

Распределенные алгоритмы и системы

mk.cs.msu.ru → Лекционные курсы → Распределенные алгоритмы и системы

Блок 43

Консенсус:
Паксос

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

Вступление и общие особенности

Паксос — это семейство алгоритмов консенсуса, широко применяющихся на практике, например, для

- ▶ согласования транзакций в распределённых базах данных,
- ▶ согласования истории блоков в блокчейне,
- ▶ согласованного доступа к файлам в распределённых файловых системах (например, в менеджере блокировок Google Chubby для Google File System) и
- ▶ в целом в тех случаях, когда требуется разумный и достаточно эффективный консенсус в сети, допускающей много «сильных» неисправностей

При этом, ввиду **известных результатов о невозможности консенсуса**, придётся «пожертвовать» свойством завершаемости: алгоритм будет завершаться только в том случае, если в сети достаточно много исправных узлов и передача сообщений между ними происходит достаточно быстро

При этом в алгоритме возможны любые решения, не только 0 и 1

Вступление и общие особенности

Название «Паксос» (Paxos) можно считать бессмысленным: его предложил в 1998 году Л. Лэмпорт в первом описании этого алгоритма в рамках иллюстрации его как схемы голосования в парламенте на (реально существующем) греческом острове Паксос

Далее обсудим вариант алгоритма Паксос, предложенный Л. Лэмпортом в 1998 году («Part-time parliament»), переизложенный в 2001 году («Paxos made simple»)

В этом алгоритме используется особая модель отказов:

1. Узлы могут выходить из строя и восстанавливаться с частичным восстановлением состояния на момент сбоя
 - ▶ *О том, какие именно части состояния восстанавливаются, будет рассказано позже*
2. Сообщения могут доставляться сколь угодно долго, теряться и дублироваться, но не могут искажаться
 - ▶ *То есть, в частности, в сети нет византийских отказов*

Роли

Узлам сети раздаются роли в голосовании:

- ▶ **Заявитель**
Заявитель **выдвигает** на голосование **заявку** — некоторое значение
- ▶ **Избиратель**
Избиратель **одобряет** или **отвергает** заявку
- ▶ **Слушатель**
Слушатель старается узнать, какое значение выбрали избиратели

Одному узлу может быть присвоено и несколько ролей

Все узлы знают свои роли и роли всех других узлов

T1 и выбор большинством голосов

Самый простой способ организовать выборы — это предоставить выбор одному избирателю:

- ▶ Заявители доносят свои заявки до единственного избирателя
- ▶ Единственный избиратель решает, какую заявку одобрить

Так как требуется одобрить заявку даже в том случае, если она всего одна, то возникает следующее требование:

T1: избиратель должен одобрить первую принятую заявку

Но в такой схеме выборов выход из строя всего одного узла-избирателя приводит к тому, что никакое значение не будет выбрано

Чтобы преодолеть эту проблему, можно увеличить число избирателей:

1. каждый избиратель может одобрить или отклонить заявку,
2. но заявка **выбирается** только в том случае, если она одобрена **большинством** голосов избирателей

T1 и выбор большинством голосов

Если каждый избиратель будет одобрять **только** первую принятую заявку, то можно легко придумать вычисление, в котором ничего не будет выбрано — например:

- ▶ К пяти избирателям поступили заявки a , b и c
- ▶ Два избирателя первой приняли заявку a , два — заявку b и один — заявку c
- ▶ Ни для одной заявки не набрано большинство голосов

Чтобы преодолеть эту проблему, следует разрешить избирателю одобрять не только первую заявку, но и другие

Но тогда следует ограничить свободу действий каждого избирателя так, чтобы только одно значение могло быть одобрено большинством

Нумерация заявок

Чтобы можно было отслеживать статус конкретных заявок (как с одинаковыми, так и с разными значениями), пронумеруем их: при выдвижении заявитель присваивает заявке **номер** так, чтобы

- ▶ различные заявки имели различные номера и
- ▶ номера заявок были линейно упорядочены

Например, если узлам сети присвоены уникальные идентификаторы из линейно упорядоченного множества, то

- ▶ номером заявки может служить пара $[p, n]$, где p — идентификатор заявителя и n — то, какую по счёту заявку он выдвигает, и
- ▶ линейно сравнивать такие номера заявок можно, например, лексикографически

Таким образом, заявка представляет собой пару (v, n) , состоящую из **значения** v и **номера** n

T2, T2и

Ограничить свободу действий избирателя можно при помощи такого требования:

T2: если выбирается заявка (v, n) , то заявка (w, m) для $m > n$ может быть выбрана только в том случае, если $w = v$

T2 гарантирует, что может быть выбрано (и стать решением) только одно значение, хотя, быть может, и много раз для разных заявок

Чтобы соблюсти T2, ограничим свободу действий избирателя так:

T2и: если выбирается заявка (v, n) , то заявка (w, m) для $m > n$ может быть одобрена избирателем только в том случае, если $w = v$

Утверждение. Если T2а верно для каждого избирателя, то верно и T2

T2з

Так как обмен сообщениями асинхронный и возможны потери сообщений, то T1 и T2a могут друг другу противоречить, если не ограничивать свободу выдвижения заявок:

1. Выдвигаются заявки $(a, 1)$ и $(b, 2)$ для трёх избирателей p, q, r
2. Заявка $(a, 1)$ одобряется p и q и выбирается большинством голосов
3. После этого r принимает заявку $(b, 2)$ и по T1 обязан её одобрить, а по T2a — отклонить

Чтобы таких противоречий не возникало, ограничим свободу действий заявителя так:

T2з: если выбирается заявка (v, n) , то заявка (w, m) для $m > n$ может быть выдвинута заявителем только в том случае, если $w = v$

Утверждение. Если T2з верно для каждого заявителя, то T2a верно для каждого избирателя

T2з

Чтобы более «конструктивно» соблюсти T2з, добавим следующее требование:

T2к: заявка (v, n) может быть выдвинута только в том случае, если существует множество избирателей S , содержащее более половины избирателей и такое что

- ▶ либо ни один узел из S не одобряет заявки с номерами, меньшими n ,
- ▶ либо для заявки (w, m) , одобренной кем-либо из S , с наибольшим номером m , меньшим n , верно $w = v$

Утверждение (Задача 1). Если для каждого заявителя верно T2к, то для каждого заявителя верно и T2з

Но так как заявка с меньшим номером может быть выдвинута хронологически позже заявки с бóльшим номером, и тем более может быть позже одобрена избирателями, то для соблюдения T2к может быть затруднительно (*и неэффективно*) отслеживать значение и принятие такой «запаздывающей» заявки

Бронирование заявки

Вместо отслеживания статуса заявки можно добиться соблюдения T2к при помощи бронирования заявки:

- ▶ Заявитель отправляет произвольному большинству избирателей номер n заявки, которую хочет выдвинуть
- ▶ Избиратель, получив такой номер,
 - ▶ обещает больше не одобрять заявки с номерами, меньшими n , и
 - ▶ отправляет в ответ одобренную им заявку (v, m) с наибольшим номером m , таким что $m < n$, или сообщение об отсутствии таких заявок
- ▶ Если в ответ на бронирование заявитель получил хотя бы одну заявку, то перед рассылкой заявки он изменяет значение заявки согласно T2к на значение полученной заявки с наибольшим номером

Утверждение. Для каждого заявителя, бронирующего заявку перед выдвижением, верно T2к

T1o

Теперь возникло новое противоречие — между T1 и обещанием, данным избирателем при бронировании:

- ▶ Бронируется заявка $(a, 2)$
- ▶ При бронировании избиратель p обещает не одобрять заявки с меньшими номерами
- ▶ Бронируется и выдвигается заявка $(b, 1)$
- ▶ По T1 избиратель p вынужден одобрить $(b, 1)$ вопреки обещанию

Чтобы исключить такие противоречия, «ослабим» T1:

T1o: избиратель одобряет первую принятую заявку с номером, не меньшим всех забронированных (если ни один номер не забронирован, то просто первую принятую, как в T1), и отклоняет все заявки с меньшими номерами

Оптимизированное бронирование

Перед окончательной формулировкой действий отправителей и избирателей добавим следующую **оптимизацию бронирования**:

- ▶ Если избиратель ответил на бронирование номера n , то он не отвечает на бронирование номеров, меньших n
- ▶ Если избиратель принял заявку с номером n , то он не отвечает на бронирование номера n

С учётом такой оптимизации, избирателю достаточно хранить только один (самый большой) забронированный номер, игнорируя запросы на бронирование всех меньших

Утверждение. Избиратель с оптимизацией бронирования и без неё одинаково принимает и отклоняет заявки согласно T1o

Восстановление данных после отказа

Для соблюдения $T1_0$ и $T2_k$ с учётом отказов и потери сообщений избиратель при выходе из строя с последующим восстановлением должен сохранять (восстанавливать)

- ▶ принятую им заявку с наибольшим номером и
- ▶ наибольший забронированный номер

Заявитель при выходе из строя с последующим восстановлением должен сохранять только номер последней заявки

Алгоритм Паксос: код заявителя

Переменные заявителя p :

- ▶ $n_p \in \mathbb{N}_0 = 0$

Процедура выдвижения значения v заявителем p :

1. $n_p := n_p + 1$;
2. Произвольно выбранному большинству избирателей отправить запрос на бронирование номера (p, n_p) : (**reserve**, $[p, n_p]$)
3. Принять подтверждение бронирования (**(ack, x)**, где $x = (w, m)$ или $x = \perp$) от большинства избирателей
4. Если хотя бы одно подтверждение содержит заявку, то среди принятых заявок выбрать (w, m) с наибольшим номером, и отправить (**accept**, (w, n)) избирателям, от которых приняты подтверждения
5. Иначе отправить этим избирателям (**accept**, (v, n))

Кроме того, в любой момент заявитель может принудительно завершить процедуру выдвижения значения («забросить» заявку) и выдвинуть новое

Алгоритм Паксос: код избирателя

Переменные избирателя p :

1. z_p — заявка или \perp , начальное значение \perp
2. $n_p \in \mathbb{N}_0 \cup \{\perp\} = \perp$

Действие приёма (**reserve**, n) избирателем p :

1. Если $n_p < n$:
 - 1.1 Отправить (**ack**, z_p) в ответ
 - 1.2 $n_p := n$;

Действие приёма (**accept**, (v, n)) избирателем p :

1. Если $n_p \leq n$:
 - 1.1 $z_p := (v, n)$;
 - 1.2 $n_p := n$;

Алгоритм Паксос: итог

Утверждение (Задача 2). Если в некоторой конфигурации вычисления алгоритма избирателями выбрано значение v , то и во всех последующих конфигурациях выбирается только значение v

Это утверждение означает, в числе прочего, что выбор значения v избирателями — это **устойчивое свойство** конфигураций

Значит, для достижения консенсуса достаточно время от времени собирать снимок сети в каком-либо узле, проверять по снимку, выбрано ли какое-либо значение, и рассылать это значение во все узлы для принятия решения

Алгоритм Паксос со слушателями

Но для *несколько более* эффективного принятия решения в алгоритм Паксос добавлена роль **слушателя**:

- ▶ Всякий раз, когда избирателем p впервые принимается значение v , пара (v, p) отправляется заранее заданному (параметром алгоритма) числу произвольно выбранных слушателей
- ▶ Слушатель, получив пару (v, p) , рассылает её всем остальным слушателям
- ▶ Если из полученных сообщений следует, что большинство избирателей приняло значение v , это значение рассылается всем узлам для принятия решения

Алгоритм Паксос со слушателями

Задача 3. Положим, что сообщения не теряются, выходить из строя может не более t узлов и заявитель изначально выдвигает значение своей входной переменной. Какое наименьшее число (а) заявителей, (б) избирателей, (в) слушателей и (г) узлов в целом требуется, чтобы алгоритм Паксос обладал свойствами единогласия и невырожденности? Ответ обосновать.

Задача 4. Положим, что сообщения не теряются, узлы не выходят из строя и в сети есть два заявителя и пять избирателей. Предложите способ нумерации заявок, и для него — бесконечное справедливое вычисление алгоритма Паксос со слушателями или без них, в котором решение не принимается (консенсус не достигается).