Распределенные системы

Содержание

1 Формализм. Логические часы Лампорта (свойства и алгоритм)

Кратко опишем используемые далее обозначения.

Обозначение	Объект
$P, Q, R, \ldots \in \mathbb{P}$	Процессы
$a, b, c, \ldots \in \mathbb{E}$	События в процессах `proc(e)` $\in \mathbb{P}$
$m \in \mathbb{M}$	Сообщения, `snd(m)`, `rcv(m)` $\in \mathbb{E}$.

Таблица 1: Общие обозначения

Определение. Отношение *Произошло-до* $(\rightarrow)$ – минимальный строгий частичный порядок на $\mathbb{E} \times \mathbb{E}$ такой, что

- $e \rightarrow f$, если e, f в одном процессе и e идет перед f.
- Если m – сообщение, то `snd(m)` $\rightarrow$ `rcv(m)`.

Определение. *Логические часы*. Определим функцию $C : \mathbb{E} \rightarrow N$ так, чтобы

$$\forall e, f \in \mathbb{E}\ e \rightarrow f \Longrightarrow C(e) < C(f).$$

Алгоритм. (Логические часы Лампорта)

- Каждый процесс хранит счетчик.
- Перед посылкой процесс увеличивает счетчик на единицу.
- При посылке дополнительно посылается счетчик.
- Получатель обновляет свое время следующим образом:

$$C \leftarrow \max(C, C_r) + 1.$$

Свойства логических часов Лампорта:

- Время события не уникально.
- Являются логическими часами в смысле определения.

2 Формализм. Векторные часы (свойства и алгоритм)

Определение. *Векторные часы*. Определим функцию $VC : \mathbb{E} \to N^k$ так, чтобы

$$\forall e, f \in \mathbb{E}\ e \to f \iff VC(e) < VC(f).$$

Сравнение производится покомпонентно.

Алгоритм. (Векторное время)

- Каждый процесс хранит свой вектор-время (размер – число процессов).
- Перед посылкой сообщения процесс увеличивает свою компоненту на единицу.
- При приеме сообщение берется покомпонентный максимум:

$$VC \leftarrow \max(VC, VC_r).$$

Свойства векторного времени:

- Векторное время уникально для каждого события.
- Векторное время полностью передает отношение произошло-до.
-

$$\forall e, f \in \mathbb{E}\colon \texttt{proc(e)} = P_i,\ \texttt{proc(f)} = P_j \Longrightarrow \left(e \to f \iff \begin{pmatrix} VC(e)_i \\ VC(e)_j \end{pmatrix} < \begin{pmatrix} VC(f)_i \\ VC(f)_j \end{pmatrix}\right).$$

3 Формализм. Часы с прямой зависимостью (свойства и алгоритм)

Определение.

$$e \to_d f \iff e < f \vee \exists m \in \mathbb{M}\colon\ e \leqslant \texttt{snd(m)} \wedge \texttt{rcv(m)} \leqslant f.$$

Определение. *Часы с прямой зависимостью*. Определим функцию $VC_d\colon \mathbb{E} \to N^k$ так, чтобы

$$\forall e, f \in \mathbb{E}\colon\ e \to_d f \iff VC_d(e) < VC_d(f).$$

Алгоритм. (Часы с прямой зависимостью)
Алгоритм полностью повторяет алгоритм для векторных часов, за исключением того, что посылается только та компонента времени, которая соотвествует процессу-отправителю.

4 Взаимное исключение в распределенной системе. Централизованный алгоритм.

Обозначение	Объект
CS_i	Критическая секция с номером
$\texttt{Enter}(CS_i)$	Вход в критическую секцию
$\texttt{Exit}(CS_i)$	Выход из критической секции

Таблица 2: Общие обозначения

Определение. *Взаимное исключение*. Основное требование

$$\texttt{Exit}(CS_i) \rightarrow \texttt{Enter}(CS_{i+1}).$$

Определение. *Требование прогресса:*

- Каждое желание процесса попасть в критическую секцию будет рано или поздно удовлетворено.
- Может быть гарантирован тот или иной уровень честности удовлетворения желания процессов о входе в критическую секцию.

Алгоритм. (Централизованный алгоритм)

- Весь процесс контролируется выделенным координатором.
- Общение происходит по следующему протоколу:

Вид запроса	Действие
`request`	Запрос разрешения у координатора
`ok`	Одобрение координатором входа в секцию
`release`	Освобождение пользователем критической секции

Таблица 3: Виды запросов

- При входе в критическую секцию узел шлёт запрос координатору, дожидается разрешения, затем входит в критическую секцию. При завершении работы узел посылает координатору сообщения, что секция свободна. Данный алгоритм всегда требует 3 сообщения для работы с критической секцией.
- Не масштабируется из-за необходимости иметь выделенного координатора.

5 Взаимное исключение в распределённой системе. Алгоритм Лампорта

Вид запроса	Действие
`request`	От запрашивающего ко всем другим узлам
`ok`	Подтверждение получения (не даёт права входа в CS)
`release`	Освобождение узлом критической секции (всем узлам)

Таблица 4: Виды запросов алгоритма Лампорта

Алгоритм. (Алгоритм Лампорта)

- Координатор отсутствует, все узлы равны.
- Сообщения request и release рассылаются всем другим узлам, всего $3n - 3$ сообщения на CS.
- Используются логические часы лампорта. Для установления порядка "кто раньше". Обязательно требуется порядок FIFO на сообщениях.
- Все узлы хранят у себя очередь запросов.
- В критическую секцию можно войти, если
  - Мой запрос первый в очереди, т.е. его время меньше времени остальных запросов (при равенстве времен порядок определяется по номеру узла, который посылается вместе с часами).
  - Получен ok от всех других узлов, т.е. они знают о вашем запросе.
- Если узел хочет войти в CS, то он посылает всем другим узлам request со своими часами и id. Ждёт от всех ok. Если других запросов не поступало, либо время нашего запроса меньше времени других запросов, то входим в критическую секцию. Иначе ждем release от всех узлов, которые раньше нас в очереди.

6 Взаимное исключение в распределённой системе. Алгоритм Рикарда и Агравалы

Вид запроса	Действие
`request`	От запрашивающего ко всем другим узлам
`ok`	После выхода из критической секции

Таблица 5: Виды запросов алгоритма Рикарда и Агравалы

Алгоритм. (Алгоритм Рикарда и Агравалы)

- Оптимизация алгоритма Лампорта.
- Всего $2n - 2$ сообщений.
- Если узел хочет войти в CS, то он шлет request всем узлам. Если узел получивший запрос не хочет войти в CS, либо его номерок запроса (в часах) больше, то он отсылает разрешение ok. Узел, который входит в CS, хранит в очереди какие ok-ответы он должен послать после выхода.

7 Взаимное исключение в распределённой системе. Алгоритм обедающих философов.

Определение. В частном случае ресурсы – вилки, процессы – философы, граф конфликтов – кольцо.

Теорема 7.1. В ориентированном графе без циклов всегда есть исток.

Теорема 7.2. Если у истока перевернуть все ребра, то граф останется ациклическим.

Алгоритм. (Алгоритм обедающих философов)

- Философ владеет вилкой, если ребро в графе конфликтов исходит из его вершины.
- Философ может принять пищу, если владеет обеими вилками, т.е. он исток.
- После еды вилки надо отдать (ленивый способ):
  - После еды вилки помечаются грязными.
  - Моем вилки и отдаём их по запросу, даже если сами хотим есть.
  - Чистые вилки не отдаём, если сами хотим есть. Ожидаем все вилки, едим, отдаем, если был запрос.

Алгоритм. (Обобщение алгоритма обедающих философов на произвольный граф)

- Взаимное исключение эквивалентно полному графу конфликтов (ребро между каждой парой процессов).
- При инициализации вилки раздаются в каком-то порядке (например, по порядку id процессов).

Замечание. (Результат)

- 0 сообщений на повторный заход в критическую секцию.
- В худшем случае $2n - 2$ сообщения.
- Количество сообщений пропорционально числу желающих попасть в критическую секцию.

8 Алгоритм на основе токена.

Определение. Токен – некоторый объект, который даёт владельцу право на вход в критическую секцию.

Алгоритм. (Алгоритм на основе токена)

- В система существует один токен для конкретного ресурса (критической секции).
- Все узлы в системе объединены в кольцо.
- Токен пересылается по кругу, и каждый процесс делает следующее:
  - Если нет желания войти в критическую секцию, то пересылаем токен дальше.
  - Если желание есть, то входим (т.к. у нас уникальное право). После завершения передаем токен дальше.

Замечание. Количество сообщений в системе стабильно, но необходимо ждать, пока токен дойдет до тебя.

9 Взаимное исключение в распределённой системе. Алгоритмы на основе кворума (простое большинство, рушащиеся стены).

Определение. *Кворум:*

- Семейство подмножеств множества процессов $Q \subset 2^{\mathbb{P}}$.
- Любые два кворума имеют непустое пересечение:

$$\forall A, B \in Q: \; A \cap B \neq \emptyset$$

Примеры. Виды кворумов:

- Централизованный алгоритм как частный случай кворума.
- Простое большинство (больше половины процессов) и взвешенное большинство.
- Рушащиеся стены.

Определение. *Кворум «рушащиеся стены»*

- Процессы образуют квадратную матрицу (приблизительно).
- Кворумом назовем набор процессов, состоящий из некоторого столбца целиком и представителей всех остальных столбцов.
- Заметим, что пересечение любым двух таких множеств непусто, что удовлетворяет определению кворума.

Замечание. Не все кворумы тривиальны и плохо мастурбируются. Например, “рушащиеся стены” имею размер порядка $2\sqrt{n}$.

Замечание. При пересечении кворумов потенциально возможен deadlock. Решением служит *иерархическая блокировка*.

10 Согласованное глобальное состояние (согласованный срез). Алгоритм Чанди-Лампорта. Запоминание сообщений на стороне отправителя.

Определение. *Срезом* называется любое $G \subseteq E$, удовлетворяющее условию

$$\forall e \in E, f \in G \; e < f \Longrightarrow e \in G.$$

Определение. Срез G называется *согласованным*, если

$$\forall e \in E, f \in G \; e \to f \Longrightarrow e \in G.$$

Алгоритм. (Чанди, Лампорт)

- Снача все процессы помечаются как белые (w).
- Процесс-инициатор запоминает свое состояние, помечается красным (r) и посылает токен всем соседям.
- При получении сообщения w-процесс запоминает свое состояние и становится красным, после чего посылает токен всем соседям.
- Запомненные состояния образуют согласованный срез.

Замечание. Алгоритм работает корректно только в случае, когда соблюдается FIFO порядок на сообщениях.

Замечание. (Классификация сообщений)
Сообщения делятся на 4 вида:

- ww-сообщения. Их не надо сохранять, состояние их уже учитывает.
- rr-сообщения. Их не надо сохранять, они просто сами произойдут потом.
- wr-сообщения. Такие сообщения нужно обязательно сохранять для дальнейшего восстановления состояния системы.
- rw-сообщения. Таких не может быть по определению согласованного среза.

Алгоритм. (Запоминание сообщений на стороне отправителя)

- w-процесс обязательно отправляет токен-подтверждение на каждое полученное сообщение.
- Процесс-отправитель сохраняет только те сообщения, на которые не успело прийти подтверждение.
- r-процесс не отправляет токен-подтверждение, поэтому wr-сообщения и только они не удалятся из буфера.
- Буфер готов тогда, когда процесс становится красным. После этого он не может участвовать в wr-сообщениях.

11 Согласованное глобальное состояние (согласованный срез). Алгоритм Чанди-Лампорта. Запоминание сообщений на стороне получателя.

Первую часть вопроса см. в предыдущем билете.

Алгоритм. (Запоминание сообщений на стороне получателя)
Процесс P запоминает все сообщения от процесса Q, пришедшие в отрезок времени после того, как P стал красным, до того, как Q пришлет маркер из алгоритма Чанди-Лампорта.

12 Глобальные свойства. Стабильные и нестабильные предикаты. Слабый конъюнктивный предикат. Централизованный алгоритм.

Определение. *Глобальным предикатом* называется предикат, определенный над состоянием системы в целом. Под состоянием системы подразумевается согласованный срез.

Определение. Предикат $P(G)$ называется стабильным, если для любых согласованных срезов G, H выполняется:

$$G \subset H \wedge P(G) \Longrightarrow P(H).$$

Алгоритм. (Простой алгоритм для стабильных предикатов)
Строим согласованный срез при помощи алгоритма Чанди-Лампорта, проверяем на нём выполненность предиката. Если он верен, то будет верен и в дальнейшем.

Определение. *Локальным* называется предикат, зависящий от состояния только одного процесса.

Замечание. Если глобальный предикат является дизъюнкцией локальных, то его предельно просто проверять даже без построения каких-либо срезов.

Определение. Предикат называется *слабым конъюнктивным*, если он верен тогда и только тогда, когда он верен на хотя бы одном согласованном срезе.

Теорема 12.1. Срез согласован тогда и только тогда, когда векторные времена процессов на этом срезе попарно несравнимы.

Алгоритм. (Централизованный алгоритм для слабого конъюнктивного предиката)

- Каждый процесс отслеживает свое векторное время VC.
- При наступлении истинности локального предиката, отправляем сообщение координатору C (делая при этом все необходимые манипуляции со временем).
- Координатор поддерживает *срез-кандидат* и очередь необработанных сообщений.
  - Для каждой компоненты среза-кандидата координатор хранит флажок. Красный – элемент не может быть частью согласованного среза. Зеленый – может. Начальное состояние – нулевой вектор, все флажки красные.
  - Обрабатываем сообщения только от красных процессов; от зеленых сообщения идут в очередь.

- Сравниваем пришедший вектор попарно с другими процессами (достаточно сравнить только две соответствующие компоненты). Если нарушилась согласованность (новый вектор оказался больше), то делаем меньший процесс красным. После обработки делаем процесс зеленым.
- Как только все флажки стали зелеными, найден согласованный срез.

Теорема 12.2. (Корректность)

- Алгоритм никогда не пропустит согласованный срез. Действительно, пусть есть согласованный срез. В каком-то порядке процессы дойдут то момента истинности предиката, после чего пошлют сообщения координатору. Ни одно из этих сообщений не может сделать другой процесс красным (если он стал зеленым после обработки сообщения их этого среза), потому что срез согласован. Поэтому все процессы станут зелеными сразу после обработки соответствующих сообщений.
- Компонента согласованного среза становится зеленой и всгеда будет такой оставаться.

13 Слабый конъюнктивный предикат. Распределенный алгоритм.

Первую часть вопроса см. в предыдущем билете.

Алгоритм. (Распределенный алгоритм для слабого конъюнктивного предиката)

- Каждый процесс имеет своего собственного координатора.
- Процессы шлют сообщения своим координаторам. Координаторы общаются между собой, пересылая друг другу срезы-кандидаты с флажками.
- Красные координаторы обрабатывают сообщения от своих процессов. После обработки, координатор становится зеленым. Если другой процесс был помечем красным, то соответствующее сообщение шлется нужному координатору.

14 Диффундирующие вычисления, пример. Останов. Алгоритм Дейксты и Шолтена.

Определение. *Диффундирующим* называется вычисление, для которого верно:

- Процессы бывают в двух состояниях: активный и пассивный.
- Получение сообщения делает процесс активным.
- Посылать сообщения могут только активные процессы.
- Активный процесс в любой момент может стать пассивным.
- Алгоритм начинается с одного активного процесса-инициатора.

Пример. Алгоритм Дейкстры – пример диффундирующего вычисления.

Определение. Диффундирующее вычисление завершилось, если все процессы пассивны и нет сообщений в пути.

Определение. *Проблема останова* – как процессу-инициатору узнать, когда алгоритм завершился?

Алгоритм. (Дейкстра, Шолтен. Останов диффундирующего вычисления)

- Все процессы будут выстраиваться в дерево.
- На все сообщения требуются подтверждения.
- Каждый процесс знает своего предка в дереве, число своих детей и разницу между числом отправленных сообщений, и сообщений, на которые было получено подтверждение.
- *Зеленым* назовем пассивный процесс без детей и неподтвержденных сообщений. В противном случае, процесс считается красным. Дерево состоит из красных процессов.
- При получении сообщения, зеленый процесс становится красным, делая родителем отправителя сообщения и высылая тому подтверждение. После получения подтверждения отправитель увеличивает счетчик детей.
- Аналогично, как только процесс становится зеленым, он удаляет себя из дерева, посылая предку соответствующее сообщение.
- Вычисление остановилось, как только корень дерева (то есть, инициатор), становится зеленым.

15 Локально-стабильные предикаты: согласованные интервалы, барьерная синхронизация (3 алгоритма). Применение для определения взаимной блокировки (deadlock).

Определение. Пара срезов $F, G \subseteq E$ называется *интервалом* $[F, G]$, если $F \subseteq G$.

Определение. Интервал $[F, G]$ называется согласованным, если

$$\forall e \in E, f \in F \; e \to f \Longrightarrow e \in G.$$

Замечание. Интервал $[G, G]$ согласован тогда и только тогда, когда G – согласованный срез.

Теорема 15.1. Интервал $[F, G]$ согласован тогда и только тогда, когда существует согласванный срез H такой, что $F \subseteq H \subseteq G$.

Определение. Интервал $[F, G]$ называется *барьерно-синхронизированным*, если

$$\forall f \in F, g \in E \setminus G \; f \to g.$$

Теорема 15.2. Любой барьерно-синхронизированный интервал согласован.

Алгоритм. (Алгоритмы построения барьерной синхронизации)

- Построение через координатора. Каждый процесс посылает координатору сообщение. Когда координатор получил сообщение от *всех*, он посылает всем сообщение. Срезы для интервала: по посылке сообщений процессами и по приему сообщений от координатора.
- Посылка каждый каждому.
- Посылка токена два раза по кругу.

Определение. *Локально-стабильным* называется стабильный предикат, определяемый группой процессов с неизменным состоянием.

Пример. Взаимная блокировка – пример локально-стабильного предиката. Для проверки такого предиката необходим согласованный срез. Для этого воспользуемся барьерной синхронизацией $[F, G]$. Запомним состояние системы на срезе F (например, это может сделать координатор, если он используется). После этого каждый процесс будет помнить, менялось ли у него состояние (относительно блокировки). Если на момент G состояние у процессов не менялось и на момент F была зафиксирована взаимная блокировка, то она в действительности есть.

16 Общий порядок (total order). Алгоритм Лампорта.

Определение. Пусть в системе сообщения рассылаются нескольким получателям. Обозначим $\texttt{rcv}_p(m)$ – события получения сообщения процессами $p \in \mathbb{P}$. Будем говорить, что соблюдается *общий порядок*, если

$$\nexists m, n \in \mathbb{M},\ p, q \in \mathbb{P}\colon\ \texttt{rcv}_p(m) < \texttt{rcv}_p(n) \wedge \texttt{rcv}_q(n) < \texttt{rcv}_q(m).$$

Замечание. Для случая, когда сообщения отправляются только одному процессу, это свойство всегда выполняется.

Алгоритм. (Централизованный алгоритм обеспечения общего порядка)
Пусть в системе соблюдается FIFO порядок сообщений. Тогда если процесс P хочет сделать рассылку сообщения, он сообщает об этом координатору, который в свою очередь рассылает сообщения в фиксированном порядке.

Замечание. Централизованный алгоритм также обеспечивает причинно-согласованный порядок.

Алгоритм. (Лампорт)
Обобщим алгоритм взаимной блокировки. Пусть в системе соблюдается FIFO порядок сообщений. Все multicast-сообщения придется заменить на broadcast. Процесс, собирающийся послать сообщения, берет “билет”, соответствующий его логическому времени, и посылает `request` запрос всем другим процессам. Те, в свою очередь, отвечают ему `ok`. После того, как был получен `ok` от всех процессов, отправитель начинает рассылку. Как и в алгоритме Лампорта для взаимного исключения, порядок обработки сообщений определяется парой из билета и номера процесса.

17 Общий порядок (total order). Алгоритм Скина.

Первую часть вопроса см. в предыдущем билете.

Алгоритм. (Скин)
Модифицируем алгоритм Лампорта. Для этого алгоритма не требуется FIFO порядок на сообщениях, и он умеет делать multicast-сообщения.

- Пусть процесс P хочет сделать рассылку. В таком случае, он шлет всем тем, кому надо, `request`, приписывая к нему свое логическое время в качестве *предварительного* билета.
- Все процессы, как и в алгоритме Лампорта, имеют очередь сообщений, приоритетную по билетам.
- Если обрабатываемое сообщение – запрос на рассылку, то автору отправляется `ok` (аналогично алгоритму Лампорта).
- Как только отправитель получает все `ok`, он отправляет настоящие сообщения, приписывая к ним текущее логическое время в качестве *финального* билета. Эти сообщения обрабатываются другими процессами в общем порядке, приоритетном по номеру билета.
- Сообщения из рассылки обрабатываются процессами как только доходят до вершины очереди.