1 Предварительные сведения

С точки зрения парадигмы обмена сообщениями распределенная система состоит из *конечной сети* (или *графа*) с *N процессами* (или *узлами*). Каждый процесс в сети имеет уникальный *идентификатор*. Процессы в сети соединены *каналами (ребрами)*, посредством которых они могут пересылать друг другу сообщения. От одного процесса к другому идет не более одного канала, процесс не имеет канала для связи с самим собой. Иногда процессу может понадобиться передать сообщение самому себе, но очевидно, что для этого канал не требуется. Будем обозначать буквами *E* и *D* количество каналов и радиус сети соответственно.

Процесс может записывать собственное состояние и сообщения, которые он отправляет или принимает. Процессы не имеют общей памяти или глобальных часов. Каждый процесс знает только своих непосредственных соседей по сети. Предполагается, что топология сети сильно связная, то есть существует путь от каждого из процессов ко всем остальным.

Коммуникация в сети осуществляется *асинхронно*. Это значит, что получение и отправка сообщения — отдельные события в процессе получения и передачи соответственно. Задержка сообщения в канале произвольна, но конечна. Мы предполагаем, что для предотвращения возникновения дублирующихся, поврежденных или потерянных сообщений используются коммуникационные протоколы. Каналы не обязаны обрабатывать сообщения в порядке поступления, то есть сообщения могут опережать друг друга.

В *направленной* сети сообщения могут передаваться по каналу только в одном направлении, в то время как в *ненаправленной* сети они могут передаваться в обоих направлениях. Ненаправленные каналы необходимы для распределенных алгоритмов с подтверждением. Ациклические сети всегда будут ненаправленными, поскольку иначе сеть не будет сильно связной. Топология сети называется *полной*, если между каждыми двумя процессами существует ненаправленный канал.

Остовное дерево ненаправленной сети — связный ациклический граф, который состоит из всех узлов и подмножества ребер сети. Ребра остовного дерева называются *ребрами дерева*, а все остальные ребра — *хордами*. Если у ребер остовного дерева задано направление (что бывает довольно часто), то

такое дерево называется *деревом входов*, в котором все пути ведут к одному узлу, называемому *корнем*. Всякое направленное ребро ведет от *потомка* к *предку*.

В некоторых местах мы будем отклоняться от упомянутых ранее предположений и переходить к рассмотрению, например, синхронной коммуникации, при которой отправка и получение одного сообщения формируют одно атомарное событие, или ненадежных процессов и каналов, или каналов с очерёдной обработкой, или процессов, специфицированных вероятностным, а не детерминированным образом. В таком случае об этом будет говориться явно.

1.1. Системы переходов

Распределенный алгоритм, работающий в распределенной системе, предоставляет (обычно детерминированные) спецификации отдельных процессов. Глобальное состояние распределенного алгоритма, называемое *конфигурацией*, развивается посредством *переходов*. Общее поведение распределенной системы описывается *системой переходов*, которая состоит из:

- множества конфигураций C;
- бинарного отношения перехода $\rightarrow$, определенного на множестве C;
- множества *начальных* конфигураций $I \subseteq C$.

Конфигурация γ называется *терминальной*, если она не имеет исходящих переходов: не существует $\delta \in C$ таких, что $\gamma \rightarrow \delta$. *Выполнением* распределенной системы называется последовательность конфигураций $\gamma_0, \gamma_1, \gamma_2...$, которая либо бесконечна, либо заканчивается такой терминальной конфигурацией γ_k, что:

- $\gamma_0 \in I$ и
- $\gamma_i \rightarrow \gamma_{i+1}$ для любых $i \geq 0$ (и в случае конечного выполнения $i < k$).

Конфигурация δ называется *достижимой*, если существует $\gamma_0 \in I$ и последовательность $\gamma_0, \gamma_1, ..., \gamma_k$ с $\gamma_i \rightarrow \gamma_{i+1}$ для всех $0 \leq i < k$ и $\gamma_k = \delta$.

1.2. Состояния и события

Конфигурация распределенной системы состоит из локальных *состояний* ее процессов и сообщений, находящихся в процессе передачи. Переход между конфигурациями распределенной системы связан с *событием* (в случае синхронной коммуникации — с двумя) в одном или двух ее процессах. Процесс может обрабатывать события *internal*, *send* и *receive* (внутренние события, события отправки и события получения соответственно).

Внутреннее событие влияет только на состояние процесса, в котором оно обрабатывается. Типичные внутренние события: запись или чтение переменной. Присвоение нового значения переменной записывается в виде $\leftarrow$; например, $n \leftarrow n + 1$ означает, что значение переменной n, представляющей собой

натуральное число, увеличивается на единицу. Событие отправки в общем случае приводит к возникновению соответствующего события получения для того же сообщения в другом процессе. Предполагается, что два различных события никогда не происходят в один и тот же момент реального времени (за исключением синхронной коммуникации).

Процесс называется *инициатором*, если его первое событие является внутренним событием или событием отправки, то есть инициатор может начинать обрабатывать события без получения входных данных от другого процесса. Алгоритм называется *централизованным*, если в нем ровно один инициатор. В *децентрализованном* алгоритме может быть несколько инициаторов.

1.3. Утверждения

Утверждение представляет собой предикат относительно конфигураций алгоритма. А именно в каждой конфигурации утверждение будет либо истинным, либо ложным.

Утверждение называется *свойством безопасности*, если оно истинно в каждой достижимой конфигурации алгоритма. Свойство безопасности обычно выражает то, что нечто плохое никогда не произойдет. Примеры свойств безопасности:

- можете всегда на меня рассчитывать;
- стоимость жизни никогда не снижается;
- если происходит прерывание, сообщение будет выведено в течение одной секунды.

Задача определения достижимости в общем случае неразрешима. Утверждение P *инвариантно*, если $P(\gamma)$ для любого $\gamma \in I$; $\gamma \to \delta$ и $P(\gamma)$, то $P(\delta)$.

Иными словами, инвариант истинен во всех начальных конфигурациях и сохраняет значение при всех переходах. Очевидно, что всякий инвариант является свойством безопасности. Обратите внимание, что проверка инвариантности утверждения не подразумевает достижимости.

Утверждение называется *свойством живучести*, если выполнения, начиная с определенного момента, содержат конфигурацию, в которой утверждение истинно. Свойство живучести обычно выражает тот факт, что нечто хорошее со временем произойдет. Примеры свойств живучести:

- то, что повысилось, должно понизиться;
- в конце концов, программа всегда завершается;
- если произошло прерывание, будет выведено сообщение.

Свойство живучести иногда выполняется только в отношении так называемых *справедливых* выполнений алгоритма. Рассмотрим, к примеру, простой алгоритм подбрасывания монеты до тех пор, пока не выпадет «решка». По-

скольку существует шанс бесконечного выполнения, в котором каждым исходом будет «орел», то свойство живучести, подразумевающее возможность выпадения «решки», в таком случае не выполняется. Однако, если монета «справедливая», такое бесконечное выполнение имеет нулевую вероятность появления; мы бесконечно подбрасываем монету с вероятным исходом «решка», но исход «решка» никогда не наступает. Выполнение называется справедливым, если каждое событие, которое может произойти в бесконечном количестве конфигураций выполнения, происходит бесконечно часто во время выполнения. Бесконечное выполнение, в котором каждым исходом подбрасывания монетки будет «орел», не является справедливым. Заметьте, что всякое конечное выполнение будет справедливым по умолчанию.

1.4. Отношение каузального порядка

В каждой конфигурации распределенной асинхронной системы события, происходящие в различных процессах, независимы. Это значит, что они могут происходить в любом порядке. Отношение *каузального порядка* $\prec$ — такое бинарное отношение на событиях в выполнении, что $a \prec b$ тогда и только тогда, когда a должно происходить перед b. Иными словами, события в выполнении не могут быть переупорядочены таким образом, что a происходит перед b. Каузальный порядок выполнения — минимальное отношение, удовлетворяющее следующим условиям:

- если a и b — события одного и того же процесса и a происходит перед b, то $a \prec b$;
- если a — событие отправки и b — соответствующее ему событие получения, то $a \prec b$;
- если $a \prec b$ и $b \prec c$, то $a \prec c$.

Будем использовать запись $a \preceq b$, если $a \prec b$ или $a = b$. Разные события в выполнении, не связанные отношением каузального порядка, называются *параллельными*. При проектировании распределенных систем главное — совладать с параллелизмом (например, для предотвращения эффекта гонок).

Перестановка параллельных событий в выполнении не влияет на его результат. Все вместе эти перестановки образуют *вычисление*. Все выполнения в вычислении начинаются в одной конфигурации и, если они конечны, завершаются в одной и той же конечной конфигурации. В общем случае мы будем рассматривать вычисления, а не выполнения.

1.5. Логические часы

Сложно поддерживать общие для всех отдельных процессов распределенной системы физические часы, аппроксимирующие глобальное реальное время. Для многих приложений, однако, нас интересуют не точные моменты во времени,

в которые происходят события, а только порядок их возникновения во времени. *Логические часы C* отображают порядок событий в вычислении на такое частично упорядоченное множество, что:

$$a \prec b \Rightarrow C(a) < C(b).$$

Часы Лэмпорта LC ставят в соответствие всякому событию a длину k самой длинной каузальной цепи $a_1 \prec \ldots \prec a_k = a$ в вычислении. Очевидно, что из $a \prec b$ следует $LC(a) < LC(b)$, поэтому часы Лэмпорта являются логическими часами. Значения времени, которые часы Лэмпорта ставят в соответствие событиям, могут быть вычислены в процессе исполнения следующим образом. Рассмотрим событие a, и пусть k — это значение времени предыдущего события в том же процессе ($k = 0$, если такого события не существует).

- Если a — внутреннее событие или событие отправки, то $LC(a) = k + 1$.
- Если a — событие получения и b — событие отправки, соответствующее a, то $LC(a) = \max\{k, LC(b)\} + 1$.

Часы Лэмпорта могут упорядочивать параллельные события; то есть может случиться так, что $LC(a) < LC(b)$ для параллельных событий a и b. Иногда удобно использовать логические часы, для которых это никогда не выполняется. *Векторные часы VC* имеют следующее свойство:

$$a \prec b \Leftrightarrow VC(a) < VC(b).$$

Пусть сеть состоит из процессов $p_0 \ldots p_{N-1}$. Векторные часы ставят в соответствие событиям в вычислении значения из N^N, посредством чего это множество снабжается отношением частичного порядка, определенного следующим образом:

$$(k_0 \ldots k_{N-1}) \leq (\ell_0 \ldots \ell_{N-1}) \Leftrightarrow k_i \leq \ell_i \text{ для любого } i = 0 \ldots N-1.$$

(В отличие от лексикографического порядка на N^N, этот порядок является частичным.) Векторные часы определены следующим образом: $VC(a) = (k_0 \ldots k_{N-1})$, где k_i — длина самой длинной каузальной цепи $a_1^i \prec \ldots \prec a_{k_i}^i$ событий процесса p_i с $a_{k_i}^i \preceq a$.

Очевидно, что из $a \prec b$ следует $VC(a) < VC(b)$; что, в свою очередь, следует из того, что $c \preceq a$ означает, что $c \prec b$ для каждого события c. Обратно — из $VC(a) < VC(b)$ из $a \prec b$.

Для конкретики рассмотрим самую длинную каузальную цепь $a_1^i \prec \ldots \prec a_k^i = a$ событий в процессе p_i, где происходит a. Тогда $VC(a) < VC(b)$ подразумевает то, что i-й коэффициент $VC(b)$ по меньшей мере равен k и поэтому $a \preceq b$. Поскольку очевидно, что a и b должны быть различными, то $a \prec b$. Значение времени по векторным часам также может быть рассчитано в процессе исполнения (см. упражнение 1.7).

1.6. Базовые и управляющие алгоритмы

В нескольких главах будут рассматриваться распределенные алгоритмы, предоставляющие некоторые сервисы или обнаруживающие некоторое свойство во время выполнения распределенного алгоритма. Например, в главе 2 показано, как процессы могут делать снимок конфигурации идущего вычисления, глава 5 рассматривает то, как процессы могут обнаруживать завершение, а глава 6 обсуждает сборку мусора для освобождения памяти, занятой недоступными объектами. Основной распределенный алгоритм, для которого выполняется снимок состояния, обнаружение завершения или сборка мусора, называется *базовым* алгоритмом, а распределенный алгоритм, надстраиваемый для выполнения конкретной задачи, называется *управляющим* алгоритмом.

Библиографические замечания

Часы Лэмпорта впервые описаны в [45]. Векторные часы были независимо предложены в [29] и [55]. Существуют специализированные парадигмы, поддерживающие распределенные вычисления на графах [53, 41].

Упражнения

Упражнение 1.1. Что из указанного является более общим?

1. Алгоритм для направленных или ненаправленных сетей.
2. Управляющий алгоритм для централизованных или децентрализованных базовых алгоритмов.

Упражнение 1.2 [76]. Приведите такой пример системы переходов S и такое утверждение P, чтобы P являлось свойством безопасности S, но не являлось его инвариантом.

Упражнение 1.3 [76]. Определите объединение $S_1 = (C, \rightarrow_1, I)$ и $S_2 = (C, \rightarrow_2, I)$ как $S = (C, \rightarrow, I)$, где $\rightarrow = (\rightarrow_1 \cup \rightarrow_2)$. Докажите, что если P — инвариант S_1 и S_2, то P — инвариант S.

Упражнение 1.4. Рассмотрим следующее выполнение алгоритма, включающего процессы p_0, p_1, p_2 и p_3; события даны в том порядке, в котором они выполняются в реальном времени:

- p_0 отправляет сообщение p_2;
- p_3 отправляет сообщение p_2;
- p_0 отправляет сообщение p_1;
- p_2 принимает сообщение от p_0;
- p_1 принимает сообщение от p_0;

- p_2 выполняет внутреннее событие;
- p_2 отправляет сообщение p_3;
- p_3 принимает сообщение от p_2;
- p_2 принимает сообщение от p_3;
- p_3 выполняет внутреннее событие;
- p_1 отправляет сообщение p_0;
- p_3 отправляет сообщение p_2;
- p_0 принимает сообщение от p_1;
- p_2 принимает сообщение от p_3.

Используя логические часы Лэмпорта, поставьте в соответствие данным событиям значения времени. Проделайте то же самое, используя векторные часы.

Упражнение 1.5 [76]. Определите отношение каузального порядка на переходах в системе с синхронной коммуникацией. Адаптируйте логические часы Лэмпорта для таких систем и предложите распределенный алгоритм для вычисления значения времени в процессе исполнения.

Упражнение 1.6. Приведите пример, в котором $LC(a) < LC(b)$, но при этом a и b — параллельные события.

Упражнение 1.7. Предложите алгоритм вычисления значения времени по векторным часам в процессе исполнения.