия между Raft и Paxos нет, а потому выбор за разработчиками. Было установлено, что дизайн Raft соответствует предъявляемым требованиям.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Мир вычислительных технологий за свою жизнь пережил значительные изменения: от монолитных приложений прошлых лет до современных микросервисов - подходы к обработке данных претерпели глубокую трансформацию. Централизованные приложения, некогда считавшиеся вершиной технологий, уже не отвечали запросам времени, и цифровая реальность потребовала чего-то более гибкого, масштабируемого и устойчивого.

Так наступила эпоха распределенных систем. Они разделяют задачи на части, распределяя их между множеством узлов, работающих в гармонии друг с другом. Любое высоконагруженное приложение или база данных использует распределенные системы для обработки миллионов одновременных запросов.

Однако под всей этой кажущейся простотой скрывается фундаментальная проблема: необходимость согласования действий и состояний множества узлов, разбросанных по разным частям мира и часто подверженных сбоям. Здесь на помощь и приходят алгоритмы консенсуса, которые в распределенных системах выступают хранителями целостности данных, гарантами отказоустойчивости.

Обоснование актуальности темы исследования

В современных распределённых системах алгоритмы консенсуса (такие как Raft, Paxos) являются основой для обеспечения согласованности данных и состояний между узлами. Учитывая растущую зависимость от распределённых систем в различных отраслях, корректность этих алгоритмов критически важна для обеспечения надёжности и доступности сервисов.

Формальная верификация предоставляет математически строгие методы проверки алгоритмов. Использование TLA+ для анализа и проверки алгоритма Raft позволяет доказать корректность его ключевых свойств. Это снижает вероятность ошибок и позволяет проверить корректность дизайна алгоритма. Только после уверенности в алгоритме можно приступать к его имплементации в коде.

Верификация алгоритмов распределённых систем становится всё более востребованной в сфере информационных технологий. Компании, такие как Amazon, Google, и Microsoft, активно используют формальные методы для проверки своих систем. Исследование формальной верификации Raft с применением TLA+ предоставляет актуальные знания и навыки, востребованные на рынке труда.

Таким образом, анализ и формальная верификация алгоритма Raft с использованием TLA+ представляют собой актуальную тему, находящуюся на пересечении интересов академического сообщества и индустрии IT. Результаты исследования будут полезны как исследователям, изучающим механизмы согласования в распределённых системах, так и практикующим разработчикам, создающим надёжные и отказоустойчивые системы.

Цели и задачи НИРС

Целью исследования является изучение алгоритма Raft, его формальный анализ и верификация. Для достижения этой цели предполагается решение следующих задач:

- изучение и описание принципов работы Raft;
- сравнительный анализ Raft и других алгоритмов консенсуса;
- формальная спецификация алгоритма Raft на TLA+ и проверка его корректности.

В рамках данной работы рассматриваются только невизантийские алгоритмы консенсуса.

ОСНОВНАЯ ЧАСТЬ

1 Модель распределенной системы

Эта часть задает основу для последующего анализа алгоритмов консенсуса, определяя термины распределенной системы и консенсуса. Она задает условия и ограничения, в которых эти алгоритмы могут быть применены.

1.1 Определение распределенной системы

Формального определения распределенной вычислительной системы в настоящее время не существует. Из множества различных определений, можно выделить ироничное определение Лесли Лампорта [**lamport_email**], которое он дал в мае 1987 года, в своем письме коллегам по поводу очередного отключения электроэнергии в машинном зале:

«Распределенной вычислительной системой можно назвать такую систему, в которой отказ компьютера, о существовании которого вы даже не подозревали, может сделать ваш собственный компьютер непригодным к использованию».

Эндрю Таненбаум предложил следующее, более серьзное, определение [**tanenbaum_distributed_systems**]:

«Распределенная вычислительная система (РВС) – это набор соединенных каналами связи независимых компьютеров, которые с точки зрения пользователя некоторого программного обеспечения выглядят единым целым».

Именно это определение и будет использоваться в данной работе. Таким образом, в РВС есть несколько автономных участников (иногда называемых процессами, узлами или репликами). Каждый участник обладает своим локальным состоянием. Участники выполняют некий алгоритм, общаются, обмениваясь сообщениями через каналы связи между ними. Вне системы существуют пользователи, которые могут отправлять запросы к различным узлам системы и ожидать от них ответов.

1.2 Модель каналов связи

Связь через каналы часто ненадежна: сообщения могут теряться, задерживаться или приходить в неправильном порядке. В качестве отправной точки взят канал с приемлемыми потерями (fair-loss) [**cachin11**]:

– Допустимые потери. Если отправитель и получатель функционируют корректно и процесс отправляет сообщение бесконечно много раз, оно в конечном итоге будет доставлено бесконечное число раз.
– Ограниченное дублирование. Одно отправленное сообщение не будет доставлено бесконечное количество раз.
– Без создания. Канал не создает новых сообщений, то есть доставляются только те сообщения, которые были отправлены.

Канал с приемлемыми потерями является полезной абстракцией и первым строительным блоком для протоколов обмена данными с сильными гарантиями. По своей сути он похож на протокол UDP, который позволяет отправлять сообщения от одного процесса другому, но не предлагает надежной семантики доставки на уровне протокола. Однако гарантии, предоставляемые такой моделью недостаточны для построения надежных распределенных систем.

Для повышения надежности связи можно использовать подтверждения (acknowledgement, ACK), позволяющие получателю уведомить отправителя о доставке сообщения. Для этого применяются полнодуплексные каналы связи и добавляются механизмы, которые помогают различать сообщения, например, уникальные порядковые номера (sequence numbers), монотонно возрастающие идентификаторы.

До тех пор, пока отправитель не получит подтверждение о доставке сообщения, он не может быть уверен, было ли оно обработано, будет ли обработано в будущем, утеряно, либо удаленный процесс вышел из строя до его получения. Отправитель может повторно передать сообщение, но это может вызвать его дублирование. Безопасная обработка дубликатов возможна только в случае, если выполняемая операция является идемпотентной. Поскольку в реальных условиях не всегда удается обеспечить идемпотентность операций, требуется применять эквивалентные гарантии. Для этого можно использовать методы дедупликации (deduplication), предотвращающие повторную обработку одного и того же сообщения.

Повторные передачи и несоблюдение порядка доставки могут приводить к неверному порядку поступления сообщений. Благодаря введению порядковых номеров получатель может использовать их для восстановления порядка и обеспечения обработки сообщений по принципу FIFO (first-in, first-out).

Применив все описанные выше ограничения к каналу с приемлемыми ограничениями, получаем совершенный канал (perfect link), предоставляющий гарантии [**cachin11**]:

- Надежная доставка. Каждое сообщение, отправленное один раз корректным процессом А корректному процессу Б, в конце концов будет доставлено.
- Порядок сообщений. Сообщения будут доставлены в том порядке, в котором они были отправлены.
- Без дублирования. Ни одно сообщение не будет обработано более одного раза.
- Без создания. Канал доставляет только отправленные сообщения, не создавая новые.

Именно совершенный канал будет использоваться как модель канала для построения алгоритмов коненсуса распределенных систем. Он похож на протокол TCP.

Однако даже совершенный канал не защищает алгоритм от ситуации разделения сети, под которой понимается, что два или более процесса не могут связаться друг с другом. Независимые группы процессов могут продолжать выполнение алгоритмов и выдывать противоречивые результаты, могут происходить асимметричые отказы каналов, при которых сообщения могут проходить только в одну сторону, но не обратно. Все это необхоимо учитывать при разработке алгоритмов в распределенных системах.

1.3 Определение консенсуса

Алгоритм консенсуса описывает работу распределенной системы, которая при наличии нескольких процессов, начинающих работу с некого начального состояния, переводит все процессы в одинаковое состояние. Чтобы алгоритм консенсуса был корректным, должны выполняться следующие условия [**petrov**]:

- Согласованность. Принимаемое протоколом решение должно быть единодушным: каждый процесс выбирает некоторое значение, которое должно быть одинаковым для всех процессов.
- Действительность. Согласованное значение должно быть предложено одним из процессов. Т.е. это не может быть некое произвольное значение.

– Окончательность. Согласованность принимает окончательный характер после того, как уже не остается процессов, не достигших состояния принятия решения.

1.4 Теорема Фишера-Линча-Патерсона

Мысленный эксперимент, известный как «задача двух генералов», наглядно иллюстрирует сложности обеспечения консенсуса в распределенных системах. Эксперимент демонстрирует, что в асинхронной системе, где нет временных ограничений на доставку и ответы, невозможно достичь полного согласия между сторонами, даже с использованием многочисленных подтверждений.

Суть задачи такова: две армии под командованием генералов расположены по разные стороны от города и должны атаковать одновременно, чтобы достичь успеха. Генералы передают сообщения через посланников, чтобы договориться об атаке. Однако посланники могут быть перехвачены или не доставить сообщение, что создает неопределенность. То есть генералам необходимо достичь консенсуса по времени начала атаки.

Например, генерал А отправляет сообщение MSG(N), предлагая атаку в определенное время. Генерал Б, получив сообщение, посылает подтверждение ACK(MSG(N)). На рис. 1 показано, что сообщение отправляется в одну сторону и подтверждается другой стороной.

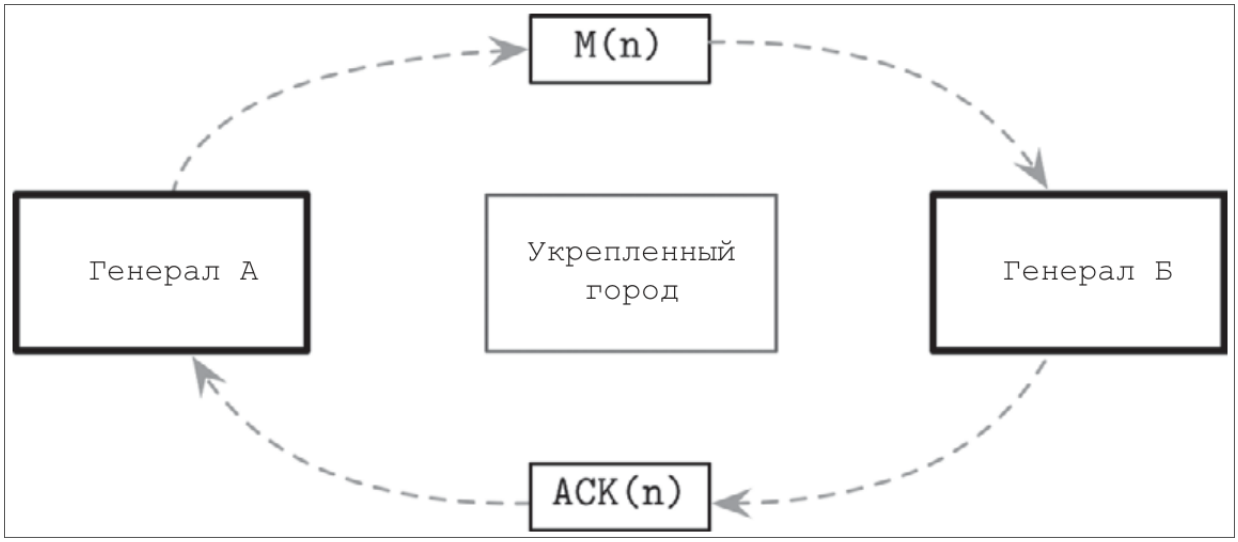

Рисунок 1 – Задача двух генералов

Однако генерал A не может быть уверен, что это подтверждение дошло, оно может быть утеряно. Чтобы устранить сомнения, требуется второе подтверждение — ACK(ACK(MSG(N))). Этот процесс может продолжаться бесконечно, поскольку всегда остается риск того, что последнее сообщение не достигло адресата.

В задаче не сделано предположений о времени, не установлено лимита, за которое генералы должны ответить, связь полностью асинхронна.

В работе Фишера, Линча и Патерсона рассматривается проблема, известная как невозможность ФЛП (FLP Impossibility) [**fischer85**]. Эта теорема, также именуемая теоремой Фишера–Линча–Патерсона, изучает консенсус в системах, где процессы начинают с начального значения и стремятся согласовать новое общее значение. После выполнения алгоритма это значение должно быть одинаковым для всех корректно работающих процессов.

В исследовании предполагается полностью асинхронная среда, где процессы не имеют общего времени. Алгоритмы в таких системах не могут полагаться на время ожидания, а у процесса отсутствует возможность определить, отказал ли другой процесс или просто работает медленно. В статье доказывается, что при указанных предположениях невозможно создать протокол, который гарантирует достижение консенсуса за ограниченное время. Даже один сбой процесса, не сопровождающийся уведомлением, делает консенсус недостижимым для полностью асинхронной распределенной системы.

Тем не менее, теорема ФЛП не утверждает, что достижение консенсуса в принципе невозможно. Она лишь показывает, что в асинхронной системе консенсус не всегда достижим за конечное время. В реальных системах часто присутствует некоторая степень синхронности, и эту особенность нужно учитывать.

1.5 Синхронность

Из теоремы ФЛП следует, что ключевой характеристикой распределенной системы является предположение о времени. В асинхронных системах невозможно гарантировать ограниченные задержки в доставке сообщений, упорядоченность их получения. Процесс выдает ответ через неопределенной долгое время.

Однако, одним из аргументов против асинхронных систем является их оторванность от реальности: процессы не имеют произвольно разные скорости обработки, а задержки передачи сообщений не бывают бесконечными.

Можно сделать предположения менее строгими, считая систему синхронной. Может предполагаться, что процессы работают с сопоставимыми скоростями, задержки передачи сообщений в канале ограничены, их доставка не может занимать бесконечно долгое время.

В модель синхронной системы также можно добавить локальные для процесса синхронизированные часы: при этом существует некоторая верхняя граница в раз нице во времени между двумя локальными для процессов источниками времени [**cachin11**].

Свойства асинхронных и синхронных моделей можно объединить, рассматривая систему как частично синхронную. Частично синхронная система обладает некото рыми свойствами синхронной системы, но при этом ограничения на время доставки сообщений, уход показаний часов и относительные скорости обработки могут быть приблизительными и действовать лишь в большинстве случаев.

1.6 Модели отказов

Модель отказов описывает, каким образом могут происходить сбои в работе процессов распределенной системы. Например, можно предположить, что процесс может полностью завершить работу и не восстановиться, восстановиться через некоторое время или начать выдавать некорректные данные из-за сбоя или преднамеренно.

Поскольку процессы в распределенных системах взаимодействуют друг с другом при выполнении алгоритма, сбои в одном из них могут нарушить выполнение всей системы. Для разработки алгоритмов в распределенных системах необходимо понимать, какие отказы могут случиться, чтобы правильно обрабатывать каждый из них.

Аварийное завершение

В этой модели предполагается, что процесс перестает выполнять шаги алгоритма и не отправляет сообщений другим процессам. После сбоя процесс остается в этом состоянии и больше не участвует в текущем выполнении алгоритма.

Важно отметить, что эта модель не запрещает восстановление процессов. Процесс может восстановиться, синхронизироваться с текущим состоянием системы и участвовать в новых раундах алгоритма.

Процесс, который восстанавливается после сбоя, может продолжить выполнение с последнего известного ему шага. Алгоритмы, поддерживающие восстановление, должны вводить в систему понятия устойчивого состояния алгоритма и протокола восстановления [**skeen83**].

Пропуск

В этой модели процесс пропускает выполнение отдельных шагов алгоритма, либо эти шаги невидимы для других участников, либо процесс не может отправлять и получать сообщения. Пропуски включают сетевые сбои, вызванные неисправностями каналов связи, отказами оборудования или перегрузкой сети.

Пропуски возникают, когда алгоритм не завершает определенные действия, или их результаты не достигают других процессов. Например, потерянное сообщение может привести к тому, что отправитель считает его доставленным, несмотря на его окончательную утрату.

Византийские ошибки

Произвольные, или византийские, ошибки (Byzantine faults) представляют собой наиболее сложный класс отказов. В этом случае процесс продолжает выполнять шаги алгоритма, но делает это некорректно.

Такие сбои могут быть вызваны ошибками в программном обеспечении, выполнением разных версий алгоритма или намеренными действиями злоумышленников. Например, в распределенных системах без централизованного управления, таких как криптовалюты, процессы могут фальсифицировать данные, чтобы ввести систему в заблуждение.

Обработка отказов

Одним из способом обработки отказов при невизантийских ошибках является введение в алгоритм избыточности. Таким образом отказ может быть замаскирован: даже если один или несколько процессов откажут, то пользователь этого не заметит [**christian91**]. Raft и Paxos, которые будут рассмотрены в данной работе, основываются на модели отказов и минимизирует влияние отказов путем использования избыточности процессов.

В случае работы в распределенной системе с византийскими ошибками используется перекрестная проверка других узлов на каждом шаге, поскольку узлы не могут полагаться друг на друга или на лидера и должны проверять поведение других узлов, сравнивая возвращаемые результаты с ответами большинства. PBFT, PoW, PoS борются с ошибками именно так.

1.7 Итоги

Таким образом, к распределенной системе, в которой будет исполняться любой из описанных в данной работе алгоритм консенсуса, применяются следующие ограничения:

- Система передает сообщения по совершенным каналам, однако допустимо разделение сети.
- Необходимо учитывать возможность отказа узлов: аварийное завершение и пропуск части алгоритма процессом.
- В системе невозможны византийские ошибки. Предполагается, что все узлы честные и выполняют свои функции корректно.
- Система является частично синхронной, в ней существует понятие времени, которое используется для синхронизации локальных часов. Сообщения в системе могут отправляться бесконечно, процессы могут работать с произвольной скоростью.

2 Paxos

Алгоритм Паксос (Paxos) считается одним из самых известных методов достижения консенсуса. Его предложил Лесли Лэмпорт в своей работе The Part-Time Parliament («Парламент с неполной занятостью») [**lamport98**]. В данной статье идея консенсуса была представлена через метафору законодательного процесса и голосования, происходящих на острове Паксос. Позднее, в 2001 году, Лэмпорт опубликовал статью Paxos Made Simple («Простой Паксос») [**lamport01**], где использовал упрощенную терминологию, которая и используется в данной работе.

В рамках алгоритма Паксос участники могут выполнять одну из трех ролей:

- Заявители (proposers). Принимают значения от клиентов, формируют предложения для их утверждения и пытаются получить голоса акцепторов.
- Акцепторы (acceptors). Участвуют в голосовании за принятие или отклонение предложений, сформированных заявителями. Для устойчивости к отказам система включает несколько акцепторов, однако для принятия решения достаточно кворума (то есть большинства голосов).
- Ученики (learners). Отвечают за хранение результатов принятых решений, выполняя роль реплик.

  В большинстве реализаций эти роли совмещены в одном процессе.

Каждое предложение содержит уникальный номер, который монотонно увеличивается, и значение, предложенное клиентом. Номер предложения используется для установления общего порядка операций и определения их последовательности («до» или «после»). Номера предложений часто оформляются как пара (идентификатор узла и временная метка). Идентификаторы узлов являются упорядочиваемыми и могут быть использованы для разрешения конфликтов между временными метками. Для корректной работы алгоритма требуется синхронизация локальных часов.

2.1 Алгоритм Paxos

Алгоритм Паксос можно разделить на два ключевых этапа: голосование (или этап предложения) и репликацию. На этапе голосования заявители борются за установление своего лидерства, а на этапе репликации заявитель распространяет согласованное значение среди акцепторов.

Заявитель выступает как начальная точка контакта для клиента, получая значение, которое должно быть принято, и пытается собрать голоса от акцепторов. После этого акцепторы рассылают информацию о принятом значении среди учеников, что позволяет ответить пользователю. Ученики, в свою очередь, увеличивают коэффициент репликации согласованного значения.

Только один заявитель может получить большинство голосов. В случае, если голоса распределяются равномерно, ни один из заявителей не сможет набрать большинство, и тогда им придется начать процесс заново.

На этапе предложения заявитель отправляет запрос (n) (где n — номер предложения) большинству акцепторов, пытаясь собрать их голоса. Акцептор, получив запрос, должен ответить, соблюдая несколько инвариантов [**lamport01**]:

- Если акцептор еще не отвечал на запрос с более высоким порядковым номером, он обещает не принимать предложения с меньшими номерами.
- Если акцептор уже принял какое-то предложение, он уведомляет заявителя о принятом значении через сообщение (m, v). Если акцептор ранее ответил на запрос с большим порядковым номером, он уведомляет заявителя о существующем предложении с более высоким номером. Акцептор может ответить на несколько запросов, если последний имеет наибольший порядковый номер.

Когда заявитель получает большинство голосов, начинается этап репликации. Заявитель фиксирует предложение, отправляя акцепторам сообщение Принятие(n, v), где v — это значение, соответствующее предложению с наибольшим номером среди полученных ответов от акцепторов, или собственное значение заявителя, если акцепторы не приняли других предложений.

Акцептор примет предложение с номером n, если на этапе предложения он не ответил на запрос с большим номером. Если акцептор отклоняет предложение, он отправляет заявителю максимальный порядковый номер, который он видел, для актуализации данных заявителя [LAMPORT01].

После того как консенсус относительно значения достигнут (хотя бы один акцептор принял решение), последующие заявители должны принять это значение, чтобы сохранить согласованность. Поэтому акцепторы возвращают последнее принятое значение. Если ни один акцептор не видел предыдущего значения, заявителю разрешается выбрать собственное.

Ученики получают информацию о принятом значении, когда большинство акцепторов сообщает им об этом. Акцепторы могут сразу уведомить учеников о принятом значении. При наличии нескольких учеников акцепторы должны уведомить каждого, однако можно выделить несколько учеников, которые будут отвечать за уведомление других.

Таким образом, основной целью первого этапа алгоритма является установление лидера и определение принятого значения, что позволяет перейти ко второму этапу — распространению значения. В базовом алгоритме оба этапа выполняются каждый раз, когда необходимо принять решение. Однако на практике можно уменьшить количество шагов, разрешив заявителю предложить несколько значений одновременно. Этот подход будет рассмотрен позже в разделе "Мульти-Паксос".

2.2 Сценарии отказа

2.3 MultiPaxos

3 Raft

3.1 Роль лидера в Raft

3.2 Добавление процессов в Raft

3.3 Сценарии отказа

4 Реализация алгоритма Raft на TLA+

Описание TLA+.

4.1 Формальная спецификация алгоритма

4.2 Проверка модели

ЗАКЛЮЧЕНИЕ

ПРИЛОЖЕНИЕ А

Спецификация Raft на TLA+

ПРИЛОЖЕНИЕ Б

Конфигурация модели для спецификация Raft