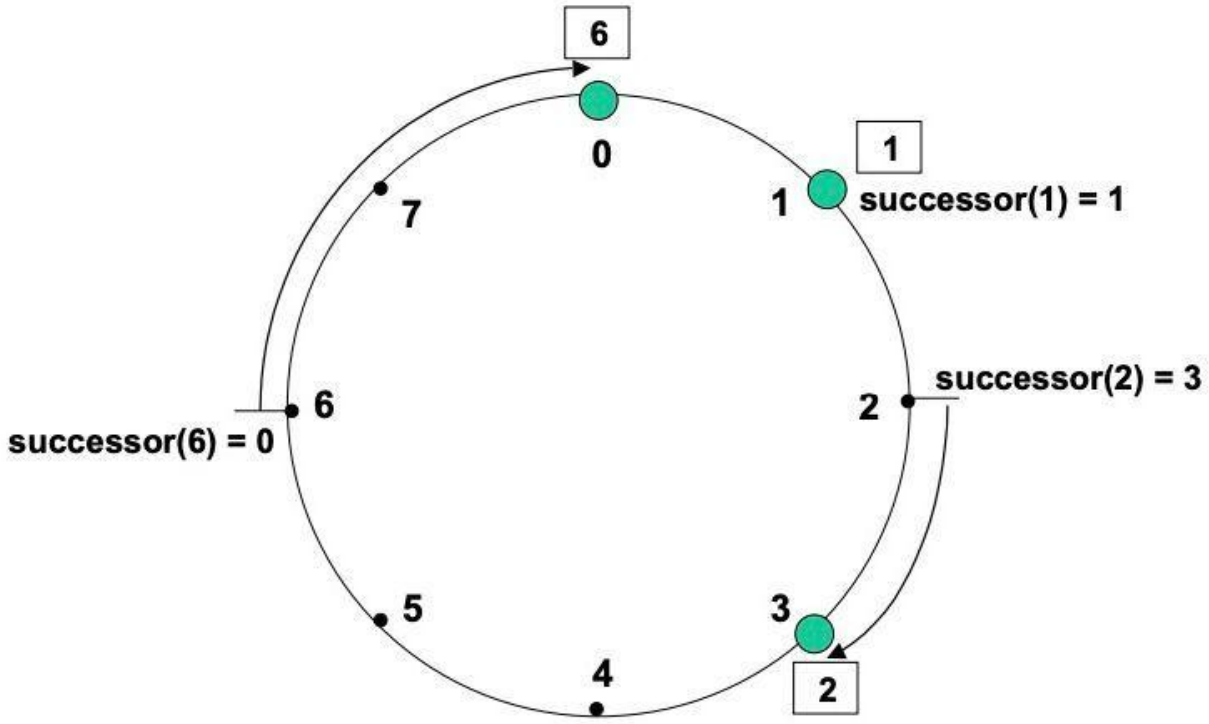
Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Драганов Александр Андреевич БПМИ 203

Итак, в данной статье описывается Chord-алгоритм дизайна p2p системы, где демонстрируется децентрализованная хеш-таблица, {key, value} хранилище. Со следующими свойствами:

- Все узлы у нас равноправны, нет каких-то особенных узлов
- Чтобы найти какой-то ключ нам максимум понадобится O(log(N)) шагов, где N - количество узлов
- При добавление / удаления узла нам понадобится O(log^2(N)) шагов

Где хеш-таблица - там и хеш-функция, поэтому для подробного понимания процесса предоставим пример: пусть наша хеш-функция H является отображением из key -> m-битное число. Тогда давайте представим нашу систему в виде 2^m вершин, выглядеть она будет следующим образом:

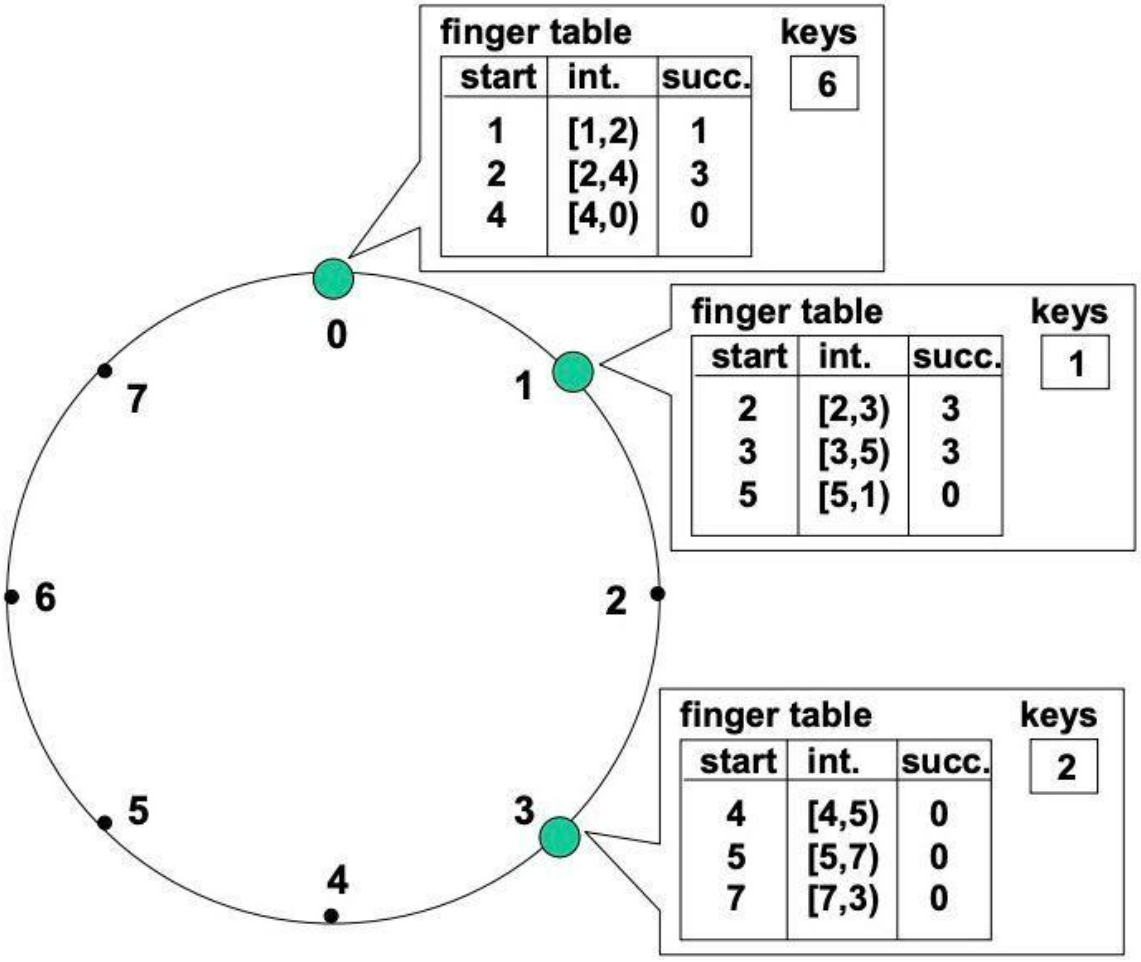


В данном случае у нас хеш-функция возвращает трех-битное число. Мы видим на картинке 8 вершин, зеленые вершины символизируют node, где что-то лежит, а черные - фиктивные. В момент когда к нам приходит ключ key (для простоты будем считать что это уже хеш от ключа), мы хотим найти ближайшую к нему

node (по часовой стрелке) и засунуть его туда, таким образом у нас в node-0 лежит ключ 6, в node 1 лежит ключ 1, в node 3 лежит ключ 2.

Понятно, что запросы найти ключ key будут приходить только в зеленые вершины (черные у нас фиктивные). Тогда самый простой вариант для каждой node сохранять следующую по-порядку, таким образом каждая node будет уметь общаться только с соседней, и поиск нужной nod-ы у нас будет занимать в худшем случае O(N), где N - количество node.

Попробуем это ускорить, давайте в каждой node хранить m значений (напоминаю, 2^m - размер хеш-функции), пусть мы сейчас в node под номером u, i-е значение говорит нам номер ближайшей по часовой стрелке вершины от вершины (2^i + u) mod 2^m. Тогда картинка уже будет выглядеть следующим образом:



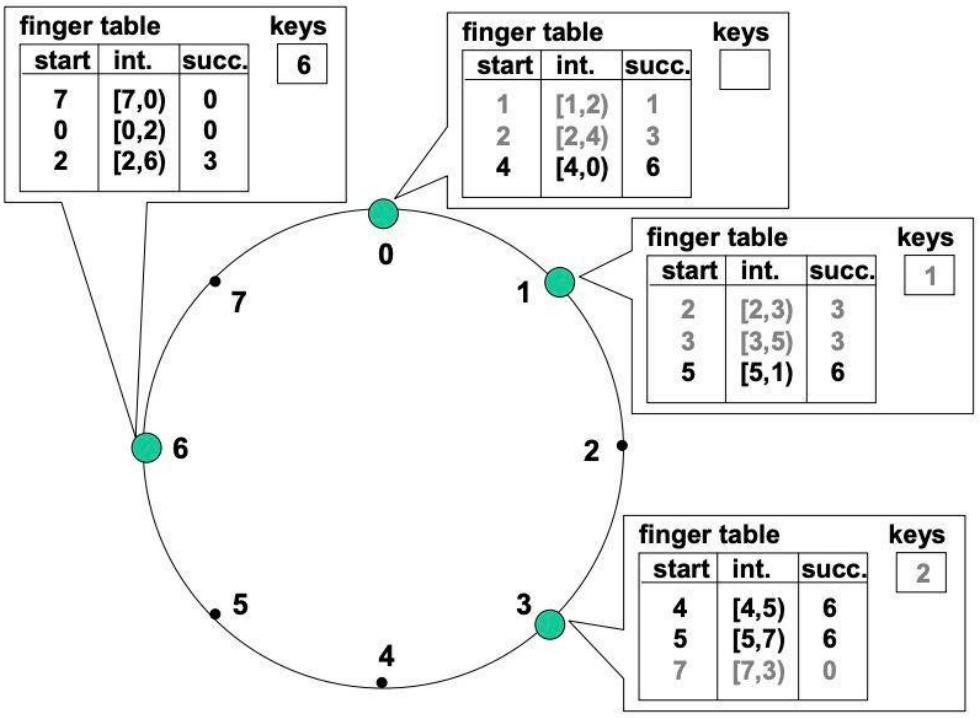
Для вершины 0:

- Если мы отойдем на расстояние 2^0 окажемся в вершине start = 1 => ближайшая к нам node будет 1
- Если мы отойдем на расстояние 2^1 окажемся в вершине start = 2 => ближайшая к нам node будет 3
- Если мы отойдем на расстояние 2^2 окажемся в вершине start = 4 => ближайшая к нам node будет 0

Аналогично и для других вершин. Далее будем опираться на равномерность хеш-функции и считать что m ~ O(log(N)).

Теперь давайте промоделируем запрос поиска ключа из какой либо вершины. Пусть запрос пришел в вершину u с ключом key, тогда мы должны посмотреть в нашу таблицу из m значений и найти максимальное расстояние на которое мы можем сдвинуться, чтобы мы не прошли вершину, в которой может лежать key. Тогда расстояние между вершиной u и вершиной key будет равно: (key - u + 2^m) mod 2^m, давайте найдем СТАРШИЙ бит в этом расстояние, пусть он будет под номером i. Тогда мы можем за O(1) перейти из вершины u в вершину с номером, не меньше чем (u + 2^i) mod 2^m. Таким образом мы МИНИМУМ сократим наше расстояние в два раза. Отсюда и появляется оценка на количество таких переходов - O(log(N)).

Рассмотрим как добавлять узел в нашу систему.



Пусть мы хотим сделать вершину u (= 6) зеленой, давайте тогда за O(log(N)) найдем ближайшую по часовой стрелки зеленую вершину запросом поиска ключа key (= u). Нам выдаст вершину 0, действительно, если бы ключ 6 лежал где-то => он лежал бы в вершине 0. Теперь давайте построим нашу таблицу для вершины 6, используя таблицу для вершины 0. Очевидно все значения, которые меньше нашего расстояния между вершиной 0 - мы заполним 0, ведь это ближайшая к нам вершина. Как тогда заполнить остальные значения? Возьмем первый такой i, что мы перепрыгнем через вершину где мы должны были бы лежать, тогда найдем к нему ближайший справа просто запросом поиска ключа key = (u + 2^i) mod 2^m. За O(log(N)). Таким образом для того чтобы построить таблицу для вершины u которую мы хотим добавить в систему нам будет стоить O(log^2(N)). Вопрос о переносе некоторых ключей из вершины 0 остается незакрытым в статье, поэтому сложность этого вопроса я не стал считать, очевидно что в реальном мире это не O(1), но видимо мы опираемся что у нас хорошая хеш-функция и распределение равномерное.

Теперь рассмотрим как мы будем менять остальные таблицы? Тут мы будем тоже обходиться запросами поиска ближайшей зеленой node.

1) Пройдемся циклом от i = [0; m), сделаем запрос поиска ближайшей зеленой node с ключом равным (u - 2^i + 2^m) mod 2^m (просто отступим против часовой стрелки на расстояние 2^i). Если ближайшая будет вершина в которой мы бы лежали => значит наше добавление изменит все значения в табличке под индексами j = [0; i]. Работать это будет за O(log^2(N))
2) Как только мы нашли первую такую вершину - переходим в нее, пусть это вершина v. Давайте тогда подумаем, какие вершины могли бы смотреть на нашу вершину u? Очевидно такие вершины x, что (x + 2^i) mod 2^m > v и (x + 2^i) mod 2^m < u для какого-либо i. Так как шагать мы можем только на 2^(m - 1) таких вершин не больше m => не больше O(log(N)). Для того чтобы найти каждую такую вершину будем честно от текущей вершины v шагать назад к ближайшей, согласно шагу 1 (как в нем и описывается, за O(log^2(N)). В статье уверяют если у нас хорошая хеш-функция и равномерное распределение => таких вершин будет немного, поэтому асимптотика этого пункта O(log^2(N)).

Операция удаления очень похожа на операцию добавления, только в инверс-порядке, так как мы явно обходим все node которые с нами соседи - будем также их обходить но совершать другие действия. Таким образом мы построили систему, о которой было оговорено в самом начале.

Мое впечатление от статьи

Статья мне понравилась, но честно сказать я в ней не нашел все ответы на вопросы которые оговаривались в самом начале.

1) В статье не сказано как оптимально перемещать ключи, конечно я понимаю что опираясь на равномерное распределение хеш-функции как-будто бы в нашей системе не может быть в одной вершине много ключей, но я могу сделать очень много контр-примеров когда могут, к примеру чтобы изначально сеть была из одного компьютера, и после множества запросов я начал добавлять остальные => тогда ключи будут шагать туда сюда
2) Опираясь на пункт 1 мне стало грустно, так как в моей голове решение этой проблемы => добавить какой-то еще node, которые будет все это балансить, но тогда система уже не децентрализованная
3) В статье вообще много тезисов о том что вот мы можем найти вершину за O(log^2(N)), но то что вершин не слишком много (их к примеру не N), которые надо обновить, опять же не доказывается а говорится о хорошей вероятности.
4) В целом мне очень понравилось, я бы даже взял себе такой проект, а то я не очень понял в конечном итоге хорошая это технология или нет, если алгоритм децентрализованный, поэтому хотелось бы разобраться

Статью брал отсюда:

https://docs.google.com/document/d/18NZKPBIVNFthc99ANPymzVfMDnrPOoDIiyCVN8hHMt0/edit