

Распределенные алгоритмы и системы

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

Блок 5

Иллюстрация трудности разработки
распределённых алгоритмов

Лектор:

Подымов Владислав Васильевич

E-mail:

valdus@yandex.ru

Постановка задачи

Рассмотрим такую **задачу**: требуется передать данные d от узла A к узлу B по каналу связи C

Положим, что сообщения, пересылаемые через C , имеют следующую структуру:

- ▶ (\mathbf{t}, x) : тип \mathbf{t} и данные x — или
- ▶ \mathbf{t} : тип \mathbf{t} без данных

Будем считать, что канал C всегда готов к пересылке сообщений, но при этом **ненадёжен**: сообщение может

- ▶ доставляться сколь угодно долго или
- ▶ потеряться в сети без доставки (но при многократной отправке рано или поздно будет доставлено)

Постановка задачи

Положим, что до начала и после завершения выполнения своих частей протокола A и B не хранят никакую информацию о передаче сообщения и находятся в особом **закрытом состоянии** (\square), и что каждый узел может **выйти из строя** с последующим перезапуском в \square

Для обозначения результатов в конце передачи будем использовать следующие команды:

- ▶ $done_A(x)$ в узле A для обозначения уверенности узла в том, что данные x доставлены
- ▶ $done_B(x)$ в узле B для обозначения того, что данные x приняты

Будем говорить, что по завершении пересылки данные

- ▶ **потеряны**, если A уверен, что эти данные доставлены, но B ни разу их не принимает
- ▶ **дублированы**, если B более одного раза принимает эти данные

Пересылку данных будем считать **надёжной**, если передача данных завершается без их потери и дублирования

Полноценное решение задачи невозможно

Гарантированно надёжная пересылка данных в условиях выхода узлов из строя невозможна

Чтобы в этом убедиться, достаточно представить себе два сценария пересылки, начинающихся с таких фрагментов:

1. A отправляет данные; B принимает сообщение с данными и выходит из строя непосредственно **перед** выполнением $done_B$
2. A отправляет данные; B принимает сообщение с данными и выходит из строя непосредственно **после** выполнения $done_B$

По условиям пересылки, выполнение узлов A и B в этих двух сценариях одинаково с точностью до обозначенного выполнения $done_B$

Следовательно:

- ▶ Если пересылка по сценарию 1 надёжна, то в сценарии 2 данные дублируются
- ▶ Если пересылка по сценарию 2 надёжна, то в сценарии 1 данные теряются

Передача данных с одним сообщением

Далее полагаем, что узлы A и B не выходят из строя (но канал C по-прежнему считается ненадёжным)

В связи с надёжностью узлов далее команду `done` будем записывать в конце и считать завершающей передачу в узле

Протокол:

Узел A :

`send(data, d)`

`doneA(d)`

Узел B :

`receive(data, x)`

`doneB(x)`

Значение команды `receive(t, x)`: дождаться, пока в C появится сообщение типа **t**, и принять это сообщение, сохранив его данные в x

В таком протоколе невозможно дублирование данных, но возможно «зависание» узла B , если сообщение (**data**, d) теряется в канале C

Если считать, что выполнение узла B при зависании завершатся принудительно, то данные в этом случае теряются

Передача данных с двумя сообщениями

Чтобы устранить зависание и соответствующую потерю данных, добавим оповещение $B \rightarrow A$ о приёме данных (**ack**)

Протокол:

Узел A:

```
do {  
  send(data,  $d$ )  
} until await(ack)  
receive(ack)  
doneA( $d$ )
```

Узел B:

```
receive(data,  $x$ )  
send(ack)  
doneB( $x$ )
```

Значение команды **await**(t): некоторое заданное время ожидать сообщение типа t в C

- ▶ если сообщение появилось, то условие истинно
- ▶ если время ожидания истекло и сообщение не появилось, то условие ложно

Передача данных с двумя сообщениями

Узел *A*:

```
do {  
  send(data, d)  
} until await(ack)  
receive(ack)  
doneA(d)
```

Узел *B*:

```
receive(data, x)  
send(ack)  
doneB(x)
```

Тогда оказывается возможно дублирование данных:

A: send(**data**, *d*), await(**ack**) = \mathbb{f} , send(**data**, *d*)

B: receive(**data**, *d*), send(**ack**), done_B(*d*) \square

C: сообщение **ack** потеряно

B: receive(**data**, *d*), send(**ack**), done_B(*d*) \square

A: receive(**ack**), done_A(*d*) \square

Передача данных с двумя сообщениями

Узел A :

```
do {  
  send(data,  $d$ )  
} until await(ack)  
receive(ack)  
done $_A(d)$ 
```

Узел B :

```
receive(data,  $x$ )  
send(ack)  
done $_B(x)$ 
```

Если от A к B передаётся несколько блоков данных, то возможна и потеря:

A : send(**data**, d_1), await(**ack**) = \mathbb{f} , send(**data**, d_1)

B : receive(**data**, d_1), send(**ack**), done $_B(d_1)$

B : receive(**data**, d_1), send(**ack**), done $_B(d_1)$

A : await(**ack**) = \mathbb{t} , receive(**ack**), done $_A(d_1)$

A : send(**data**, d_2)

C : сообщение (**data**, d_2) потеряно

A : await(**ack**) = \mathbb{t} , receive(**ack**), done $_A(d_2)$

Передача данных с тремя сообщениями

Попробуем устранить дублирование данных, возникающее из-за потери сообщения **ack**

Для этого добавим оповещение $A \rightarrow B$ о том, что сообщение **ack** получено (**close**)

Протокол:

Узел A:

```
do {  
  send(data,  $d$ )  
} until await(ack)  
receive(ack)  
send(close)  
doneA( $d$ )
```

Узел B:

```
receive(data,  $x$ )  
do {  
  send(ack)  
} until await(close)  
receive(close)  
doneB( $x$ )
```

Передача данных с тремя сообщениями

Узел A:

```
do {  
  send(data,  $d$ )  
} until await(ack)  
receive(ack)  
send(close)  
doneA( $d$ )
```

Узел B:

```
receive(data,  $x$ )  
do {  
  send(ack)  
} until await(close)  
receive(close)  
doneB( $x$ )
```

Потеря данных по-прежнему возможна при передаче нескольких блоков данных:

A: send(**data**, d_1)

B: receive(**data**, d_1), send(**ack**)

A: await(**ack**) = \top , receive(**ack**), send(**close**), done_A(d_1) \square

A: send(**data**, d_2)

C: сообщения **close** и (**data**, d_2) потеряны

B: await(**close**) = ff , send(**ack**)

A: await(**ack**) = \top , receive(**ack**), send(**close**), done_A(d_2) \square

B: await(**close**) = \top , receive(**close**), done_B(d_1) \square

Передача данных с тремя сообщениями

Чтобы избежать потери данных из-за невозможности различить разные сеансы передачи, разрешим использовать в узле X идентификатор сеанса s_X , уникально обновляющийся при каждом переходе в \square

Улучшенный протокол:

Узел A :

```
do {
  send(data,  $\langle d, s_A \rangle$ )
} until await( $\langle \mathbf{ack}, s_A \rangle$ )
receive( $\langle \mathbf{ack}, s_A \rangle, x$ )
send( $\langle \mathbf{close}, s_A, x \rangle$ )
done $_A(d)$ 
```

Узел B :

```
receive(data,  $\langle x, y \rangle$ )
do {
  send( $\langle \mathbf{ack}, y \rangle, s_B$ )
} until await( $\langle \mathbf{close}, y, s_B \rangle$ )
receive( $\langle \mathbf{close}, y, s_B \rangle$ )
done $_B(x)$ 
```

Чтобы избежать зависаний из-за потери сообщений в канале, будем считать, что, получая сообщение m , не относящееся к текущему сеансу, узел передаёт обратно (**nocon**, m)

В частности, если после отправки **ack** узел B получил соответствующее сообщение **nocon**, то это как то, что узел A получил **ack** и завершил сеанс передачи (с потерей сообщения **close**)

Передача данных с тремя сообщениями

Узел A:

```
do {  
  send(data,  $\langle d, s_A \rangle$ )  
} until await( $\langle \mathbf{ack}, s_A \rangle$ )  
receive( $\langle \mathbf{ack}, s_A \rangle, x$ )  
send( $\langle \mathbf{close}, s_A, x \rangle$ )  
done_A(d)
```

Узел B:

```
receive(data,  $\langle x, y \rangle$ )  
do {  
  send( $\langle \mathbf{ack}, y \rangle, s_B$ )  
} until await( $\langle \mathbf{close}, y, s_B \rangle$ )  
receive( $\langle \mathbf{close}, y, s_B \rangle$ )  
done_B(x)
```

Тогда из-за локальности информации, доступной узлу, по-прежнему возможно дублирование данных:

A: send(**data**, $\langle d_1, a_1 \rangle$), await($\langle \mathbf{ack}, a_1 \rangle$) = ff, send(**data**, $\langle d_1, a_1 \rangle$)

B: receive(**data**, $\langle d_1, a_1 \rangle$), send($\langle \mathbf{ack}, a_1 \rangle, b_1$)

A: receive($\langle \mathbf{ack}, a_1 \rangle, b_1$), send($\langle \mathbf{close}, a_1, b_1 \rangle$), done_A(d_1) □

B: receive($\langle \mathbf{close}, a_1, b_1 \rangle$), done_B(d_1) □

B: receive(**data**, $\langle d_1, a_1 \rangle$), send($\langle \mathbf{ack}, a_1 \rangle, b_2$)

A: receive($\langle \mathbf{ack}, a_1 \rangle, b_2$), send(**nocon**, $\langle \mathbf{ack}, a_1 \rangle, b_2 \rangle$)

B: receive(**nocon**, $\langle \mathbf{ack}, a_1 \rangle, b_2 \rangle$), done_B(d_1) □

Передача данных с четырьмя сообщениями

Последний ошибочный сценарий передачи можно исключить, если заставить узлы синхронизировать общение не в конце, а в начале

Узел A :

```
do {
  send(data,  $\langle d, s_A \rangle$ )
} until await( $\langle$ open,  $s_A$  $\rangle$ )
receive( $\langle$ open,  $s_A$  $\rangle$ ,  $x$ )
do {
  send( $\langle$ agree,  $s_A$ ,  $x$  $\rangle$ )
} until await( $\langle$ ack,  $s_A$ ,  $x$  $\rangle$ )
receive( $\langle$ ack,  $s_A$ ,  $x$  $\rangle$ )
done $_A$ ( $d$ )
```

Узел B :

```
receive(data,  $\langle x, y \rangle$ )
do {
  send( $\langle$ open,  $y$  $\rangle$ ,  $s_B$ )
} until await( $\langle$ agree,  $y$ ,  $s_B$  $\rangle$ )
receive( $\langle$ agree,  $y$ ,  $s_B$  $\rangle$ )
done $_B$ ( $x$ )
send( $\langle$ ack,  $\langle s_A, x \rangle$  $\rangle$ )
```

Задача 1. Покажите, что если обработка сообщений типа **носон** в узле A устроена корректно относительно возможности выхода узла B из строя, то в этом протоколе возможно дублирование данных, даже если узел B при этом фактически не выходит из строя.

Задача 2. Предложите протокол с двумя сообщениями, согласно которому невозможна потеря данных (хотя, быть может, допускается дублирование) даже с учётом нескольких сеансов передачи. Приведите обоснование того, что данные действительно не теряются

Подсказка: исключить потерю данных можно при помощи идентификаторов сеанса, как в протоколе с тремя сообщениями

Задача 3 (трудная). Предложите протокол передачи с пятью сообщениями, который даже с учётом нескольких сеансов передачи и выхода одного из узлов из строя

- ▶ не допускает потерю данных и
- ▶ допускает дублирование только в том случае, если узел фактически выходит из строя