

Hashgraph-inspired consensus flavors

Abstract:

Ниже мысли, пришедшие в голову после просмотра и (наконец) понимания видео:

Dr. Leemon Baird: How Hashgraph Works - A Simple Explanation w/ Pictures

<https://www.youtube.com/watch?v=wgwYU1Zr9Tg>

Ключ к (не)полному пониманию видоса для меня был в timescode 28:22 - на том слайде ключевая информация о 'round created', которую почему-то не озвучивают и толком не объясняют; однако на слайде можно прочитать, хоть и в минималистичном виде.

А также после того, как я выяснил, что сам алгоритм распространяется под лицензией Apache 2.0, то есть можно сделать свою имплементацию и модифицировать как захочешь.

см: <https://docs.hedera.com/hedera/core-concepts/hashgraph-consensus-algorithms>

или же мой пружф-форк здесь:

<https://github.com/codekrolik2/hedera-docs/blob/master/core-concepts/hashgraph-consensus-algorithms/README.md>

Hashgraph это довольно сложный алгоритм, и многие из движущихся частей можно если и не заменить, то определенно потвикиать по следующим причинам:

- Алгоритм не настаивает ни на каких низкоуровневых деталях имплементации, то есть настолько, насколько все вводные, описанные в алгоритме соблюдаются, (или их эквиваленты) имплементировать можно миллионом разных способов.
- Улучшение производительности - более простой траверс последовательностей событий, фильтрация получаемой информации, и пр.;
- Упрощение структур/утилитарных алгоритмов при использовании алгоритма в доверенном пространстве, например если мы создаем кластерную БД и все ноды по определению принадлежат нам;
- Возможность наблюдения за поведением системы с разными настройками протокола

Их ранние презентации ссылаются на некую легендарную (т.к. о ней слышали только в легендах) программу HashgraphDemo app (в вышеупомянутом видео таймкод 51:17), но найти ее я так и не смог, поэтому надо записать свою не только потому, что хотелось бы сделать кучу всяких крутилок и переключателей, чтобы тестировать алгоритм, но и так как оригинальная программа если и может быть найдена (чего мне не удалось), то наверное существует без поддержки в заброшенном виде, и похоже, что ее публикацию посчитали за бизнес-ошибку и теперь пытаются это скрыть, к тому же она заточена явно на базовый алгоритм и блокчейн, а хотелось бы также поиграться со своими идеями и расширениями; в общем, 100 причин чтобы не тратить больше времени на ее поиск.

Bottomline - пилим свою HashgraphDemo и свою имплементацию алгоритма, с игрой 21 и гейшами.

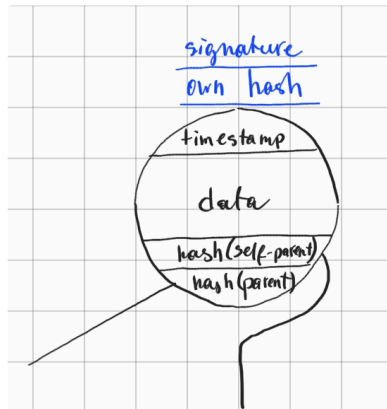
По структуре самого event - их можно разделить на 2 группы

1. Эвенты, содержащие данные, т.е. например для блокчейна это транзакции. Назовем упрощенно data/new.
2. Эвенты без данных, содержащие лишь информацию о распространении госсипа. Назовем упрощенно gossip/reference.

По идее сами данные не обязательно эмбеддить в структуру эвента, это могут быть какие-нибудь референсы типа id объектов во внешнем хранилище, и сэкономить на трафике распространения госсипа.

Event-ы и как я их вижу.

Рассмотрим структуру эвента, которую они показывают в видеосе:



То, что черным - есть на картинке в видеосе, а то, что синим - они об этом говорят, но почему-то остается впечатление, что не считают это частью event-a, по непонятным причинам.

Если мы задумаемся в то, что же такое `self-parent-hash`, `other-parent-hash` и `own-hash`, то станет понятно, что это не что иное, чем уникальный идентификатор event-a и ссылка на предыдущие event-ы по их идентификаторам. Это имеет свои плюсы, т.к. жестко приаттачен к данным, т.е. выполняет также роль чексуммы (как и любой hash), но имеет и свои недостатки - например, не является монотонно растущими и не содержит другой метаданных, кроме как отсылка на родителя. Поэтому, навскидку, для выяснения откуда конкретный event взялся, нужны будут лукапы, а для получения последовательности эвентов от конкретной ноды - траверс графа, так как таймстемпы могут совпадать (а кроме того, если я правильно понял, ничто им не мешает еще и скакать взад-вперед, причем на отдельно взятой ноды, таким образом не гарантируя формирования упорядоченной последовательности).

Помимо того, вспомним, что и у хешей в теории могут быть коллизии, например md5.

Из жизни: один мой коллега как-то раз написал систему, где использовал в качестве первичного ключа md5 хеш, и они у него таки совпали в проде, что оказало влияние на его настроение в тот момент - тут надо заметить, что он решил эту проблему, поменяв один хеш md5 на несколько, типа комбинация md5+sha1 или что-то там еще, но когда я ему сказал, а почему бы не использовать нормальные уникальные id, посмотрел на меня как на мудака, что-то пренебрежительно бросил про вероятности и пресек развитие темы, в общем, я так и не понял, чего же я такого сказал неправильного 😊 При этом, справедливости ради, следует заметить, что у него больше не происходило коллизий. Возможно, что и в Heder-е у них тоже не будет коллизий, возможно, что даже никогда. Такая вероятность, безусловно, существует.

Но перейдем к практическому вопросу - незаменим ли в таком случае хэш? и нельзя ли его заменить на пару id/checksum? - и появляется такая мысль, что если мы в небезопасном окружении типа блокчейна, где есть ноды, от которых можно ожидать отсылку всякой дичи, лишь бы сломать всю распределенную систему, то могут быть преимущества в том, что наш id крепко завязан с данными, и значительно сложнее будет начать генерировать какие-то злонамеренно сформированные или дублированные hash, нежели id. В то же время в ситуации "собственный кластер БД" можно оптимизировать на том, чтобы не выполнять трудоемкое вычисление hash, и при этом добавить полезные свойства, как - референс на ноду-создателя, свойство монотонного прироста значения, и т.п.

Также, для оптимизации производительности локального доверенного кластера мы можем отказаться и от полноценной криптографической подписи, например, если вделаем создателя в айдишник - и это тоже экономия вычислительных ресурсов.

Замечу, что как было показано выше (или же я попытался, как мог), алгоритм Hashgraph (во всяком случае теоретически) может быть реализован без использования хеш-функций. Засчитываем это, как каламбур.

Если заменить все-таки на id, то можно называть более понятно (according to me)

own-hash	id or event_id
----------	----------------

<code>self-parent-hash</code>	<code>previous_event_id</code> - если представить эвенты, исходящие с одной ноды, как связный список, то структура этого списка формируется именно этим полем.
<code>other-parent-hash</code>	<code>@Nullable outer_event_id</code> - референс на порождающий эвент извне, принадлежащий последовательности id, существующей на другой ноде.

О parent-референсах

Очевидно, что помимо самого первого события (появления ноды), у всех последующих событий ноды всегда будет `self-parent`.

При этом не совсем понятно, может ли быть такая ситуация, когда `other-parent IS NULL`. Если посмотреть их видос выше, то в их представлении графа нет таких нод, у которых бы был только `self-parent`, а `other-parent` бы отсутствовал. Я не знаю, с чем это связано, в том числе еще туману напускает тот факт, что у них на той диаграмме непонятно в каком эвенте есть данные, а в каком нет, но наверное, какие-то данные там все же есть.

Лично мое видение этого алгоритма в том, что у любого события `data/new` всегда будет только `self-parent`, и никогда не будет `other-parent`. Возможно, их имплементация оперирует на иных предпосылках, а значит у них неправильно. (*“неправильно” as in - не так, как будет у нас*).

К примеру, почему это вдруг я не могу просто взять и создать данные, а должен ждать какого-то там события госсип от кого-то там, да даже если оно и приходит часто и регулярно, зачем, вообще, это делать частью создания данных. Мне думается, лучше будет так: надо создать данные - взял и создал, вот сам и отправляй госсип другим, так явно и проще, и понятнее, и быстрее.

Подобную discrepancy со стороны их диаграммы/реализации просто игнорируем, и реализуем так, как видим сами.

Монотонно прирастающие id, групповые версии и их роль в оптимизации получения изменений графа

Если мы меняем хеши на монотонно прирастающие id

НЕЗАКОНЧЕНО, Я ДОЛЖЕН ОПУБЛИКОВАТЬ ЭТУ ВЕРСИЮ ASAP