Лекция 9. Задача о консенсусе.

Содержание

- Понятие консенсуса, число консенсуса.
- Протокол, его состояние и валентность.
- Создание консенсуса из примитивов.
- Консенсус в системах со сбоями. FLP-теорема.

Понятие консенсуса, число консенсуса.

Понятие консенсуса

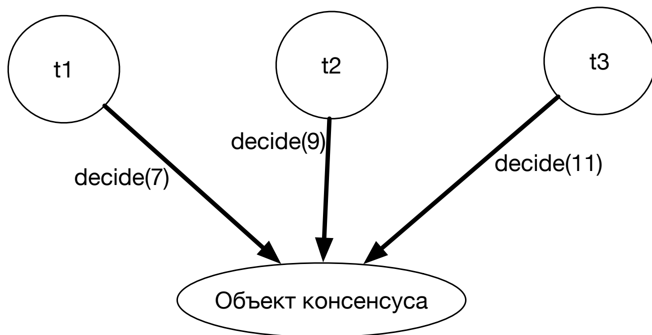
- Пусть имеется `N` заинтересованных потоков.
- Каждый из потоков однократно предлагает своё значение.
- Все потоки принимают решение использовать одно из предложенных значений.

```
class Consensus {
public:
   T decide(T val) = 0;
};

void thread_I() {
   ...
   auto val = c.decide(myval);
   ...
   // Здесь val во всех потоках едино
}
```

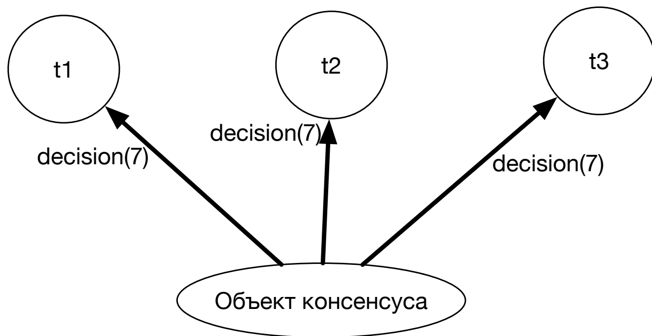
Понятие консенсуса

Потоки предлагают свои значения объекту консенсуса.



Понятие консенсуса

Объект консенсуса возвращает одно из предложенных значений.



Задача консенсуса

- Пусть имеется система из `N` потоков.
- Требуется создать объект консенсуса.
- *Протокол* консенсуса — множество вызовов методов разделяемого объекта (*ходов*).
- Класс `C` решает задачу консенсуса, если для произвольного числа экземпляров класса `C` и атомарных регистров существует протокол достижения консенсуса.
- Число консенсуса для класса `C` — максимальное число потоков, для которого этот класс решает задачу консенсуса.

Композиционность консенсуса

- Если класс `C` можно воспроизвести из конечного количества экземпляров класса `D` и атомарных регистров, то если класс `C` решает задачу консенсуса для `N` потоков, то класс `D` тоже её решает.

Протокол, его состояние и валентность.

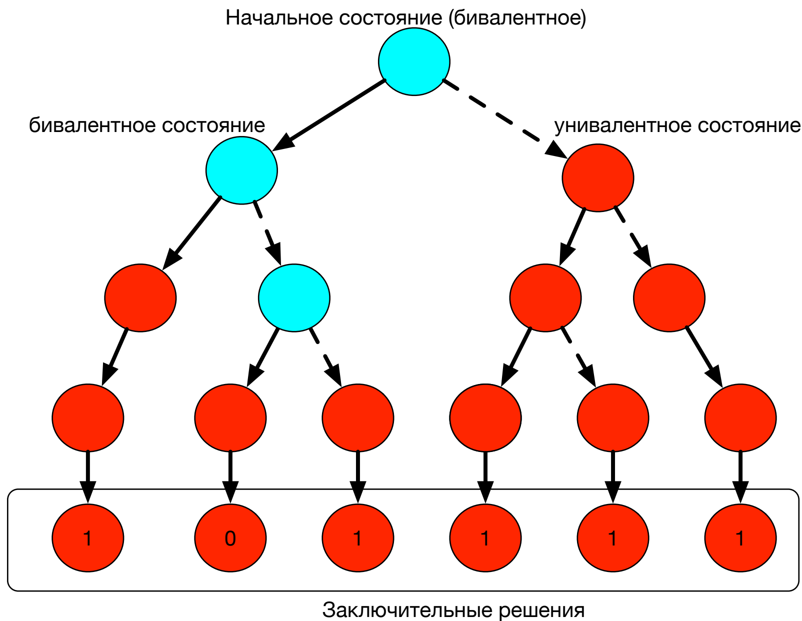
Протокол консенсуса

- *Состояние протокола* — состояние разделяемых объектов и потоков.
- *Начальное состояние* — состояние до совершения первого хода.
- *Конечное состояние* — состояние после завершения фазы *response* всех потоков.

Валентность состояния

- Пусть имеется два потока.
- Пусть возможные значения есть `{0, 1}`.
- *Валентность* — способность совершить очередной ход в точке принятия решения.
- *Унивалентное состояние* — состояние, все исходы из которого приводят к одному и тому же решению.
- *Бивалентное состояние* — состояние, для которого существует несколько допустимых последовательностей, приводящих к различным результатам.

Дерево состояние протокола



Некоторые леммы

- **Лемма.** Любой 2-поточный протокол консенсуса имеет бивалентное начальное состояние.
- **Лемма.** Любой `N`-поточный протокол консенсуса имеет бивалентное начальное состояние.
- **Определение.** Состояние протокола является *критическим*, если оно бивалентное и любой ход любого потока ведёт к унивалентному состоянию.
- **Лемма.** Любой неблокирующий протокол консенсуса имеет критические состояния.

Атомарные регистры

- Регистр *безопасен*, если операция чтения, не соисполняющаяся с операцией записи, возвращает последнее значение операции записи.
- Ни одно чтение не может вернуть значение из будущего.
- Ни одно чтение не может вернуть значение из далёкого прошлого.
- Недопустимо переупорядочивание записей.

Атомарные регистры согласованы по состоянию покоя.

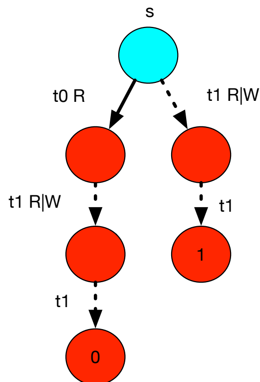
Теорема об консенсусе атомарных регистров

- **Теорема.** Число консенсуса атомарного регистра равно 1.
- *Метод доказательства.* Рассматриваются все возможные выходы из критического состояния.
  1. Чтение одиночного регистра.
  2. Запись различных регистров.
  3. Запись одиночного регистра.

Обозначим потоки как 0-валентный и 1-валентный, по их путям из критического состояния.

Пункт 1.

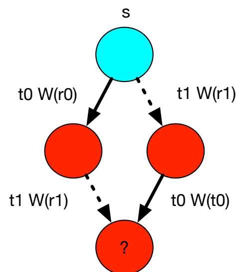
Чтение одиночного регистра.



Первый ход — 0-валентный $\rightarrow$ 1-валентный завершается в 0.
Первый ход — 1-валентный $\rightarrow$ игнорируется начальное чтение 0-валентным.

Пункт 2.

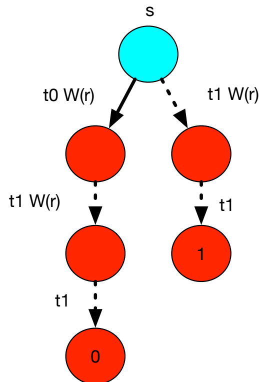
Запись разных регистров.



Первый ход — 0-валентный $\to$ результат 0-валентный.
Первый ход — 1-валентный $\to$ результат 1-валентный.

Пункт 3.

Запись одного регистра.



Первый ход — 0-валентный → 1-валентный завершается в 0.
Первый ход — 1-валентный → не может принять решение, случаи неразличимы.

Следствие из теоремы о консенсусе атомарных регистров

С использованием только атомарных регистров протокол консенсуса построить нельзя.

Следствие 2: **С использованием только атомарных регистров невозможно построить неблокирующие разделяемые структуры данных.**

Протокол консенсуса

```
class ConsensusProtocol: Consensus {
public:
   void propose(T val) {
      proposed[threadId] = val;
   }
   T decide(T val);
protected:
   map<threadId_t, T> proposed;
}
```

Создание объекта консенсуса из примитивов.

Объект консенсуса queue/stack

- **Теорема:** Очередь с двумя читателями имеет число консенсуса по меньшей мере 2.
- **Метод доказательства:** Построить пример, в котором это возможно.
- Создаётся линеаризованная очередь на два читателя с начальным значением {1, 0}.

```
propose: propose[my] = val;
decide:  dequeue() == 0 ? proposed[my] : proposed[his];
```

- **Следствие:** Любая очередь имеет число консенсуса 2.

Объект консенсуса multiple-assignment

- Объекты (m-n)-присвоения содержат N полей и атомарно присваивают M из них.
- **Теорема:** невозможно построить wait-free реализацию такого объекта на атомарных регистрах.

```
class MultipleAssignment23 {
    vector<T> values(2);
    void assign(pair<index, value>, pair<index, value>);
    T get(int i) {
        return values[i];
    }
}
```

Теорема о multiple-assignment объекте

- **Метод доказательства:** построить 2-консенсус реализацию объекта через (2,3) объект.

```
 obj.assign(make_pair<0,a>, make_pair<1,a>);
 obj.assign(make_pair<1,b>, make_pair(2,b>);

auto t = obj.get((taskId+2)%3);
if (t == nullptr || t == obj.get(1)) return my;
else                                 return his;
```

Обобщение теоремы о multiple-assignment объекте

- **Теорема:** `(n, (n+1)*n/2)` объект имеет число консенсуса как минимум `n`.
- **Метод доказательства:** рассмотрим нижнюю треугольную матрицу $R_{i,j} = N \times N$.
- Первый столбец $R_{i,1}$ есть значения эксклюзивных записей i-го потока.
- Остальные элементы $R_{i,j}$ есть значения пересекающихся записей двух потоков i и j, $i > j$.
- **Протокол:** для всех возможных пар потоков установить порядок по регистрам $R_i, R_j, R_{i,j}$

RMW-регистр

Этот класс регистров определяется функцией $f(v)$:

```
atomic<T> RMW(T &v, Func f) {
    auto tmp = v;
    v = f(v);
    return tmp;
}
```

Примеры:

- $RMW_{=}$ — тривиальный RMW, операция чтения.
- RMW_{const} — get-and-set
- RMW_{++} — atomic increment
- RMW_{CAS} — почти RMW.
  $f_{exp,new}(v) = (v == expected)?new : v;$

Теорема о нетривиальном RMW-регистре

- **Теорема:** нетривиальный RMW-регистр имеет число консенсуса не менее 2.
- **Метод доказательства:** через построение модели.
- **Следствие:** нетривиальный RMW-регистр нельзя собрать из атомарных.

Теорема о CAS-регистре

- **Теорема:** регистр, реализующий CAS, имеет число консенсуса, равное бесконечности.
- **Метод доказательства:** перед любым вызовом заполняем регистр уникальным для системы значением U_{all}. Пусть каждый из потоков имеет уникальное для него число U_i. Тогда каждый из потоков при обращении выдаёт $CAS(obj, U_{all}, U_i)$.
- По линеаризации событий ровно один вызов для всех потоков будет успешным.
- Значение будет принято первым по линеаризации потоком.
- **Следствие:** CAS-регистр нельзя собрать из атомарных регистров, очередей, стеков и других комбинаций.

LL/SC регистр

Load linked/ Store conditional

```
class LLSC {
   T LL(T *addr) { // atomic
      saved = addr;
      return *addr;
   }
   bool SC(T *addr, T newVal) { //atomic
      if (*addr == *saved) {
         *addr = newVal;
         return true;
      } else {
         return false;
      }
   }
   T *saved;
};
```

Теорема о LL/SC регистре

- **Теорема:** регистр, реализующий LL/SC, имеет число консенсуса, равное бесконечности.
- **Следствие:** LL/SC-регистр эквивалентен регистру CAS.
- **Преимущества над CAS:**
  1. использует RISC-операции load/store, не нарушая конвейер.
  2. использует два внутренних регистра вместо трёх у CAS.
  3. Лучше подходит для RISC архитектур.

Информация к размышлению: алгоритм булочной Лампорта для n потоков

```
bool    tickets[n]; // init: false
int     seqno[n];   // init: 0
void enter(int x) {
   tickets[x] = true;
   seqno[x] = max all seqno + 1;
   tickets[x] = false;
   for (int i = 0; i < n; i++) {
      while (tickets[i]) { }
      while ((seqno[i] != 0) &&
         ((tickets[j], j) < (tickets[i], i))) { }
   }
}
void leave(int x) {
   seqno[x] = 0;
}
```

Консенсус в системах со сбоями. FLP-теорема.

Асинхронные системы

- Последовательность операций недетерминирована.
- Имеется понятие *буфер* и операции над ним
  - `send(p,m)` — сообщение **m** направляется потоку `p`
  - `m = receive(p)` — сообщение удаляется из буфера и возвращается. Может вернуть `null`.
- сообщения могут не доставляться, порядок доставки не детерминирован (нет `queue`, `stack` ...)

Типа ошибок в асинхронных системах

- Отказы.
  - Протокол `Paxos` Лампорта.
  - Протокол консенсуса Чандра-Туэга.
- Умышленные ошибки
  - Протокол `Byzantine Paxos`.

Протокол Paxos

- Имеется множество процессов, желающих договориться об общем факте (консенсус)
- Система асинхронная и возможны отказы.
  - Участник недоступен (полный отказ)
  - Участник умышленно искажает данные (византийский отказ)
- Каждый участник предлагает свой вариант.
- Общий вариант должен быть принят всеми участниками.
- Общий вариант должен быть одним из предложенных.

Протокол Paxos

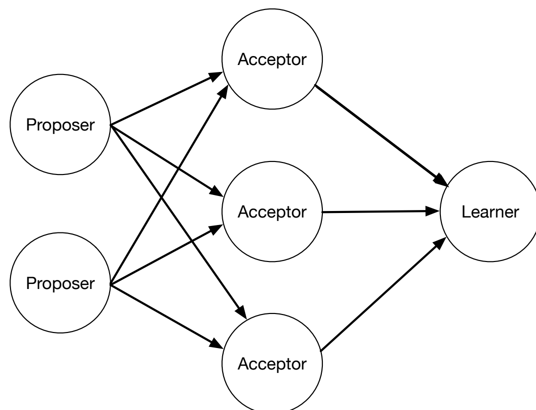
- Суть алгоритма — принятие решение на основе *кворума*.
- *Кворум* — простое большинство голосов.
- Если большинство решает установить предложение, то остальные подчиняются.

Протокол Paxos

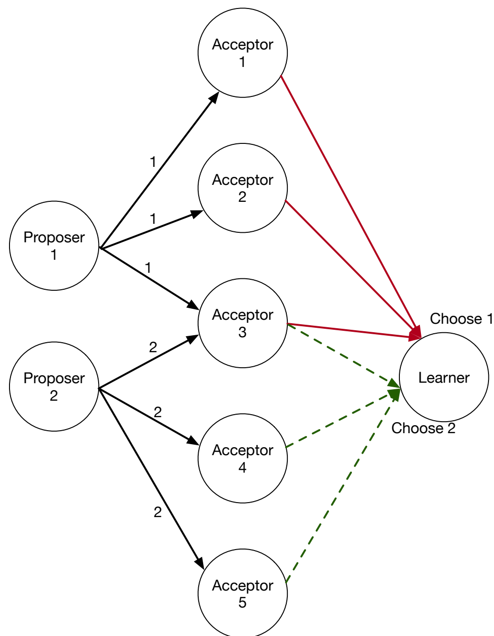
- *клиент (Client)* — взаимодействует с системой, посылая сообщения с запросами и получая сообщения с ответами;
- *пропоузер (Proposer)* — активный участник. Они предлагают свои вариант значений, участвуют в голосовании; Они отвечают клиентам на запросы, исполняя роль *Learner*.
- *акцептор (Acceptor)* — пассивные участники, отвечающие на предложения *пропоузеров*. Они хранят всё выбранное решение или его часть и в конечном счёте узнают о принятом решении.
- *Learner* — принимает утверждённые предложения и запоминает их.

Протокол Paxos

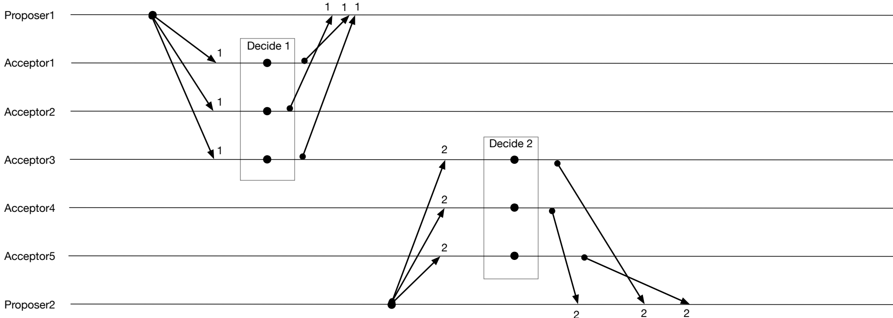
- Задача каждого пропоузера — получить голоса от акцепторов.



Протокол Paxos



Протокол Paxos

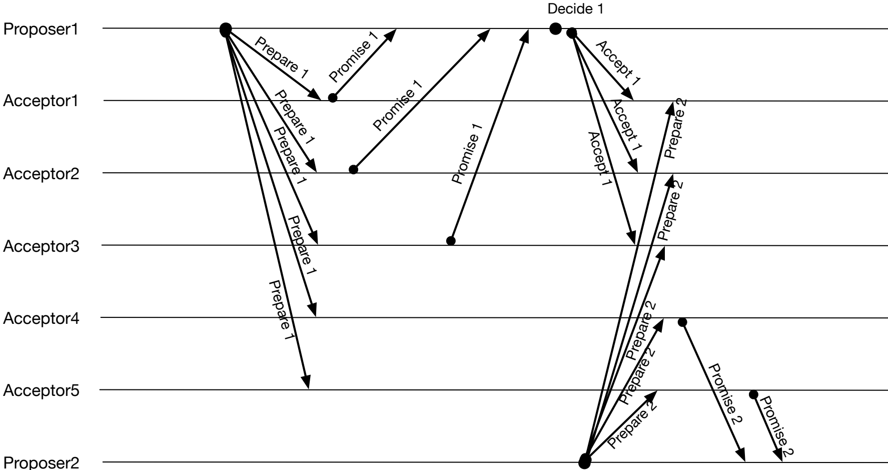


Временная диаграмма неверного принятия решения

Протокол Paxos

- Каждое предложение имеет порядковый номер — отметку времени Лампорта.
- Протокол принятия решения должен состоять из двух фаз:
  - Фаза 1: Посылка сообщения `prepare(N,V)` каждому из акцепоров.
  - Фаза 1: акцептор при первом приёме `prepare` отвечает сообщением `promise(N,V)` пропоузеру. Акцептор обещает не принимать предложения с номерами, меньшими `N`
  - Фаза 2: пропоузер дожидается `prepare` и если достигнут кворум, принимает решение и отправляет `accept(N,V)`, где `V` — значение из сообщения с наибольшим номером.

Протокол Paxos



Алгоритм Paxos

- Возможна ситуация, когда кворума достичь попеременно не удаётся.
- `live-lock`

Консенсус в системах со сбоями

- Алгоритмы класса `wait-free` — устойчивые к сбоям алгоритмы.
- Если один поток преждевременно завершился или остановился, алгоритм завершается за конечное число шагов.

Теорема FLP impossibility

- Фишер-Линч-Патерсон
- **Теорема:** для асинхронной системы N потоков с хотя бы одним сбойным потоком нельзя построить решение задачи консенсуса.

Идеи доказательства теоремы FLP

- Вводится понятие *конфигурации* как множества состояний потоков и буферов.
- Определяется понятие *корректности* потока, как потока, считывающего все предназначенные ему сообщения.
- Определяется понятие *частичной корректности* протокола.
- Определяется понятие *последовательности конфигураций*.
- Определяется понятие *решающего состояния протокола*.
- Показывается, что начальное состояние бивалентно.
- Показывается, что существует такая последовательность исполнения, которая каждый раз будет приводить систему в бивалентное состояние.

Спасибо за внимание.

Следующая тема — подходы к синхронизации, задача читателей-писателей, замки.