Синхронизация времени в распределенных компьютерных системах

Гайдамака Андрей, 18205

Часы Лэмпорта

Описание

Часы Лэмпорта реализуют алгоритм определения последовательности событий в распределённых системах, алгоритм разработан Лэсли Лэмпортом в 1978 году и является прообразом алгоритма векторных часов.

История

В 1978 году Лэмпорт (Lamport) показал, что синхронизация времени возможна, и предложил алгоритм для такой синхронизации. При этом он указал, что абсолютной синхронизации не требуется. Если два процесса не взаимодействуют, то единого времени им не требуется. Кроме того, в большинстве случаев согласованное время может не иметь ничего общего с астрономическим временем. В таких случаях можно говорить о *логических часах*.

Отношение “произошло до”

Для синхронизации логических часов Lamport определил отношение "произошло до". Выражение **a->b** читается как "**a** произошло до **b**" и означает, что все процессы согласны, что сначала произошло событие "**a**", а затем "**b**". Это отношение может в двух случаях быть очевидным:

1. Если оба события произошли в одном процессе.
2. Если событие **a** есть операция SEND в одном процессе, а событие **b** - прием этого сообщения другим процессом.

Отношение **->** является транзитивным.

Если два события **x** и **y** случились в различных процессах, которые не обмениваются сообщениями, то отношения **x->y** и **y->x** являются неверными, а эти события называют одновременными.

Минимальные характеристики узлов распределённой системы для применения алгоритма:

- отсутствие синхронизации с другими узлами
- способность к приёму и передаче данных один-на-один
- поле счётчика с функциями чтения, записи, сравнения и инкремента

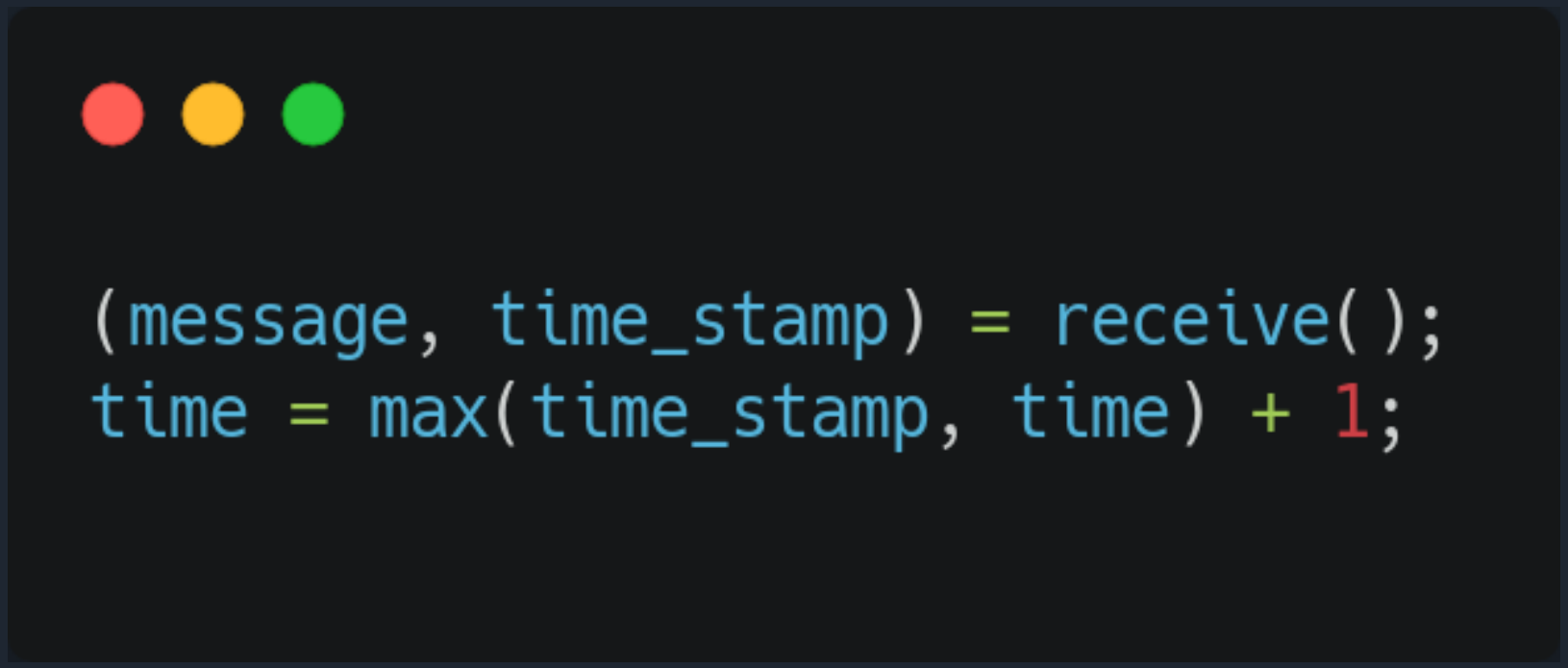
Алгоритм временных меток Лэмпорта может быть записан в нескольких правилах:

- Все счетчики процессов начинаются со значения 0.
- Процесс увеличивает свой счетчик для каждого события (внутреннее событие, отправка сообщения, получение сообщения) в этом процессе.
- Когда процесс отправляет сообщение, он включает (увеличенное) значение счетчика вместе с сообщением.
- При получении сообщения счетчик получателя обновляется до большего из его текущего счетчика и отметки времени в полученном сообщении, а затем увеличивается на единицу.

Алгоритм отправки

```
# event is known
time = time + 1;
# event happens
time_stamp = time;
send(message, time_stamp);
```

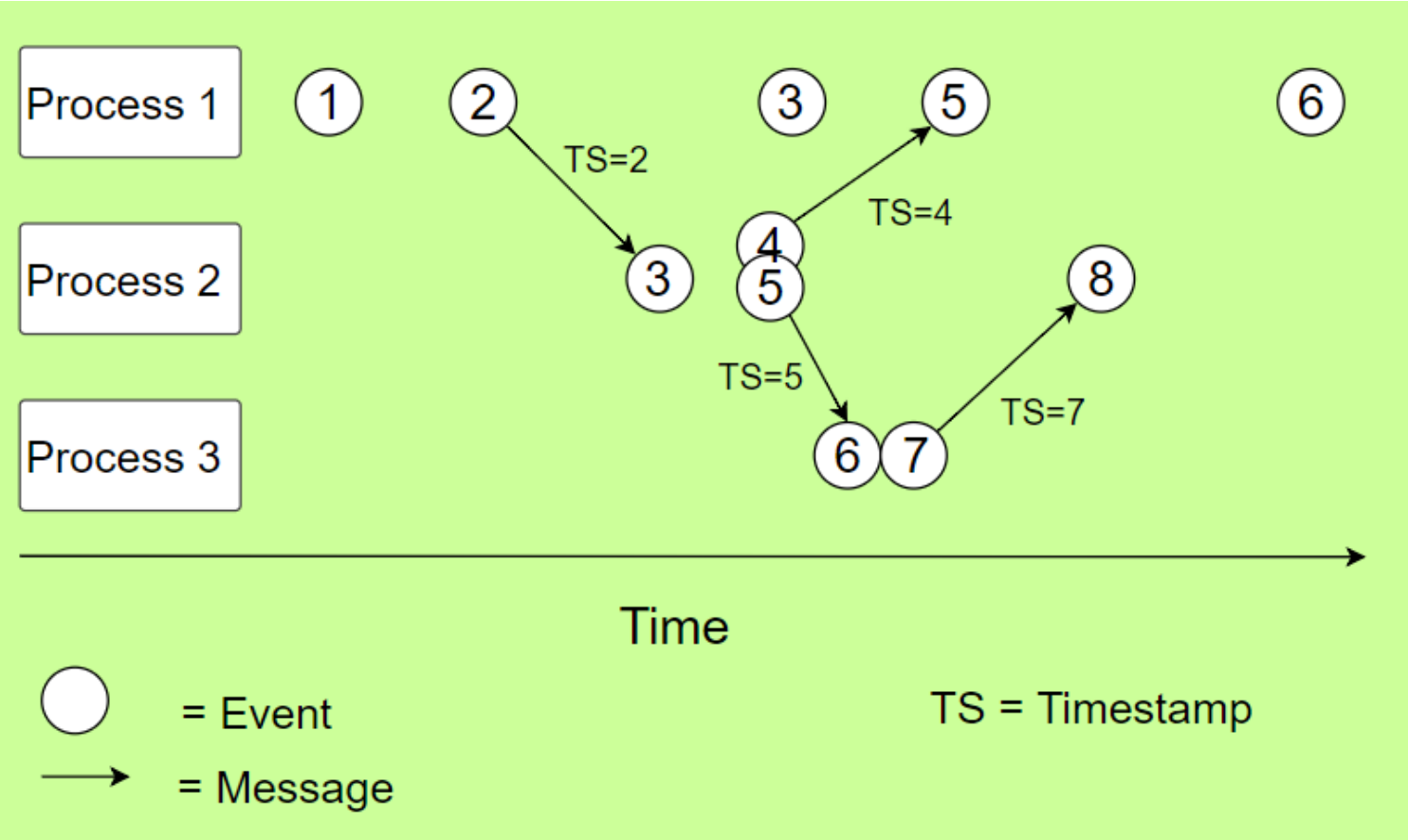
Алгоритм для приема



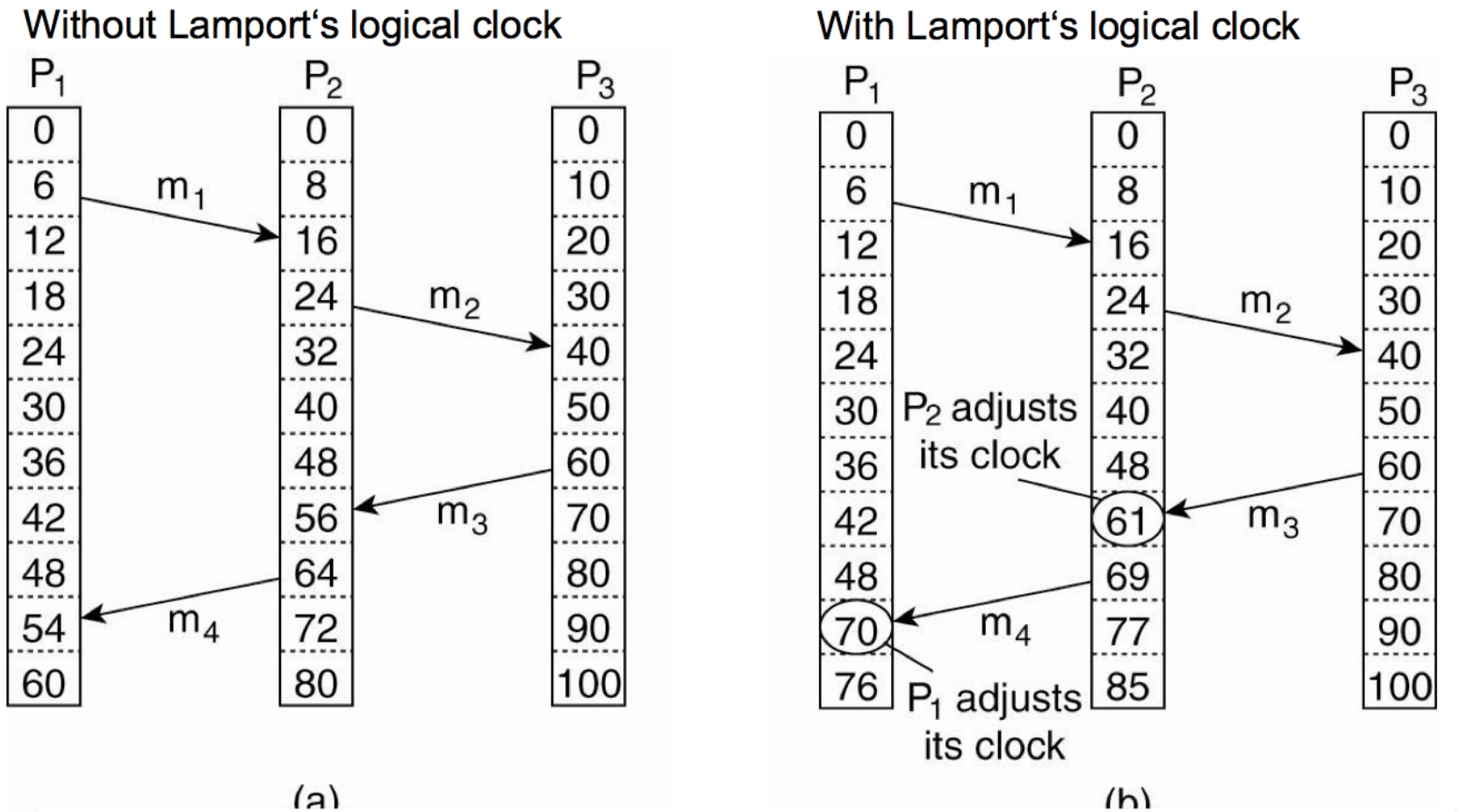
```
(message, time_stamp) = receive();
time = max(time_stamp, time) + 1;
```

Пример работы часов Лэмпорта

Все процессы начинаются с их внутреннего счетчика (часов) в нуле.



Пример работы часов Лэмпорта



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Оценка часов Лэмпорта

Алгоритм Лэмпорта создает минимальные издержки, поскольку счетчик состоит только из одного целочисленного значения. Но главный их недостаток, то что они только частично упорядочивают события, то есть они не могут определить, являются ли события одновременными или нет. Эта проблема решается с помощью векторных часов

Векторные часы

Описание

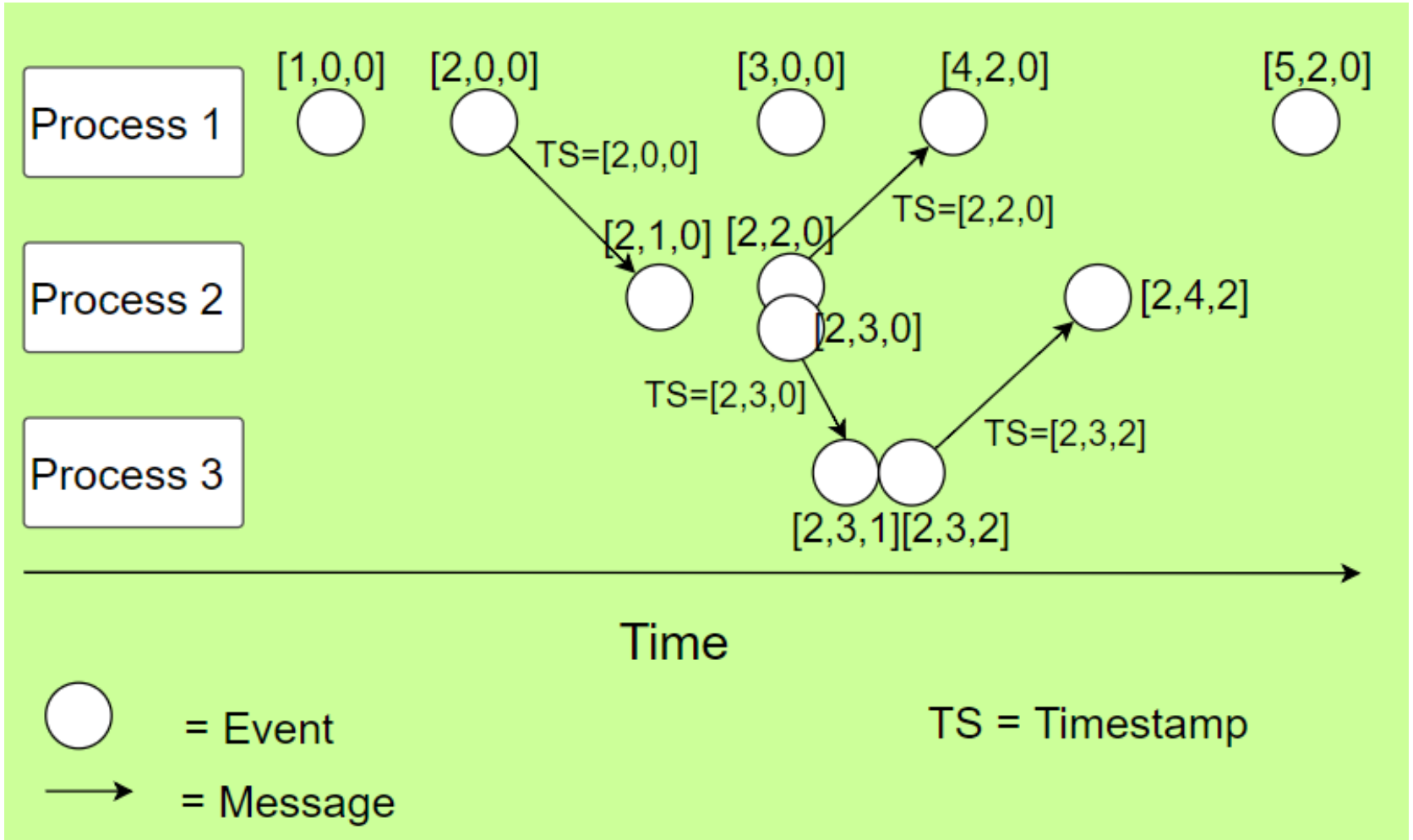
Как и во временных метках Лэмпорта, внутренние сообщения, передаваемые в системе, содержат состояние логических часов процесса. Векторные часы в системе N процессов — массив или вектор из N логических часов, одни часы на процесс.

Векторные часы были разработаны независимо Фиджем и Маттерном в 1988 году

Алгоритм построения вектора

- изначально все значения часов равны 0;
- в случае внутреннего события счётчик текущего процесса увеличивается на 1;
- перед отправкой сообщения внутренний счётчик, соответствующий текущему процессу, увеличивается на 1, и вектор целиком прикрепляется к сообщению;
- при получении сообщения счётчик текущего процесса увеличивается на 1, далее значения в текущем векторе выставляются в максимум от текущего и полученного.

Пример работы векторных часов



Строгая непротиворечивость

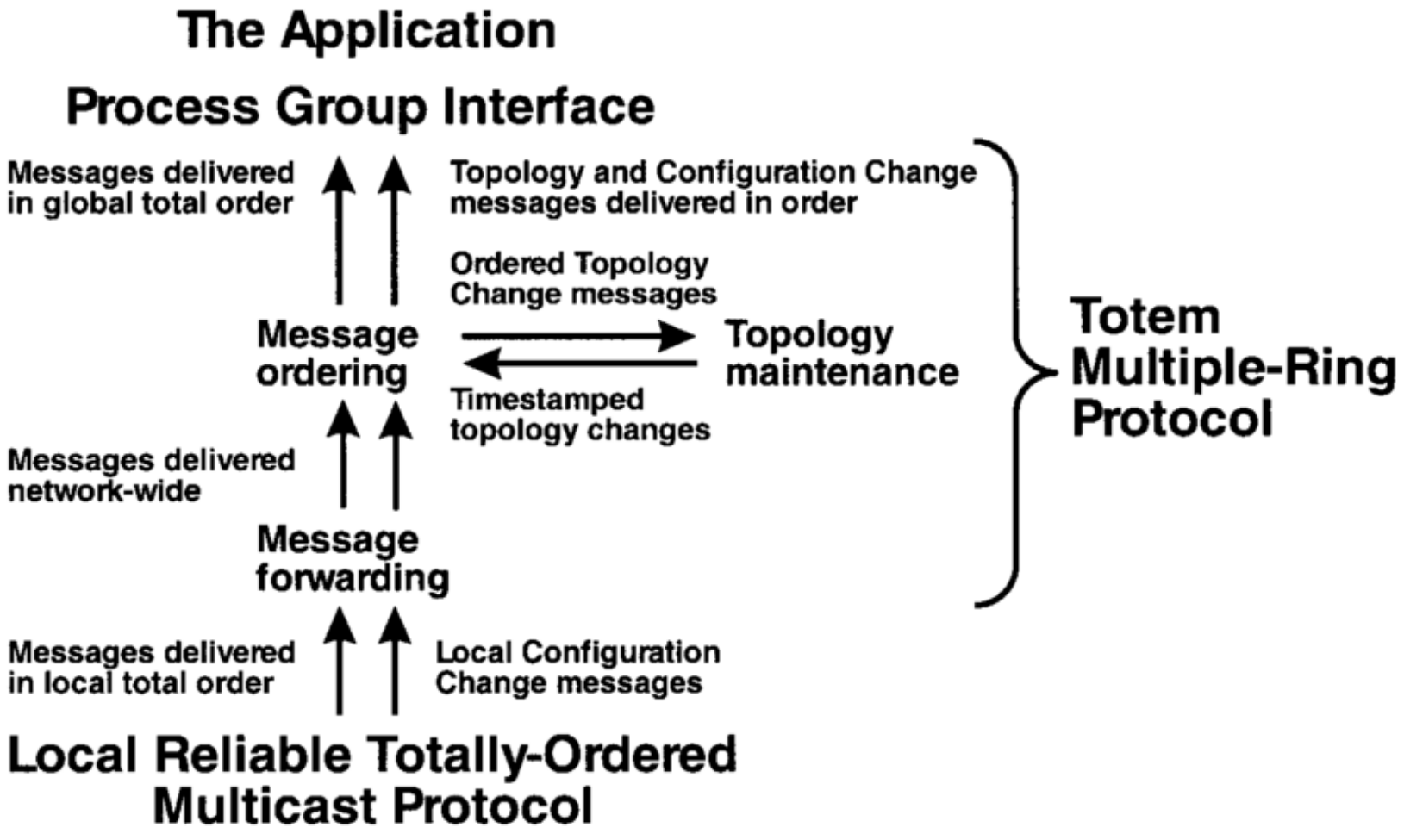
Векторные часы являются строго непротиворечивыми, что позволяет по отметкам времени двух событий судить о том, оказывают ли эти события потенциальное влияние друг на друга или нет. Поэтому, в отличие от скалярных часов, позволяющих построить *одно из эквивалентных* выполнений распределенной системы, векторные часы могут быть использованы для определения *всех возможных* выполнений. Свойство строгой непротиворечивости логических часов является очень важным: благодаря ему векторные часы широко применяются при построении распределенных алгоритмов.

Недостаток векторных часов

Основным недостатком механизма векторных часов является необходимость хранить для каждого события и пересылать в каждом сообщении отметки времени, состоящие из *N* целых чисел, что может приводить к значительным накладным расходам, особенно, в случае большого числа *N* процессов, работающих в распределенной системе. Однако было показано, что векторами меньшей длины для определения причинно-следственной зависимости между событиями по их отметкам времени обойтись невозможно.

Часы Лэмпорта в Totem

Многоуровневая иерархия протокола Totem



Часы Лэмпорта в Totem

В протоколе Totem используется модификация часов Лэмпорта, том смысле, что он применяется к сообщениям, а не к событиям, и ограничивается сообщениями, которые возникли в одной конфигурации. Это позволяет повторно объединить разделенную сеть и присоединить неисправный процессор, не требуя доставки всех сообщений в истории.

Causal Order for Configuration C

Рефлексивное, транзитивное отношение ”произошло до”, которое определено для всех процессоров p, которые являются членами C, следующим образом:

- Сообщение m1 предшествует сообщению m2, если процессор p создал m1 в конфигурации C до того, как p создал m2 в C.
- Сообщение m1 предшествует сообщению m2, если процессор p создал m2 в конфигурации C, а p доставил m1 в C до создания m2.

Algorithm executed by a gateway for the forwarding of messages

```
if msg.timestamp > seen_table[msg.src_ring_id].timestamp then
   seen_table[msg.src_ring_id].timestamp = msg.timestamp
   forward message
endif
```

Пример упорядочивания сообщений в Totem-Multiple Ring

Example. Consider the network shown in Figure 7, where the rings are represented by circles and the processors and gateways by squares. A processor p on ring A is ordering messages from rings A, B, and C. The first row of the *guarantee_array* at processor p is the vector of *max_timestamp* values at p. The second and third rows are the *guarantee_vectors* in the Guarantee Vector messages for rings B and C.

Processor p can deliver in agreed order the message with timestamp 7 from ring A, the messages with timestamps 9 and 10 from ring B, and the messages with timestamps 8 and 10 from ring C. After those messages have been delivered, the *min_timestamp* and *max_timestamp* at processor p for ring C will be set to 10 until it receives further messages from ring C. The undelivered message with the lowest timestamp is the message from ring B with timestamp 13, but p cannot deliver that message until it receives the next message from ring C. Otherwise, p may subsequently receive a message from ring C with timestamp 12.

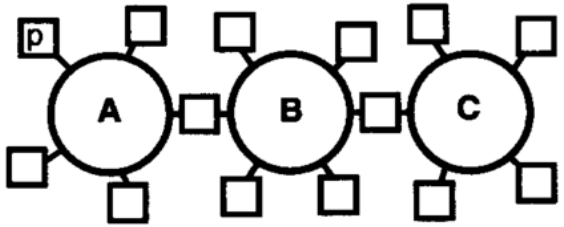
If processor p receives the message from ring A with timestamp 7 that contains a request for safe delivery, then p can deliver that message as safe because the column of the *guarantee_array* corresponding to ring A has all entries at least equal to 7, which indicates that the message is safe on all rings in the current topology. The same is true for the message from ring B with timestamp 9 and the message from ring C with timestamp 8. Processor p cannot, however, deliver the message from ring B with timestamp 10 as safe because the guarantee vector from ring C reports receipt of messages from B only up through timestamp 9.

Data structures at processor p

A: Recv_msgs = 7,14,15,26,29
Max_timestamp = 29
Min_timestamp = 7

B: Recv_msgs = 9,10,13,15
Max_timestamp = 15
Min_timestamp = 9

C: Recv_msgs = 8,10
Max_timestamp = 10
Min_timestamp = 8



	A	B	C
A	29	15	10
B	14	15	10
C	7	9	10

guarantee array at p

Список литературы

- ACM Transactions on Computer Systems, Vol 13, No.4, November 1995, Pages 311-342 Amir, Y., Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., Ciarfella, P.: The Totem single-ring ordering and membership protocol.
- Косяков М.С. Введение в распределенные вычисления. – СПб: НИУ ИТМО,2014. – 155 с.
- Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. 1978.
- ACM Transactions on Computer Systems, Vol. 16, No. 2, May 1998, Pages 93–132D. A. Agarwal, L. E. Moser, P. M. Melliar-Smith, and R. K.Budhia.:The Totem Multiple-Ring Ordering and Topology Maintenance Protocol

Спасибо за внимание!

Андрей Гайдамака
НГУ, группа 18205
a.gaidamaka1@g.nsu.ru