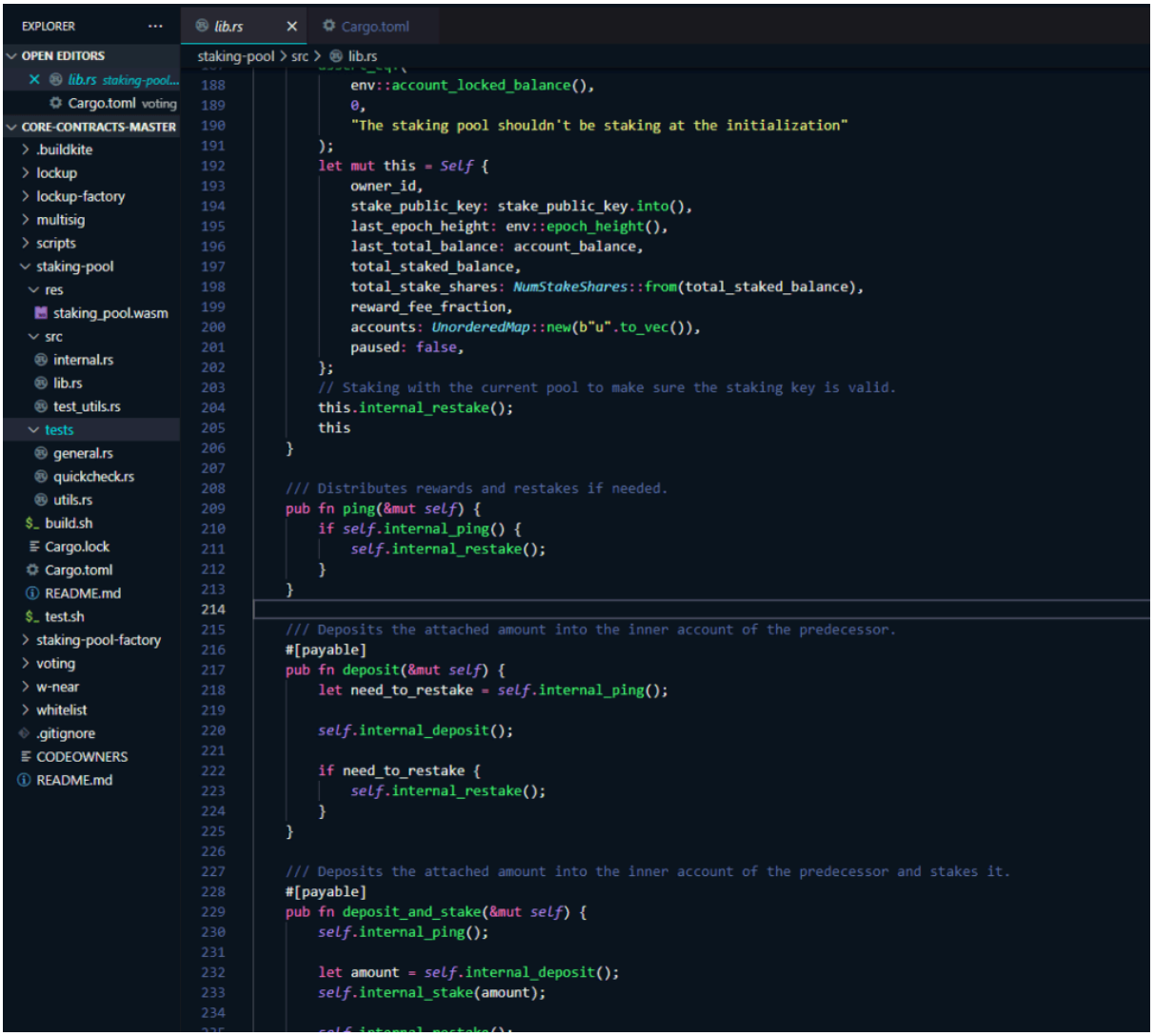
NEAR Live Обзор контрактов | Часть 2: Контракт пула ставок



Введение

Это вторая часть серии обзоров контрактов и сегодня мы собираемся рассмотреть контракт пула ставок. Он используется для защиты системы подтверждения NEAR Protocol. В основном все валидаторы, которые в настоящее время работают на протоколе NEAR, работают от имени этого контракта. Они сами не контролируют учетную запись, которая размещает количество токенов NEAR, необходимое для доказательства доли, но вместо этого контракт ставит эту сумму, и они просто предоставляют пул ставок и запускают свои ноды. Сегодня мы будем разбирать этот контракт. В основных контрактах у нас есть: контракт на пул ставок, и он немного сложнее, чем предыдущий контракт, который мы рассмотрели (контракт на голосование). Мы больше сосредоточимся на логике, а не на close_bindgen и специфичных для Rust материалах, но, вероятно, потребуется немного больше знаний о протоколе NEAR. Вот контракт пула ставок на NEAR Github. Ниже представлено оригинальное видео, на котором основано это руководство.

NEAR Live Contract Review | Episode 2: Staking Pool Contracts

lib.rs

Структура

https://gist.github.com/6codingshark/32f94874d58e8b4ee8c76e891975e840#file-staking_lib-rs

Как и прежде, контракт начинается с основной структуры. В данном случае это контракт структуры ставок. Как видите, есть Near_bindgen, BorshSerialize и BorshDeserialize. В структуре теперь намного больше полей, чем в прошлой, и к ним есть некоторые комментарии. Большинство из них, вероятно, уже обновлены. Логика контракта пула ставок позволяет нам сделать следующее: практически любой может внести некоторое количество токенов NEAR в пул ставок разместив их в пуле. Это позволяет нам объединять балансы нескольких людей (здесь мы называем их учетными записями) в одну крупную долю. Таким образом это большое количество может претендовать на места валидатора. NEAR Protocol сейчас имеет ограниченное количество мест для одного шарда, максимум 100 мест для валидаторов. Вы можете рассмотреть места следующим образом: если вы возьмете общее количество поставленных токенов и разделите его на 100, результатом будет минимальное количество токенов, необходимое для одного места, за исключением того, что удаление немного сложнее. Ставки, которые не соответствуют этой минимальной сумме, и т. д. Этот контракт в основном является автономным контрактом без какого-либо ключа доступа, который контролируется владельцем. В этом случае владелец указывается в методе инициализации.

Метод инициализации

https://gist.github.com/6codingshark/2e5b6987481ffab9e67b20f6a430f6a7#file-staking_lib2-rs

Итак, давайте перейдем к методу инициализации. Он имеет три аргумента, первый — это owner_id, который является идентификатором учетной записи владельца. У владельца есть множество разрешений на этот контракт, которые позволяют контракту выполнять действия, недоступные для остальных учетных записей. Одним из таких методов было голосование от имени пула ставок по контракту на голосование, который мы обсуждали в прошлый раз. Таким образом, владелец может вызвать метод голосования.

https://gist.github.com/6codingshark/14687860ae150ff392709803450b4170#file-staking_lib3-rs

https://gist.github.com/6codingshark/8f43d303ef2cc0e9573d2ff28ca06d48#file-staking_lib4-rs

Затем мы проверяем, что предшественник равен владельцу, поскольку этот метод может быть вызван только владельцем.

https://gist.github.com/6codingshark/8bb8b31e991a7d6f5468cf80f56cd096#file-staking_internal-rs

Итак, что делает метод голосования, он проверяет, что метод был вызван только владельцем, а затем проверяет некоторую логику, но сейчас логика нас не интересует.

https://gist.github.com/6codingshark/2e5b6987481ffab9e67b20f6a430f6a7#file-staking_lib2-rs

Итак, контракт является владельцем, и этот владелец может делать определенные вещи, у него есть дополнительные разрешения. Затем он занимает еще несколько полей: он принимает stake_public_key. Когда вы делаете ставку на протокол NEAR, вам необходимо предоставить открытый ключ, который будет использоваться вашим узлом валидатора для подписи сообщений от имени узла валидатора. Этот открытый ключ может отличаться от любого ключа доступа, и в идеале он должен отличаться от любого ключа доступа, потому что ваш узел может работать в центре обработки данных, который может быть уязвим для некоторых атак. В этом случае максимум, что могут сделать злоумышленники, это навредить сети, но не вашей учетной записи. Они не могут украсть ваши средства, и вы можете легко заменить этот ключ на другой ключ доступа. Наконец, третий аргумент, который принимает контракт, — это reward_fee_fraction. Это комиссия, которую берет владелец стейкинг-пула за запуск узла валидатора.

https://gist.github.com/6codingshark/e5f23a4d6b9b7d57e7d91293e0155592#file-staking_lib5-rs

Это дробь, у которой есть числитель и знаменатель, и она позволяет вам сказать: «Я получаю 1% вознаграждения за управление этим конкретным пулом». Допустим, у вас есть 1 000 000 токенов, они получили какое-то вознаграждение, есть вознаграждение в 10 000 токенов, тогда владелец возьмет 1% из этого, что составляет 100 токенов. Плавающие запятые ведут себя непредсказуемо, когда вы их умножаете. Например, с дробями вы можете использовать математику с большим количеством битов. Например, вы выполняете деление, сначала умножаете сумму, которая равна u128, на числитель (это уже может переполниться в u128), поэтому мы делаем это в u256. Затем вы делите его на знаменатель, который снова должен стать ниже u128. Это дает вам более высокую точность, чем float64, который не может работать с точностью до 128 бит, поэтому у него будут ошибки округления или ошибки точности при выполнении операций. Это означает, что вам нужны более точные числа с плавающей запятой, которые на самом деле не отличаются от математики, где мы моделируем это с помощью u256. Изначально Solidity не поддерживала числа с плавающей запятой, и мы тоже изначально не поддерживали, но это создавало некоторые проблемы с форматированием строк в Rust для отладки, поэтому мы решили, что нет ничего плохого в поддержке чисел с плавающей запятой, тем более что мы сделали это на стороне виртуальной машины. Самой большой проблемой с плавающей запятой было неопределенное поведение при определенных значениях нагрузки. Например, какие другие биты содержат ее, когда у вас есть бесконечное число с плавающей запятой. Мы стандартизировали это, и теперь они эквивалентны независимым платформам. Итак, теперь можно использовать float в нашей виртуальной среде.

https://gist.github.com/6codingshark/2e5b6987481ffab9e67b20f6a430f6a7#file-staking_lib2-rs

Стандартная практика с init заключается в том, что мы сначала проверяем, что состояние не существует. Затем мы проверяем ввод. Первое, что мы делаем, это проверяем правильность дроби и проверяем, что знаменатель не равен нулю. Далее у нас есть оператор else, который проверяет, что числитель меньше или равен знаменателю, что означает, что дробь меньше или равна 1. Это важно, чтобы избежать некоторых логических ошибок. Следующее, что мы делаем, это проверяем, что учетная запись действительна. Этот контракт был написан до появления некоторых вспомогательных метрик, которые существуют сейчас. Например, у нас есть действительный идентификатор учетной записи в типах JSON, который делает эту проверку автоматически во время десериализации, если он недействителен, он просто паникует. После этого подтягиваем текущий баланс счета стейкинг-контракта. Этот баланс обычно достаточно велик, потому что он должен платить за хранение этого конкретного контракта, и тогда мы говорим, что собираемся выделить несколько токенов для STAKE_SHARE_PRICE_GUARANTEE_FUND. Пул ставок имеет определенные гарантии, которые важны для локальных контрактов. Это гарантирует, что когда вы вносите депозит в пул ставок, вы должны иметь возможность вывести по крайней мере такое же количество токенов, и вы не можете потерять токены даже на 1 000 000 000 000 лет до NEAR по этому контракту, внося и выводя из пулы ставок. Фонд STAKE_SHARE_PRICE_GUARANTEE_FUND составляет около 1 триллиона NEAR в год, в то время как мы обычно потребляем около 1 или 2 триллиона NEAR в год на ошибки округления. Наконец, мы помним, каков баланс, который мы собираемся поставить от имени этого контракта. Это необходимо для установления некоторого базового уровня, чтобы ограничить разницу в округлении. Затем мы проверяем, что учетная запись еще не застейкана. Это может нарушить некоторую логику. Мы не хотим, чтобы это произошло, поэтому мы хотим инициализировать контракт до того, как он что-либо поставит. Наконец, мы инициализируем структуру, но не возвращаем ее сразу. Мы только что создали здесь структуру :StakingContract.

https://gist.github.com/6codingshark/2e5b6987481ffab9e67b20f6a430f6a7#file-staking_lib2-rs

Затем мы оформляем транзакцию рестейкинга. Это важно, потому что нам нужно убедиться, что предоставленный ключ для ставок является действительным ключом с ограниченным доступом для ристретто, например, действительным ключом 5 119. На кривой есть некоторые ключи, которые являются действительными ключами, но не специфичны для ристретто, а ключи валидатора могут быть специфичны только для ристретто. Это специфика протокола NEAR, и происходит то, что он выполняет транзакцию стейкинга с заданным ключом. Как только эта транзакция создана из контракта, мы проверяем эту транзакцию, когда она уходит. Если ключ недействителен, то он выдаст ошибку, и вся инициализация этого пула ставок завершится ошибкой. Если вы передадите недействительный стейк_публичный_ключ в качестве входных данных, ваша консолидация и развертывание контракта, а также все, что происходит в этой одной пакетной транзакции, будет отменено. Это важно, чтобы у пула не было недействительного ключа, потому что это может позволить вам заблокировать долю других людей. В качестве гарантии мы говорим, что если вы отмените ставку, ваши токены будут возвращены через 4 эпохи. Они будут иметь право на вывод, и это важно, чтобы иметь возможность вернуть их в изоляцию.

https://gist.github.com/6codingshark/82c850155526c02b6b326e97c890b5f0#file-stakinlib-rs

Я думаю, что это слишком много моментов, прежде чем я объясню общий обзор того, как работают контракты и как работают балансы. Давайте объясним концепцию того, как мы на самом деле можем распределять вознаграждения между владельцами учетных записей в постоянное время. Это важно для большинства смарт-контрактов. Они хотят действовать за постоянное время для каждого метода, а не за линейное время для количества пользователей, потому что, если количество пользователей будет расти, то количество газа, необходимое для работы линейной шкалы, также будет расти, и в конечном итоге оно иссякнет. Вот почему все смарт-контракты должны действовать в постоянное время.

Структура аккаунта

https://gist.github.com/6codingshark/3c26e0f29ba60dfba2d3552cc8454f5c#file-staking_lib6-rs

Как это работает для каждого пользователя, мы сохраняем структуру, называемую учетной записью. Каждый пользователь, делегированный в этот пул ставок, будет иметь структуру, называемую учетной записью, со следующими полями: unstaked — это баланс в yocto NEAR, который не поставлен, поэтому это просто баланс пользователя. Тогда доля_количества на самом деле является балансом, но не в ПОЧТИ, а в количестве долей доли. Stake_shares — это концепция, которая была добавлена к этому конкретному пулу ставок. Это работает следующим образом: когда вы делаете ставку, вы, по сути, покупаете новые акции по текущей цене, конвертируя свой баланс без ставки в доли доли. Цена доли ставки изначально равна 1, но со временем она растет вместе с вознаграждениями, и когда учетная запись получает вознаграждения, ее общий баланс ставки увеличивается, но количество общих долей ставки не меняется. По сути, когда учетная запись получает вознаграждение за проверку или какие-либо другие депозиты прямо на баланс, это увеличивает сумму, которую вы можете получить за каждую долю ставки. Предположим, например, что у вас изначально был 1 миллион NEAR, который был внесен на этот счет. Допустим, вы получаете 1 миллион акций (пока игнорируя yocto NEAR), если пул ставок получил 10 000 NEAR в качестве вознаграждения, у вас все еще есть 1 миллион акций, но теперь 1 миллион акций соответствует 1 010 000 NEAR. Теперь, если кто-то еще захочет сделать ставку в это время, он купит акции внутри компании в рамках контракта по цене 1,001 NEAR, потому что сейчас каждая акция стоит этого. Когда вы получаете очередное вознаграждение, вам не нужно покупать больше акций, несмотря на общий баланс, и в постоянное время каждый делится вознаграждением пропорционально количеству имеющихся у него акций. Теперь, когда вы отменяете стейкинг, вы, по сути, продаете эти акции или сжигаете их, используя концепцию взаимозаменяемых токенов в пользу незанятого баланса. Таким образом, вы продаете по текущей цене, вы уменьшаете общую сумму ставки, а также общее количество акций, а при покупке вы увеличиваете общий баланс ставки и общее количество акций, сохраняя при этом цену неизменной. Когда вы делаете ставку или снимаете ставку, вы не меняете цену, когда вы получаете вознаграждение, вы увеличиваете цену.

https://gist.github.com/6codingshark/3c26e0f29ba60dfba2d3552cc8454f5c#file-staking_lib6-rs

Цена может только расти, и это может привести к ошибкам округления, когда ваш yocto NEAR и ваш баланс не могут точно совпадать. Вот почему у нас есть этот гарантийный фонд в размере 1 триллиона NEAR, который несколько раз добавит один дополнительный Yocta NEAR. Наконец, заключительная часть, потому что протокол NEAR не отменяет ставку и не возвращает баланс немедленно, он должен ждать три эпохи, пока ваш баланс не будет разблокирован и возвращен на счет. Если вы отмените ставку, вы не сможете немедленно вывести этот баланс из пула ставок, вам нужно подождать три эпохи. Затем вы помните, на какой высоте эпохи вы вызывали последнее действие отмены ставок, и через три эпохи ваш баланс станет разблокированным, и вы сможете выйти из неразмещенных ставок. Однако есть одно предостережение: если вы вызываете отмену ставок в последнем блоке эпохи, фактическое обещание, которое делает отмену ставок, будет получено для следующей эпохи. Он прибудет в первый блок следующей эпохи, и это задержит разблокировку вашего заблокированного баланса до четырех эпох вместо трех. Это потому, что мы записали эпоху в предыдущем блоке, но фактическая транзакция произошла в следующем блоке, в следующей эпохе. Чтобы этого не произошло, мы фиксируем баланс на четыре эпохи вместо трех, чтобы учесть этот пограничный случай. Вот что представляет собой учетная запись. Идея акций не нова, потому что в Ethereum большинство поставщиков ликвидности и автоматизированных маркет-мейкеров используют аналогичную концепцию. Когда вы, например, вносите депозит в пул ликвидности, вы получаете какой-то токен из этого пула вместо фактической суммы, которая там представлена. Когда вы выходите из пула ликвидности, вы сжигаете этот токен и получаете фактически представленные токены. Идея очень похожа на то, чтобы называть их акциями, потому что они имеют соответствующую цену, и мы могли бы назвать их по-другому. Это было почти с самого начала этого контракта с пулом ставок. Мы исследовали, как мы можем сделать это должным образом, и один из способов заключался в том, чтобы ограничить количество учетных записей, которые могут вносить депозиты в определенную учетную запись пула для этого конкретного обновления. В конце концов мы остановились на постоянном времени сложности, и на самом деле это была более простая модель. Затем математика структуры доли_долей стала более или менее разумной, даже несмотря на то, что здесь тоже есть некоторые моменты.

Типы контрактов

Давайте пройдемся по этому контракту. Например, он не так хорошо структурирован, как договор блокировки, потому что блокировка еще сложнее. Типы по-прежнему связаны в одном контракте. Есть большое количество типов типов, например, вознаграждение_платы_фракция — это отдельный тип.

https://gist.github.com/6codingshark/2e163935b9832c01e9d4a6eb41883028#file-staking_lib7-rs

Учетная запись — это отдельный тип, а также существует удобочитаемая учетная запись, которая также является типом, который используется только для вызовов просмотра, поэтому он не используется для внутренней логики.

https://gist.github.com/6codingshark/879ad951e6bf813b08ce4406dc052e6b#file-staking_lib8-rs

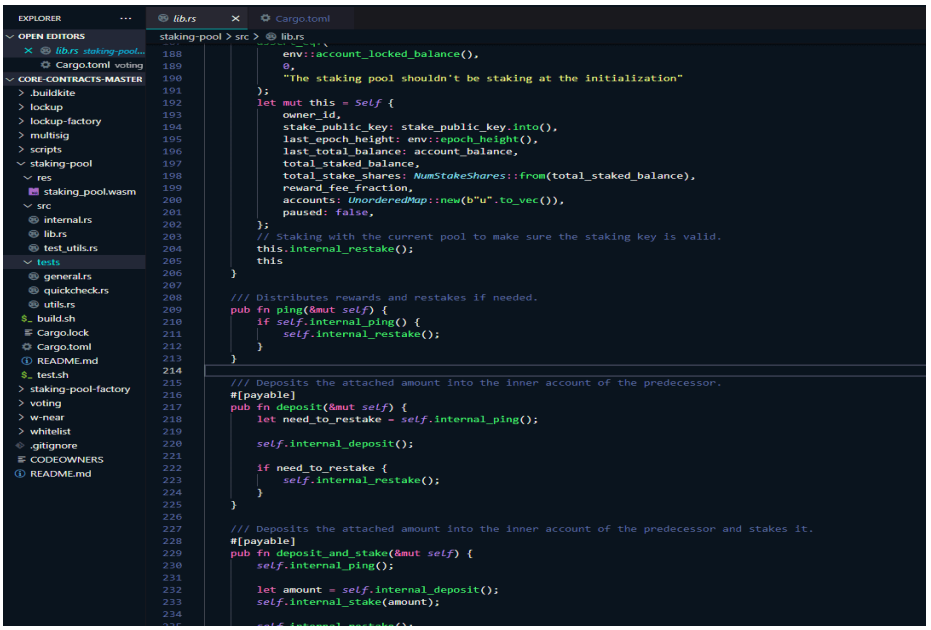
Затем, после того как мы закончим со всеми типами, у нас будут межконтрактные вызовы с использованием высокоуровневого интерфейса.

https://gist.github.com/6codingshark/1569c43a51a16504e4b5fa675b881097#file-staking_lib9-rs

Это работает следующим образом: у вас есть макрос из near_bindgen, который называется ext_contract (внешний контракт). Вы можете дать ему короткое имя, которое он сгенерирует, и которое вы сможете использовать. Затем у вас есть описание признака, описывающее интерфейс внешнего контракта, который вы хотите использовать. Это описывает тот факт, что вы можете вызвать метод голосования на удаленном контракте и передать один аргумент. Аргумент is_vote, который является истинным или ложным логическим значением. Теперь вы сможете создать промис, когда вам это нужно, и передать позиционный аргумент вместо сериализованного аргумента JSON. Макрос превратит его в низкоуровневый API обещаний за кулисами. Второй интерфейс предназначен для обратного вызова самого себя, это довольно распространено, вы можете назвать его ext_self. Когда вам нужно сделать обратный вызов и что-то сделать с результатом асинхронного обещания, вы можете использовать этот тип интерфейса. Что мы делаем, так это проверяем, удалось ли стейкинг. Наконец, у нас есть тело реализации основной структуры реализации пула ставок.

Структура файла контракта

Этот контракт разделен на несколько модулей.



У вас есть libs.rs, который является основным входом, и у вас также есть внутренний модуль. Внутренний модуль имеет реализацию без макроса near_bindgen, поэтому ни один из этих методов не будет виден для вызова по контракту кем-то еще в цепочке. Их можно вызывать только внутри этого контракта, чтобы они не генерировали форматы JSON и не десериализовали состояние. Все они действуют как

обычные методы борьбы с ржавчиной. Как этот контракт работает на высоком уровне, так это то, что по прошествии эпохи вы можете получить определенные награды в качестве валидатора.

Важные методы в контракте

https://gist.github.com/6codingshark/12e7f812f4f64847f616ab495015bae1#file-staking_lib10-rs

У нас есть метод ping, который пингует контракт. Метод ping проверяет, прошла ли эпоха, а затем нам нужно распределить награды. Если эпоха изменилась, то она также будет изменена, потому что может быть некоторое изменение в сумме общей ставки, которую должен поставить контракт. Далее депозит.

https://gist.github.com/6codingshark/a352c0735305cce3e4cd2dd127ba35ac#file-staking_lib11-rs

Метод депозита является платным, что означает, что он может принимать прикрепленный депозит. Это похоже на декоратор Ethereum, который позволяет получать средства только теми способами, которые их ожидают. Так Near_bindgen по умолчанию будет паниковать, если вы попытаетесь вызвать метод, например ping, и привязать к этому методу депозит. Следовательно, кредиторская задолженность позволяет нам прикреплять депозиты. В каждом методе есть внутренний пинг, чтобы убедиться, что мы распределили предыдущие награды, прежде чем менять какую-либо логику. Общая структура такова, что если нам нужно перестейкать, то мы сначала делаем какую-то логику, а потом перестейкаем.

https://gist.github.com/6codingshark/b4b9d0f9da9ce6c963f811c204207bbb#file-staking_lib12-rs

Следующий метод — deposit_and_stake. Это комбинация двух методов. Во-первых, вы вносите остаток на баланс своей учетной записи, и вы также хотите поставить ту же сумму сразу, а не совершать две транзакции. Это также платно, потому что он также принимает депозит.

https://gist.github.com/6codingshark/99b6f5ffa016baaf85cd05f42fff17e1#file-staking_lib13-rs

Следующий — remove_all. Он пытается вывести весь незанятый баланс со счета, который его вызвал. Когда вы взаимодействуете с пулом ставок, вам необходимо взаимодействовать с учетной записью, которой принадлежит баланс. В данном случае это предшественник_аккаунт_ид, и мы в основном проверяем учетную запись, а затем, если можем, снимаем не поставленную сумму. Если не отозвать, то паникует. Например, если он все еще заблокирован из-за отмены ставок менее 4 эпох назад.

https://gist.github.com/6codingshark/75d7d59a033881db2cb5fdefebfaccec#file-staking_lib14-rs

Вывод позволяет вывести только часть баланса.
https://gist.github.com/6codingshark/a927f2cfe985631493d3428663f31913#file-staking_lib15-rs

Затем все незанятые балансы ставятся на стейк_колл, и этот метод используется довольно редко, потому что вы обычно используете депозитную ставку, а в ней уже есть весь баланс ставки.

https://gist.github.com/6codingshark/272eef266c28f9f55c44f3121c833806#file-staking_lib16-rs

Затем в методе ставки вы просто ставите некоторую сумму баланса ставки. Кошелек Moonlight использует отдельную плату для внесения ставки, но для этого они используют пакетную транзакцию.

https://gist.github.com/6codingshark/4529d693e6389c8ea689f2fb6bacc412#file-staking_lib16-rs

Наконец, у вас есть unstake_all, который в основном снимает все ваши доли, конвертируя их в yocto NEAR. Существует вспомогательный метод, который предлагает преобразовать количество моих акций в сумму yocto NEAR и округлить в меньшую сторону, потому что мы не можем дать вам дополнительную сумму за вашу акцию, умноженную на цену. Вот как мы получаем сумму, а затем вызываем отмену ставки на указанную сумму.

https://gist.github.com/6codingshark/9fe6608c9fb6325a3c21da8fec86d7e9#file-staking_internal2-rs

Логика Staking_amount_from_num_shares_rounded_down использует u256, потому что балансы работают на u128. Чтобы избежать переполнения, мы умножаем total_staked_balance на количество долей в u256. Цена округлена в меньшую сторону.

https://gist.github.com/6codingshark/cdcc71908963743a0290272291dc36f5#file-staking_internal3-rs

Вариант с округлением Staded_amount_from_num_shares_rounded_up очень похож, за исключением того, что мы делаем проверку, которая позволяет нам округлить. В конце обоих мы возвращаем его к u128.

Затем выполняем действие отмены ставки, которое очень похоже на unstake_all, за исключением того, что вы передаете сумму.

https://gist.github.com/6codingshark/4f9c0cbcdffe80bf1ff1319ec061724b#file-staking_lib17-rs

Методы получения/просмотра

После этого есть куча методов-геттеров, которые являются вызовами просмотра, которые возвращают вам некоторые суммы. Вы можете получить незанятый баланс учетной записи, баланс учетной записи, общий баланс учетной записи, проверить, можете ли вы вывести средства, общий баланс ставки, который представляет собой общую сумму активной ставки в пуле ставок.

https://gist.github.com/6codingshark/26a52ee31f9f5f9fe997cb2888147ad1#file-staking_lib18-rs

Затем вы можете узнать, кто является владельцем пула ставок, вы можете получить текущую плату за вознаграждение или комиссию пула ставок, получить текущий ключ ставок, и есть отдельная вещь, которая проверяет, приостановил ли владелец пул ставок.

https://gist.github.com/6codingshark/1195a2bf6e0f2422b206f68fc9fded10#file-staking_lib19-rs

Допустим, владелец выполняет миграцию пула ставок на узле. Им необходимо полностью отказаться от ставок, поэтому, например, они могут приостановить пул ставок, который отправит транзакцию состояния в протокол NEAR, а затем не будет повторно делать ставки, пока они не возобновят пул ставок. Тем не менее, вы все еще можете вывести свои балансы, но вы перестанете получать вознаграждения после его прохождения.

https://gist.github.com/6codingshark/9826aa0369acf537faf38107452ffac6#file-staking_lib20-rs

Наконец, вы можете получить удобочитаемую учетную запись, в которой указано, сколько у вас на самом деле токенов для количества акций по текущей цене, и, наконец, указано, можете ли вы вывести средства или нет.

https://gist.github.com/6codingshark/dd2d83de2dd12705ebba75375485cccb#file-staking_lib21-rs

Затем он дает вам количество учетных записей, которое является количеством делегаторов в этом пуле ставок, и вы также можете получить несколько делегаторов одновременно. Это пагинация на большом количестве аккаунтов в пределах неупорядоченной карты. Один из способов сделать это — использовать помощник, который мы называем keys_as a_vector из неупорядоченной карты. Это дает вам постоянную коллекцию ключей из карты, а затем вы можете использовать итератор для запроса учетных записей из этих ключей. Это не самый эффективный способ, но он позволяет реализовать нумерацию страниц на неупорядоченных картах.

Методы владельца

https://gist.github.com/6codingshark/b823d75b728fa422afee8dda89a56320#file-staking_lib22-rs

Есть куча методов владельца. Метод владельца — это метод, который может быть вызван только владельцем. Владелец может обновить ключ стейкинга. Допустим, у них другой узел, и владельцу нужно использовать другой ключ. Все эти методы сначала проверяют, что их мог вызвать только владелец.

https://gist.github.com/6codingshark/38d7b8cc4ad032586e4c0eb11baf917b#file-staking_lib23-rs

Это метод, который изменяет комиссию в пуле ставок. Владелец может немедленно изменить комиссию, которая будет действовать в эту эпоху, начиная с этой эпохи, но все предыдущие комиссии будут рассчитываться с использованием предыдущей комиссии.

https://gist.github.com/6codingshark/7368346c094e7f04962513e965aa65fc#file-staking_lib24rs

Тогда это был метод голосования, который позволил нам перейти ко второй фазе основной сети.

Далее следуют два метода, которые я уже описал, которые позволяют приостановить ставку и возобновить ставку.

https://gist.github.com/6codingshark/54fa59b060196b1a63c35a2bc85ba32f#file-staking_lib25-rs

Остальное только тесты. Большая часть логики происходит внутри.

Тест

У нас также в основном есть симуляционные тесты для конкретного пула. Этот симуляционный тест показывает, как на самом деле будет работать сеть. Сначала мы инициализировали пул.

https://gist.github.com/6codingshark/954df2aa75422c5b55313ae603e6374a#file-general-rs

Боб делегирует. Боб вызвал метод депозита пула, который представляет собой deposit_amount, используя метод депозита. Затем Боб может убедиться, что незанятый баланс работает правильно. Затем Боб ставит сумму. Затем мы проверяем сумму ставки сейчас. Мы убедились, что Боб поставил такую же сумму.

https://gist.github.com/6codingshark/29a5b3e671f91b2a4b4b358f000fce80#file-general1-rs

Боб вызывает метод ping. Наград нет, но в симуляциях награды все равно не работают, поэтому вам нужно сделать это вручную. Мы еще раз убедимся, что сумма Боба осталась прежней. Затем бассейн возобновляется. Мы проверяем, что пул возобновил работу, затем блокируем его до нуля. Затем мы моделируем, что пул получил некоторое вознаграждение (1 NEAR), и боб пингует пул. Затем мы проверяем, что сумма, которую получил Боб, положительна. Это очень простой случай моделирования, в котором говорится, что Боб сначала внес депозит в пул, который проверяет, что пауза и возобновление работают, или имитирует, что это работает, и гарантирует, что пул не делает ставки во время паузы. Затем при возобновлении пул фактически делает ставки. Таким образом, этот тест проверяет не только это, но и то,

что Боб получил награду и получил ее. Есть еще один тест, который проверяет некоторую логику, но он более сложный. Внизу есть несколько модульных тестов, которые должны проверять определенные вещи.

https://gist.github.com/6codingshark/a9ece9b3f40c0d6c49578f7fcd40d3f0#file-staking_lib26-rs

Некоторые из этих тестов не идеальны, но они проверяют определенные вещи, которые были достаточно хороши, чтобы убедиться, что математика складывается.

internal.rs

Internal Ping Method

https://gist.github.com/6codingshark/ef8fea4185171b0df038b7a1703ff80b#file-staking_internal4-rs

Давайте перейдем к internal_ping. Это метод, который любой может вызвать с помощью ping, чтобы убедиться, что награды распределены. Прямо сейчас у нас есть активные пулы ставок, и есть учетная запись, спонсируемая одним из сотрудников NEAR, который в основном пингует каждую ставку в пуле каждые 15 минут, чтобы убедиться, что они распределили вознаграждения для отображения на балансе. Так работает распределение вознаграждения. Сначала мы проверяем текущую высоту эпохи, поэтому, если высота эпохи такая же, значит, эпоха не изменилась, мы возвращаем false, поэтому вам не нужно повторно делать ставку. Если эпоха изменилась, то мы запоминаем, что текущая эпоха (высота эпохи) существует, получаем новый общий баланс счета. Пинг может быть вызван, когда какие-то токены были депонированы через депозитные бюллетени, и они уже являются частью account_balance, и, поскольку пинг был вызван до того, как нам нужно вычесть этот баланс, прежде чем мы распределим вознаграждения. Мы получаем общую сумму, которая есть на счете, включая заблокированный и разблокированный баланс. Заблокированный баланс — это поставленная сумма, которая дает вознаграждение, а разблокированный баланс также может иметь вознаграждение в определенных сценариях, когда вы уменьшаете свою ставку, но ваши вознаграждения все равно будут отражаться в течение следующих двух эпох. После этого они придут к неразыгранной сумме. Проверяем с помощью assert! что общий баланс больше, чем предыдущий общий баланс. Это инвариант, которого требует пул ставок. В тестовой сети было много вещей, которые не сработали с этим инвариантом, потому что у людей все еще были ключи доступа к тому же пулу ставок, и когда он у вас есть, вы тратите баланс на газ, и вы можете уменьшить свой общий баланс без получения вознаграждения. Наконец, мы вычисляем сумму вознаграждения, полученного пулом ставок. Это общий баланс минус предыдущий известный общий баланс, баланс предыдущей эпохи. Если вознаграждение положительное, мы распределяем его. Первое, что мы делаем, это рассчитываем вознаграждение, которое владелец берет себе в качестве комиссии.

https://gist.github.com/6codingshark/2e163935b9832c01e9d4a6eb41883028#file-staking_lib7-rs

Мы умножаем вознаграждение_fee_fraction на общее полученное вознаграждение, и оно аналогичным образом округляется в меньшую сторону: числитель в u256 умножается на значение, деленное на знаменатель в u256.

https://gist.github.com/6codingshark/ef8fea4185171b0df038b7a1703ff80b#file-staking_internal4-rs

Owners_fee — это сумма в yocto NEAR, которую владелец оставит себе. Оставшаяся_награда — это оставшиеся вознаграждения, которые необходимо повторно застейкать. Затем происходит повторная ставка. Владелец получил вознаграждение в yocta NEAR, а не в долях, но поскольку вся логика должна быть в долях, владелец пула ставок покупает доли по цене распределения вознаграждения за пост для остальных делегаторов. Таким образом, num_shares — это количество акций, которые владелец получит в качестве компенсации за управление пулом ставок. Если он положительный, мы увеличиваем количество акций и сохраняем учетную запись владельца обратно, а также увеличиваем общую сумму доли в акциях. Если по какой-то причине при округлении в меньшую сторону этот баланс становился равным нулю, вознаграждение было очень маленьким, а цена за акцию очень большой, и пул получал только нулевое вознаграждение. В этом случае этот баланс просто пойдет на цену за акцию, а не на компенсацию владельцу. Затем мы помещаем некоторые общие данные журнала, которые говорят, что текущая эпоха существует, что мы получили вознаграждение в виде количества акций или токенов, что общий баланс ставок в пуле что-то, и мы регистрируем количество акций. Единственный способ показать количество общих ресурсов внешнему миру — через журналы. Далее, если владелец получил вознаграждение, это говорит о том, что общее вознаграждение составило столько-то акций. Наконец, мы просто запоминаем новый общий баланс и все. Мы распределяли все вознаграждения в постоянное время и обновляли только один аккаунт (аккаунт владельца) для комиссии, и только если комиссия была положительной.

Internal Stake Method

https://gist.github.com/6codingshark/b7504f4262e58f8b5113252e4bb7695d#file-staking_internal5-rs

Internal_stake — это место, где мы реализуем фонд гарантии цены. Допустим, предшественник, в данном случае мы назовем его account_id, хочет поставить определенное количество токенов. Balance на самом деле не является типом JSON, потому что это внутренний метод, поэтому здесь нам не нужен JSON. Мы рассчитываем, сколько акций нужно округлить в меньшую сторону, чтобы поставить заданную сумму, поэтому именно столько акций получит владелец. Он должен быть положительным. Затем мы проверяем сумму, которую владелец должен заплатить за акции, снова округляя в меньшую сторону. Это делается для того, чтобы гарантировать, что когда владелец купил акции и конвертировал их обратно без вознаграждения, он никогда не потерял NEAR за 1 год, потому что это может нарушить гарантию. Наконец, мы утверждаем, что на счету достаточно средств для оплаты взимаемой суммы, и мы уменьшаем внутренний неразмещенный баланс и увеличиваем внутренний баланс количества акций на счете. Затем мы округляем ставку_количества_из_количества_долей_раундед_в большую сторону, чтобы

количество долей фактически округлялось в большую сторону. Эти 1 дополнительный пенни или 1 дополнительный год NEAR будут поступать из гарантированного фонда во время округления акций. Мы взимали с пользователя меньшую плату, но мы внесли больший вклад в сумму из этого 1 триллиона лет NEAR, которую мы изначально выделили для этого. Эта разница обычно составляет всего 1 год от NEAR, которая может быть получена в результате округления в большую или меньшую сторону. После этого идет сумма total_staked_balance и total_stake_shares. Далее мы выпускаем с ними новые акции. Наконец мы ставим лог и возвращаем результат.

https://gist.github.com/6codingshark/52d7a30c4a54610500c7a36edbd17190#file-staking_internal6-rs

Анстейкинг работает очень похоже. Вы округляете до суммы акций, которую вам нужно заплатить. Затем мы вычисляем сумму, которую вы получаете, снова округляя до переплаты вам за это. Это также происходит из гарантийного фонда. Затем мы уменьшаем доли, чтобы увеличить сумму, и сообщаем, когда вы сможете разблокировать баланс между четырьмя эпохами. Unstake_amount округляется в меньшую сторону, чтобы мы откладывали немного меньше, чтобы гарантировать цену других участников пула. Примерно так работает пул ставок и как работает математика. Мы компенсируем ошибки округления из средств, которые мы выделили.

Вывод

Мы обновили ключи для ристретто во время разработки этого контракта, и было удивительно, что нам нужно было это учитывать. В STAKE_SHARE_PRICE_GUARANTEE_FUND 1 триллион yocto NEAR должно быть достаточно для 500 миллиардов транзакций, что должно быть достаточно долго для пула ставок, чтобы его нельзя было пополнить, потому что вознаграждения будут немедленно перераспределены на total_stake_balance при следующем пинге. Мы потратили довольно много времени и усилий на этот контракт, потому что мы провели множество проверок безопасности, в том числе внутренних и внешних, особенно в отношении этой математики. Это было сложно, и некоторые вещи были обнаружены, например, ключ от ристретто, который всплывал во время обзоров. Мы отметили журнал изменений этого контракта, так же в ридми есть куча всего, что всплыло во время разработки, и тестирования на живой системе, но на написание оригинальной версии ушло около недели. Позже мы его почистили, протестировали и улучшили. Потом мы сделали кучу доработок. Приостановка и возобновление были запрошены пулом, потому что в противном случае владелец не имел бы возможности отменить ставку, если его узел выйдет из строя. Они будут атаковать сеть. По сути, эта активная доля будет запрашивать проверку, а не запускать сеть. Раньше у нас не было рубки. Это не было проблемой для участников, но это было проблемой для самой сети. Таким образом, владелец может приостановить стейкинг, если он не хочет запускать пул, который они мигрируют в пул, и общаться как можно больше до этого. Затем мы обновили интерфейс голосования, чтобы он соответствовал заключительному контракту на голосование на втором этапе. Мы добавили вспомогательные методы просмотра, чтобы иметь возможность запрашивать учетные записи в удобном для человека виде. Наконец, были внесены некоторые улучшения в методы пакетной обработки, поэтому Deposit_and_stake, Stak_all, unstake_all и remove_all вместо того,

чтобы сначала делать вызов просмотра, получать сумму и помещать сумму для вызова ставки. Вот как мы это исправили.

https://gist.github.com/6codingshark/4f43617b1741b69a71d3f1d6693591b4#file-staking_internal7-rs

Когда вы делаете ставку, вы не только ставите сумму, мы также прилагаем обещание проверить, была ли ставка успешной. Это необходимо для двух вещей: если вы пытаетесь сделать ставку с недопустимым ключом (не ключом, специфичным для ристретто), то обещание не будет выполнено до выполнения. Он не пройдет проверку перед отправкой, и поэтому вам не нужно проверять его в контракте. Он вернет последний звонок, и все будет хорошо. Также мы ввели минимальную ставку на уровне протокола. Минимальная ставка составляет одну десятую от суммы цены последнего места, и если ваш контракт попытается сделать ставку меньше этой, то действие не будет выполнено, и вы не отправите обещание. Допустим, вы хотите отменить ставку, но ваш баланс упал ниже одной десятой суммы ставки. Действие стейкинга может завершиться неудачей, и вы не сможете отменить стейкинг, в то время как вам это нужно, чтобы гарантировать, что анстейкинг произойдет. В этом случае у нас есть этот обратный вызов, который проверяет, успешно ли завершено действие стейкинга. Этот обратный вызов в основном проверяет, что в случае сбоя и положительного баланса нам нужно отменить ставку. Таким образом, он вызовет отмену ставки для действия, когда сумма ставки равна нулю, чтобы убедиться, что весь баланс освобожден. Вы можете вывести средства в 4 эпохи во время тестирования этих контрактов, которые мы проводили в тестовой сети бета-версии 9 перед техническим обслуживанием. Контракт был готов примерно летом, поэтому тестирование этой итерации заняло, вероятно, около 2-4 месяцев из-за сложности, связанные с взаимодействием с протоколом. Было довольно много обучения от разбивки на страницы до вспомогательных методов и объединения некоторых вещей. Одна вещь, которая была бы действительно хороша, иметь ставку или депозит и ставить все на контракт блокировки. Прямо сейчас вам нужно вручную указать, сколько вы хотите поставить на контракт блокировки, но было бы здорово, если бы вам не нужно было думать о вашем yocto NEAR и о том, сколько он заблокирован для хранения. Вы просто хотите поставить все из своего локапа, но, поскольку он уже был развернут, было слишком поздно думать об этом. Есть также некоторый газ, который жестко запрограммирован, и с общим снижением комиссии эти номера нельзя изменить, потому что они уже в цепочке.

https://gist.github.com/6codingshark/a8bc875ff355e92970e80093ea032fef#file-staking_lib27-rs

Таким образом, голосование не важно, но метод ON_STAKE_ACTION_GAS требует, чтобы у вас было большое количество для каждой ставки, и вы не можете его уменьшить. Рискованные действия при каждом вызове по этому контракту потребуют от нас большого количества газа, и проблема в том, что это расточительно. Допустим, мы договорились сжечь весь газ, этот газ всегда будет сожжен и потрачен впустую, плюс он ограничивает количество транзакций, которые вы можете поместить в один блок, если мы ограничиваем газ на основании этого случая. Было много итераций по тестированию контракта с использованием среды имитационного тестирования, которую мы значительно улучшили. Если мы в конце концов доберемся до контрактов

блокировки, вы увидите, насколько структура контрактов блокировки улучшилась по сравнению с этой.