РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
АВТОМЕТРИЯ
№ 5 1997

УДК 681.3.06

Ю. И. Колосова
*(Новосибирск)*

# ПОСТРОЕНИЕ ТРАСС ПРИЧИННО УПОРЯДОЧЕННЫХ СОБЫТИЙ РАСПРЕДЕЛЕННОГО ВЫЧИСЛЕНИЯ

Предлагаются два класса автономных алгоритмов, которые в совокупности обеспечивают систематическое построение трасс причинно упорядоченных событий на основе трасс, полученных при распределенном вычислении, которое может выполняться как без протокола, так и с протоколом реализации причинно упорядоченной доставки сообщения. Показано, что, хотя известные протоколы и гарантируют причинный порядок, однако он может не совпадать с требуемым. Рассматриваются особенности применения новых трасс в циклической отладке таких вычислений.

**Введение.** Рассматриваются распределенные вычисления из $N$ процессов $P_1, \ldots, P_N$, которые обмениваются данными и узнают о состоянии друг друга только посредством посылки и приема сообщений по надежным каналам связи с конечной задержкой.

Любое действие в локальном вычислении каждого из процессов $P_i$ традиционно называется событием $e$. Результаты события $e$ потенциально могут влиять на результаты события $e'$, если $e'$ причинно зависит от $e$. Неформально такая зависимость означает: «если $e$ заблокировано, то $e'$ не может произойти».

В работе рассматриваются только события посылки $s$ и приема $r$, т. е. $e = s \vee r$. Если $s^x$ и $s^z$ есть посылки сообщений $x$ и $z$ в один и тот же процесс $P_k$ и $s^z$ причинно зависит от $s^x$, то вследствие асинхронной природы каналов связи приемы этих сообщений в $P_k$, т. е. события $r^x$ и $r^z$, могут произойти в произвольном порядке. Если $x$ и $z$ приняты в том же порядке, в котором они посылались в $P_k$, то события $r^x$ и $r^z$ являются *причинно упорядоченными*. Потенциальная возможность такого порядка событий и выдвигает необходимость проверки распределенного вычисления в условиях его сохранения.

Существуют протоколы [11, 9, 8, 3], реализующие *причинно упорядоченную доставку сообщения*. Они гарантируют причинную упорядоченность событий $r^x$ и $r^z$ для каждой пары событий, которые подобны указанным выше $s^x$ и $s^z$. Однако если $s^z$ не находится в причинной зависимости от $s^x$, то события $r^x$ и $r^z$ могут произойти в произвольном порядке, который может не соответствовать требуемому. Поэтому возникает проблема проверки распределенных вычислений и с такими протоколами.

В работе предлагаются два класса автономных алгоритмов, которые в совокупности обеспечивают систематическое построение трасс потенциально возможных причинно упорядоченных событий $r$ на основе трасс, содержащих только события $s$ и полученных при распределенном вычислении, которое может быть выполнено как без протокола, так и с одним из указанных выше протоколов. Результирующие трассы могут использоваться в управлении повторным выполнением распределенного вычисления на тех же самых исходных данных, обеспечивая его циклическую отладку.

1. **Модель распределенного вычисления.** С абстрактной точки зрения распределенное вычисление может быть описано типами и относительным порядком событий $s$ и $r$, случившихся в каждом из процессов $P_i$. Событие $s$ является неблокированным: выполнив посылку сообщения заданному адресату, процесс продолжает вычисление. Событие $r$ является блокированным: вычисление процесса приостанавливается до прибытия любого сообщения. Предполагается, что операторы выборочного приема не используются.

Если $s^m$ есть посылка сообщения $m$ из $P_i$, а $r^m$ есть прием этого $m$ в процессе $P_j$ $(i \neq j)$, то говорят, что $s^m$ *соответствует* $r^m$. Поскольку предполагается связь точка в точку, то каждая пара $(s_i^m, r_j^m)$ является единственной.

Пусть $E_i = \{e_{i1}, e_{i2}, \ldots,\}$ есть множество всех событий в порядке появления при вычислении процесса $P_i$, $E = E_1 \cup E_2 \cup \ldots \cup E_N$ и $\Gamma = E \times E$.

Будем рассматривать только такие распределенные вычисления (назовем их *полными*), где для каждого $r_j$ существует *соответствующее* $s_i$ и наоборот (т. е. нет сообщений в транзите к концу вычисления). Если два события $e_i$ и $e_i'$ принадлежат одному $P_i$, то по аналогии с [8] будем обозначать это как $e \sim e'$.

Рассмотрим *отношение причинности* $<^*$ на множестве событий $E$. Поскольку вычисление каждого процесса $P_i$ является последовательным, то его события тотально упорядочены своим появлением при вычислении $P_i$. Этот порядок, который будем обозначать как $<_i$, вызывает причинный порядок на событиях $E_i$.

**Определение 1.** *Отношение причинности $<$ на $E$ есть наименьшее отношение частичного порядка, которое удовлетворяет следующим трем свойствам:*

*если* $e \sim e' \wedge e <_i e'$, *то* $e < e'$;

*если* $(s, r) \in \Gamma$, *то* $s < r$;

*если* $e < e' \wedge e' < e''$, *то* $e < e''$.

Это отношение первоначально было названо в [4] отношением «случилось перед», однако здесь по аналогии с [8] выбран другой термин, чтобы подчеркнуть именно причинную зависимость между событиями.

Отметим, что не все события из $E$ должны находиться в отношении $<$. Если $e \not< e' \wedge e' \not< e$, то эти события *параллельны* $e \parallel e'$.

2. **Свойства протоколов причинно упорядоченной доставки сообщений.** Распределенные вычисления будем традиционно представлять посредством пространственно-временных диаграмм [4]. Каждый вектор слева направо представляет вычисление процесса, события показаны точками на линиях процессов, а стрелки изображают передачу сообщения. Событие $e'$ потенциально может *причинно зависеть* от события $e$, если и только если существует путь в диаграмме от $e$ к $e'$. Например, на рис. 1 процессы $P_1$ и $P_2$ посылают

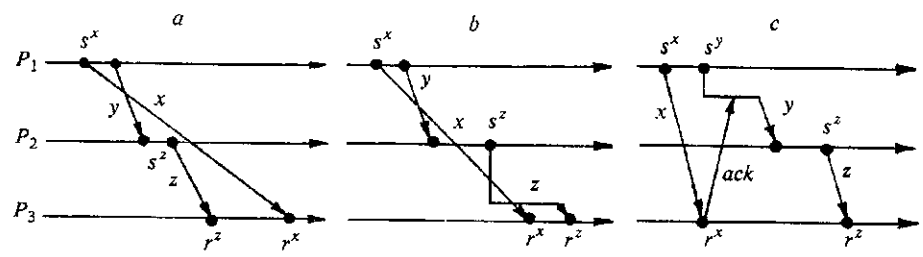


*Рис. 1.* Распределенное вычисление:
*a* — без протокола; *b* — с матричным протоколом; *c* — с буферным

* Далее часто даются ссылки на работу [8], поэтому используются основные обозначения из нее для удобства сравнения.

сообщения $x$ и $z$ соответственно процессу $P_3$. В каждой из диаграмм существует путь от события $s^x$ к событию $s^z$, поэтому $s^x < s^z$. Поскольку вычисления на рис. 1, *a* выполняются без протоколов, то процесс $P_3$ может получать сообщения в любом порядке. В данном случае сообщение $z$ «обгоняет» $x$ (т. е. $r^z < r^x$).

Протоколы причинно упорядоченной доставки гарантируют, что если $r \sim r' \wedge s < s'$, то $r < r'$ для любой пары соответствующих событий $(s, r)$, $(s', r')$. Такие протоколы разделяют на два типа: матричные [11, 9, 3] и буферные [8]. Матричный протокол может задерживать сообщение в принимающем процессе, а буферный — в посылающем. На рис. 1, *b* доставка $z$ задерживается до реализации доставки $x$ (т. е. $r^x < r^z$). На рис. 1, *c* посылка $s^y$ из $P_1$ задержана до прибытия *ack*, сообщающей, что $x$ принято во входной буфер $P_3$. Только после этого выполняется посылка $y$ в процесс $P_2$, поэтому $s^x < s^z$ и $r^x < r^z$.

На рис. 1, *b*, *c* показано, что между посылками $s^x$ и $s^z$ существует прием сообщения $y$ в процессе $P_2$. Именно событие $r^y$ и вызывает причинную зависимость $s^z$ от $s^x$. Однако поскольку в асинхронном распределенном вычислении могут произойти и параллельные события, то вместо $y$ может быть принято другое сообщение, которое может препятствовать причинно упорядоченной доставке $x$ и $z$.

На рис. 2 показано вычисление из четырех процессов. Процессы $P_1$ и $P_2$ посылают те же самые сообщения $x$ и $z$ процессу $P_3$. Процесс $P_4$ выполняет посылку $s^w$ в процесс $P_2$ такую, что $s^y \parallel s^w$. На рис. 2, *a* сообщение $y$ прибывает в $P_2$ раньше $w$, поэтому, как и на рис. 1, *b*, *c*, существует путь от $s^x$ к $s^z$, т. е. $s^x < s^z$ и $r^x < r^z$. На рис. 2, *b* сообщение $w$ «обгоняет» $y$, поэтому $s^x \not< s^z$ и доставка сообщения $x$ может следовать за доставкой сообщения $z$, т. е. $r^z < r^x$.

Только разработчик знает необходимые причинно упорядоченные доставки сообщений. Поэтому предлагаемые в следующем разделе алгоритмы и ориентированы на построение трасс потенциально возможных причинно упорядоченных событий. Вначале, однако, остановимся еще на одном свойстве, присущем буферному протоколу.

На рис. 3 все события посылки из разных процессов параллельны: $s^a \wedge s^d \parallel s^b \wedge s^f \parallel s^c \wedge s^g$. В асинхронном вычислении порядок соответствующих событий приема двух сообщений в каждом из процессов может быть произвольным. На рис. 3, *a* показано, что первыми были приняты сообщения, которые посылались первыми каждым из процессов. Чтобы не загромождать рисунки, стрелки сообщений опущены, указаны только их начало $(s^i)$ и конец $(r^i)$, т. е. пары соответствующих событий. На рис. 3, *b* показано, что первыми были приняты сообщения, которые посылались вторыми каждым из процессов. Таким структурным зависимостям событий в [8] дано понятие *корона*: последовательность $\langle (s^i, r^i), i \in \{1, \ldots, k\} \rangle$ пар соответствующих событий посылки и приема такая, что $s^1 < r^2$, $s^2 < r^3$, ..., $s^{k-1} < r^k$, $s^k < r^1$ есть корона (размером $k$).

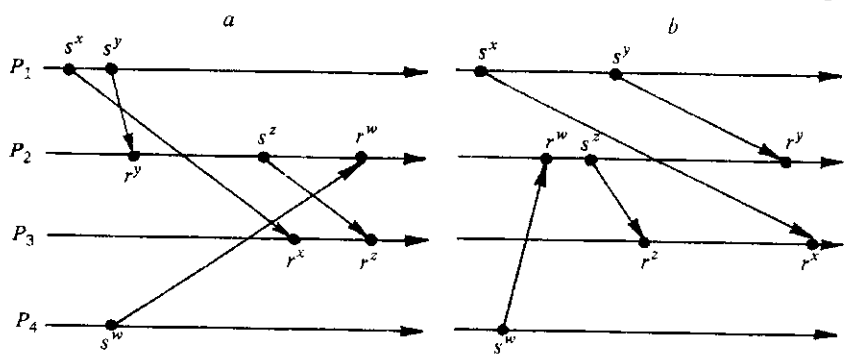


*Рис. 2.* Влияние параллельных событий $s^y$ и $s^w$:
*a* — $y$ «обгоняет» $w \Rightarrow s^x < s^z \wedge r^x < r^z$; *b* — $w$ «обгоняет» $y \Rightarrow s^x \not< s^z \wedge r^z < r^x$

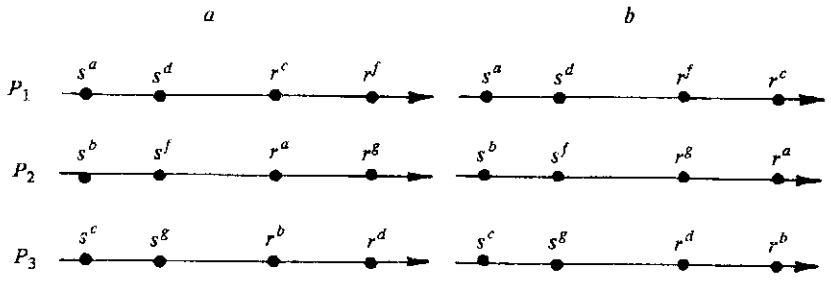


*Рис. 3.* Вычисления, содержащие короны:
$a$ — типа 1: $s^a < r^c$; $s^c < r^b$; $s^b < r^a$ и $s^d < r^f$; $s^f < r^g$; $s^g < r^d$; $b$ — типа 2: $s^a < r^f$; $s^f < r^g$; $s^g < r^d$; $s^d < r^c$; $s^c < r^b$; $s^b < r^a$

На рис. 3 показаны два характерных типа *корон*, которые зависят от порядка приема сообщений. Первый тип потенциально возможен при использовании как матричного, так и буферного протокола. Второй тип может быть воспроизведен только при матричном протоколе. Действительно, поскольку такой протокол не задерживает доставку сообщений, посланных параллельными событиями, то вторые сообщения каждого из процессов могут быть доставлены первыми, если они «обогнали» первые сообщения. При буферном протоколе хотя бы одно первое сообщение одного из процессов всегда будет принято первым. Интуитивно ясно, что поскольку сообщение посылается из выходного буфера отправителя только после расписки в получении ранее посланного сообщения, то все вторые сообщения не могут быть доставлены первыми. В [8] это рассмотрено на формальном уровне. Далее показано, что предлагаемые алгоритмы одного из классов способны воспроизвести указанную структуру.

3. **Алгоритмы построения трасс причинно упорядоченных событий.** Рассматриваются два класса автономных алгоритмов построения трасс причинно упорядоченных событий приема на основе исходных трасс, полученных при распределенном вычислении. Алгоритмы первого класса просматривают исходные трассы только в прямом направлении, однако они не охватывают все потенциально возможные структуры упорядочения событий. Алгоритмы второго класса способны получить такие структуры, однако они просматривают исходные трассы в прямом и обратном направлениях. Они могут быть полезны, если трасс, построенных более простыми алгоритмами первого класса, окажется недостаточно для отладки и анализа исследуемого вычисления.

3.1. *Структуры данных.* Ниже рассматриваются структуры данных, которые используются как в алгоритмах первого, так и второго класса.

Пусть в течение распределенного вычисления в каждом из $P_i$ ($i \in \{1, \ldots, N\}$) велся счет всех событий связи, т. е. событий $s$ и $r$, однако в трассу $S_i$ записывались только $s$. Каждое $s$ состоит из двух полей: $s[pole1] = P_j$; $s[pole2] = k_i$, где $P_j$ есть номер процесса-адресата, а $k_i$ — порядковый номер $s$. Признаком конца трассы является такое $s$, у которого $s[pole1] = 0$ и $s[pole2] = |E_i| + 1$. События $s$ указаны явно, а события $r$ — неявно: события $r$ случились между двумя последовательными $s_i'$ и $s_i''$ в трассе $S_i$, если $s_i''[pole2] - s_i'[pole2] > 1$. Число $r$ перед самым первым $s_i''$ равно $s_i''[pole2] - 1$. Например, для вычисления из рис. 2, $a$ трассы $S_i$ показаны на рис. 4 (для ясности содержимое $s[pole1]$ указано именем процесса, а не просто его номером).

Цель алгоритмов — определить все неявные события и построить новую трассу $R_i$ для каждого $P_i$, состоящую только из $r$. Причем ни одно $r \in R$, где $R = \cup_i^N R_i$, не должно нарушать потенциально возможное отношение причинности на множестве событий $E$ $R \cup S$, где $S = \cup_i^N S_i$. Каждое $r \in R_i$ также состоит из двух полей: $r[pole1] = P_j$; $r[pole2] = k_j$, где $P_j$ есть номер процесса

| $P_1$ | | $P_2$ | | $P_3$ | | $P_4$ | |
|---|---|---|---|---|---|---|---|
| $P_3$ | 1 | $P_3$ | 2 | 0 | 3 | $P_2$ | 1 |
| $P_2$ | 2 | 0 | 4 | | | 0 | 2 |
| 0 | 3 | | | | | | |

*Рис. 4.* Трассы событий $s$ из вычисления на рис. 2, *a*

источника сообщения, а $k_j$ есть (как и в первом случае) порядковый номер соответствующего $s \in S_j$.

Алгоритмы используют следующие общие переменные:

$p\_s$ — массив $[1, ..., N]$ индексов очередных $s \in S_i$.

$p\_r$ — массив $[1, ..., N]$ индексов очередных $r$, помещаемых в соответствующие $R_j$. В исходном состоянии $\forall j : p\_r[j] = 1$.

$all\_e$ — массив $[1, ..., N]$ ожидаемых порядковых номеров $all\_e[i]$ событий в каждой $S_i$. Поскольку в $S_i$ содержатся только $s$, то $s[pole2] = all\_e[i] \vee \vee s[pole2] > all\_e[i]$. Если $s[pole2] = all\_e[i]$, то ожидаемое событие есть $s$. Если же $s[pole2] > all\_e[i]$, то ожидаемое событие есть $r$. Если все события любой из $S_i$ исследованы, т. е. $s[pole2] = |E_i| + 1$, то $all\_e[i] := 0$.

$mark$ — массив $[1, ..., N]$ *отмеченных* событий $r$. Значение $mark[j]$ увеличивается, как только $r$ помещается в $R_j$, и уменьшается (если $mark[j] \neq 0$), как только ожидаемым событием в $S_j$ становится $r$.

3.2. *Алгоритмы первого класса.* На рис. 5 представлен алгоритм1, состоящий из двух вложенных циклов. Внутренний цикл выполняется $N$ раз и на каждой итерации осуществляет анализ и обработку одного или нескольких ожидаемых событий из трассы $S_i$.

Внешний цикл заканчивается, как только будут исчерпаны все события каждой из трасс.

На рис. 6 показаны возможные трассы событий $r$, построенные на основе трасс из рис. 4. Для рис. 6, *a* $P_\alpha = 1$, а для рис. 6, *b* этот параметр может быть равен 2, 3 или 4.

В алгоритме1, который представляет вариант1 алгоритмов первого класса, внутренний цикл начинается с трассы процесса, задаваемого параметром $P_\alpha$. Вариант2 — это алгоритм1, в котором изменена строка, помеченная (*): заменить $i := P_\alpha$ на $i := (i mod N) + 1$, а первое присваивание $i := P_\alpha - 1$ выполнить перед **repeat**. Тогда цикл будет начинаться с трассы иного процесса. Вариант3 — это алгоритм1, в котором изменена строка, помеченная (**): заменить go to 1 на go to 2. Тогда на каждой итерации обрабатывается только одно ожидаемое $s$. С помощью варианта3 можно получить трассы, соответствующие вычислению из рис. 3, *a*. Вариант4 — это алгоритм1, в котором изменены обе строки.

Можно привести и другие алгоритмы этого класса. Например, внутренний цикл выполняется только для группы трасс (например, для трасс с четными номерами), и внешний цикл повторяет его до тех пор, пока либо эти трассы не будут исчерпаны, либо ожидаемыми будут неотмеченные $r$. Затем цикл выполняется для другой группы трасс. Отметим, что именно при таком алгоритме можно получить трассы $R_j$, соответствующие вычислению из рис. 2, *b*.

Принципиальным для всех алгоритмов является разрешение допуска к обработке одного ожидаемого события трассы $S_i$ только после обработки предыдущего. Такой подход ограничивает получение некоторых потенциально возможных порядков событий $r$. Например, ни один из алгоритмов этого класса не может построить трассы $R_j$, соответствующие вычислению из рис. 3, *b*. Далее показано, как можно получить подобные трассы.

3.3. *Алгоритмы второго класса.* Алгоритмы второго класса разрешают допуск к обработке не одного события $s$ исходной трассы $S_i$, а фрагмента ее последовательных событий $s$.

*Фрагмент* $F_i$ есть последовательность событий $s \in S_i$ таких, что $\forall s, s' : : s, s' \in F_i : r \not\sim r'$.

Формирование $F_i$ заканчивается при выборе или неотмеченного $r$, или конца трассы, или события $s''$ такого, что $\exists s : s \in F_i : r \sim r''$. Затем $s \in F_i$ обра-

```
CausalOrder1 (P_α, N, S = ∪_{i=1}^{N} S_i, R = ∪_{j=1}^{N} R_j)
\* α ∈ {1, ..., N}; 1 ≤ i ≤ N; N — число процессов *\
  ∀k : 1 ≤ k ≤ N:
  p_s[k] := 1; p_r[k] := 1; all_e[k] := 1; mark[k] := 0;
   repeat
   i := P_α                                                    (*)
    for β := 1 to N do
     if all_e[i] = 0 then go to 2
1.   Выбор s из S_i по индексу p_s[i]
     if s[pole1] = 0 ∧ s[pole2] = all_e[i]
     then begin all_e[i] := 0; go to 2 end
     if s[pole2] = all_e[i]
     then begin                          \* Обработка s_i *\
        r[pole1] := i; r[pole2] := s[pole2]; j := s[pole1];
        Вставка r в R_j по индексу p_r[j];
        mark[j] := mark[j] + 1; p_r[j] := p_r[j] + 1;
        p_s[i] := p_s[i] + 1;
        all_e[i] := all_e[i] + 1; go to 1                        (**)
           end
     if s[pole2] > all_e[i]
     then begin
        if mark[i] = 0 then go to 2
        else begin                       \* Обработка r_i *\
           mark[i] := mark[i] − 1;
           all_e[i] := all_e[i] + 1; go to 1
            end
         end
2.   i := (imodN) + 1;
     endfor
   until ∀δ : 1 ≤ δ ≤ N : all_e[δ] = 0
   endrepeat
   return (R_j : 1 ≤ j ≤ N)
end CausalOrder1
```

*Рис. 5.* Алгоритм1 построения причинно упорядоченных трасс $R_j : 1 \le j \le N$ событий приема

*a*

| $P_1$ | $P_2$ | | $P_3$ | | $P_4$ |
|---|---|---|---|---|---|
| | $P_1$ | 2 | $P_1$ | 1 | |
| | $P_4$ | 1 | $P_2$ | 2 | |

*b*

| $P_1$ | $P_2$ | | $P_3$ | | $P_4$ |
|---|---|---|---|---|---|
| | $P_4$ | 1 | $P_1$ | 1 | |
| | $P_1$ | 2 | $P_2$ | 2 | |

*Рис. 6.* Возможные трассы событий приема, построенные на основе трасс из рис. 4:
$a$ — $P_\alpha = 1$; $b$ — $P_\alpha = 2 \vee 3 \vee 4$

батываются в любом, предусмотренном алгоритмом порядке. Принципиальным является допуск к формированию очередного $F_i$ только после обработки всех $s$ предыдущего.

На рис. 7 показан алгоритм2, внешний цикл которого содержит два основных внутренчих цикла. Первый такой цикл формирует для каждой трассы текущий фрагмент, а второй -- обрабатывает все события этих фрагментов.

Рассмотрим дополнительные переменные, которые используются в обоих циклах.

$qu\_f$ — массив $[1, ..., N]$ счетчиков $s$ в текущих фрагментах $F_i$ каждой из $S_i$. Счетчик $qu\_f[i]$ увеличивается при включении $s$ в $F_i$ и обнуляется после обработки всех $s$ из текущего $F_i$. В исходном состоянии $\forall i : qu\_f[i] = 0$, что разрешает допуск к формированию фрагмента в каждой из $S_i$.

min — минимальная длина текущего фрагмента $\forall i : 1 \le i \le N : \min \le \le qu\_f[i] \wedge qu\_f[i] \ne 0$. В исходном состоянии $\min = N - 1$, поскольку $\forall i : qu\_f[i] \le N - 1$ (все события из любого $F_i$ посылают сообщения разным процессам).

*point* — вспомогательный массив $[1, ..., N]$. В цикле формирования фрагментов $point[j]$ используется для фиксирования процесса-адресата $P_j$ каждого $s$, выбираемого в текущий $F_i$. При появлении $s$ с уже зафиксированным $P_j$ формирование $F_i$ заканчивается. В цикле обработки фрагментов каждый $point[i]$ используется вместо индекса $p\_s[i]$ для выбора события текущего $F_i$.

Алгоритм2 представляет вариант1 алгоритмов второго класса. Вариант2 — это алгоритм2, в котором строка, помеченная (*) на рис. 7, заменена на $i := P_\alpha$ и нет первого присваивания $i$ перед циклом. Тогда, как и в алгоритме1, цикл обработки всегда начинается с трассы $S_\alpha$. С помощью этих вариантов можно построить трассы, соответствующие вычислению из рис. 3, *b*. Действительно, каждый из фрагментов таких трасс содержит два события $s$. Поэтому первыми обрабатываются последние $s$ из фрагментов всех трасс.

Можно привести другие алгоритмы этого класса, например алгоритмы, обрабатывающие $F_i$, у которых $\min < qu\_f[i] < \max$, или алгоритмы, допускающие расширяемые $F_i$, но для этого использующие дополнительные переменные размерности $N$ или даже $N \times N$, если не увеличивается число просмотров $S_i$.

3.4. *Оценка сложности алгоритмов.* Сложность алгоритмов зависит от числа процессов $N$ и числа событий $|E| = |S| + |R|$, причем $|S| = |R|$. Трассы $R_i$ всегда строятся в прямом направлении.

$S_i \in S$ в алгоритмах первого класса просматриваются в прямом направлении. Поэтому сложность можно оценить как $O(N \times |E|)$.

$S_i \in S$ в алгоритмах второго класса просматриваются в прямом и обратном направлениях, причем в зависимости от вариантов два и более раз. Поэтому сложность можно оценить как $O(N(kh + h))$, где $k \ge 2$, $h = |E|/2$.

Трассы $R_j \in R$, построенные одним алгоритмом, могут отличаться от $R_j \in R$, построенных другим, только порядком вхождения тех $r \in R_j$, которые при построении имеют $n < N$ трасс $S_i \in S$, содержащих $s$ с $s[pole1] = P_j$, доступных для отметки такого $r$. Поэтому, если имеется $k$ подобных $(r_1, ... ..., r_k) \in R_j$ с соответственно $n_1, ..., n_k$, доступными $s$, то общее число $\sigma_j$ различных $R_j$ равно $n_1 ... n_k$. При среднем $n$ $\sigma_j = n^k$, а общее число различных R равно $\sum = \sum_{j=1}^{N} \sigma_j$.

**4. Доказательство корректности.** Вначале докажем три свойства событий исходных трасс, следующие из алгоритмов обоих классов и предположения, что трассы получены при распределенном вычислении, которое является *полным* (см. разд. 1).

Напомним, что на каждой итерации $\gamma_i$, $i = 1, ...,$ внешнего цикла внутренний цикл может выполняться для любых $d$ трасс $S_i \in S$, где $1 \le d \le N$, и на каждой его итерации $i$ могут быть обработаны $m \ge 1$ последовательных событий $s \in S_i$.

```
CausalOrder2 (P_α, N, S = ∪_{i=1}^N S_i, R = ∪_{j=1}^N R_j)
 ∀k : 1 ≤ k ≤ N : p_s[k] := 1; p_r[k] := 1; all_e[k] := 1;
 mark[k] := 0; qu_f[k] := 0;
 repeat
   min := N − 1;
\* Цикл формирования F_i : 1 ≤ i ≤ N *\
   for i := 1 to N do
    if all_e[i] = 0 then go to 3
    if qu_f[i] ≠ 0 then go to 2
    ∀k : 1 ≤ k ≤ N : point[k] := 0;
1.  Выбор s из S_i по индексу p_s[i]
     if s[pole1] = 0 ∧ s[pole2] = all_e[i]
     then begin all_e[i] := 0; go to 2 end
     if s[pole2] = all_e[i]
     then begin j := s[pole1];
     if point[j] ≠ 0 then go to 2
       point[j] := 1; qu_f[i] := qu_f[i] + 1;
       p_s[i] := p_s[i] + 1;
       all_e[i] := all_e[i] + 1; go to 1
           end
     if s[pole2] > all_e[i] then begin
              if mark[i] = 0 then go to 2
              else begin            \* Обработка r_i *\
                 mark[i] := mark[i] − 1;
                 all_e[i] := all_e[i] + 1; go to 1
                  end
                                      end
2.  if qu_f[i] ≠ 0 ∧ qu_f[i] < min then min := qu_f[i]
3.  endfor
\* Цикл обработки F_i : 1 ≤ i ≤ N *\
 ∀k : 1 ≤ k ≤ N : point[k] := p_s[k] − 1;
   i := P_α − 1;
 for k := 1 to min do
   i := (imodN) + 1;                                          (*)
   for β := 1 to N do
    if qu_f[i] = 0 ∨ qu_f[i] > min then go to 4
    Выбор s из S_i по индексу point[i];
    r[pole1] := i; r[pole2] := s[pole2];
    j := s[pole1];
    Вставка r в R_j по индексу p_r[j];
    mark[j] := mark[j] + 1; p_r[j] := p_r[j] + 1;
    point[i] := point[i] − 1;
4.  i := (imodN) + 1;
   endfor
 endfor
    ∀i : 1 ≤ i ≤ N :
    if qu_f[i] = min then qu_f[i] := 0;
 until ∀δ : 1 ≤ δ ≤ N : all_e[δ] = 0
 endrepeat
 return (R_j : 1 ≤ j ≤ N)
end CausalOrder2
```

*Рис. 7.* Алгоритм2 построения причинно упорядоченных трасс $R_j : 1 \le j \le N$ событий приема

**Свойство 1. (Движение).** *После завершения каждой итерации внешнего цикла, если она не последняя, по меньшей мере, одно ожидаемое событие есть либо отмеченное r, либо s.*

**Доказательство** проведем индукцией по номеру итерации.

(i). *Базис.* По алгоритмам $S_i$ продвигается только при обработке $s$ или отмеченного $r$. Допустим, что при завершении $\gamma_1$ ни одно ожидаемое событие не является ни (a) отмеченным $r$, ни (b) $s$. Рассмотрим два случая.

Случай 1. $d = N \wedge m \geq 1$.

По алгоритмам $r \in S_j$ отмечается только при обработке $s \in S_i$, где $s[pole2] = all\_e[i] \wedge s[pole1] = P_j$. В исходном состоянии $\forall i\ all\_e[i] = 1$. Если при завершении $\gamma_1$ ни одно $r$ не было отмечено, то для первых $s$ во всех трассах $s[pole2] > 1$. Значит, все $P_i$ при вычислении зависли на ожидании сообщения, что противоречит предположению о полном вычислении. Назовем это *опровержением 1*. Следовательно, $\exists S_i : s \in S_i \wedge s[pole2] = 1$. Если $S_i$ просмотрена, то $s$ остается до следующей за $\gamma_1$ итерации, что уже опровергает допущение (b).

Пусть такая $S_i$ просматривается и является единственной. Тогда при обработке $s$ с $s[pole1] = P_j$ $r \in S_j$ отмечается. Следующим за $s$ может быть другое $s$, если все предшествующие ему неявные $r$ отмечены. Если это $s$ входит в число $m$ выбираемых событий $s$, то оно обрабатывается. Если же $m$ таких $s$ уже обработаны, то данное $s$ остается ожидаемым, что и опровергает допущение (b). Если же за $s$ следует неотмеченное $r$, то при $m = 1$ существует только одна трасса $S_j$ с отмеченным $r$. Здесь возможны две ситуации.

1.1. $S_j$ уже просмотрена. Тогда $r \in S_j$ остается до следующей за $\gamma_1$ итерации, что и опровергает допущение (a).

1.2. $S_j$ просматривается. По опровержению 1 следующим за отмеченным $r$ может быть только $s$, рассуждения для которого аналогичны предыдущим для $s \in S_i$.

Случай 2. $1 \leq d < N \wedge m \geq 1$.

Для каждой из $d$ трасс одной группы первым событием может быть неотмеченное $r$. Поскольку число $g$ таких групп конечное (например, $g = N/d$), то по опровержению 1, самое большее, на $\gamma_g$-й итерации выберется трасса $S_i$: $: s \in S_i \wedge s[pole2] = 1$. Значит, такое $s$ оставалось ожидаемым во всех итерациях перед $\gamma_g$, что и опровергает допущение (b). Таким образом, в любом из случаев после итерации $\gamma_1$ остается ожидаемым либо $s$, либо отмеченное $r$, что и доказывает базис индукции.

(ii). *Шаг индукции.* Допустим в качестве гипотезы индукции, что после итерации $\gamma_{(n-1)}$, по меньшей мере, одно ожидаемое событие есть или отмеченное $r$, или $s$. Покажем, что аналогичное событие будет и после завершения $\gamma_n$, если она не последняя.

Допустим, что после $\gamma_{(n-1)}$ осталось только одно отмеченное $r \in S_i$. Значит, все остальные трассы либо уже завершены, либо все или часть из них содержит ожидаемым неотмеченный $r$. По опровержению 1 после обработки $r \in S_i$ очередным событием в $S_i$ может быть только $s$. Если очередное $s$ есть признак конца трассы (т. е. $s[pole1] = 0$), то по алгоритмам оно остается до следующей за $\gamma_n$ итерации.

Если же $s[pole1] = P_j$, то $r \in S_j$ отмечается. Следующим за $s$ может быть другое $s$, если все предшествующие ему неявные $r$ отмечены. Если это $s$ входит в число $m$ выбираемых событий $s$, то оно обрабатывается. Если же $m$ таких $s$ уже обработаны, то данное $s$ остается до следующей за $\gamma_n$ итерации. Если же за $s$ следует неотмеченное $r$, то при $m = 1$ существует только одна трасса $S_j$ с отмеченным $r$. Здесь, как и в базисе, возможны две ситуации.

2.1. $S_j$ уже просмотрена. Тогда отмеченное $r \in S_j$ остается до следующей за $\gamma_n$ итерации.

2.2. $S_j$ просматривается. Тогда рассуждения для $r \in S_j$ аналогичны предыдущим для $r \in S_i$.

Теперь допустим, что после $\gamma_{(n-1)}$ итерации осталось только $s \in S_i$. Если остальные трассы уже завершились и $s$ есть событие признака конца, то по

алгоритмам оно остается до следующей за $\gamma_n$ итерации. Если же $s[pole1] = P_j$, то $S_j$ не может быть завершенной. Иначе исходное вычисление закончилось бы сообщением в транзите от $P_i$, что противоречит предположению о полном вычислении. Далее рассуждения по обработке $s$ аналогичны предыдущим для $s \in S_i$ после обработки $r \in S_i$.

**Свойство 2. (Недетерминизм).** *Если в $S_j$ ожидаемое событие есть неотмеченное $r$ и существуют $n \geq 1$ трасс, в которых ожидаемое событие есть $s$ с $s[pole1] = P_j$, то любое из таких $s$ может стать соответствующим $r$.*

**Доказательство.** Пусть в $S_j$ ожидаемое событие есть неотмеченное $r$. По свойству 1 найдется хотя бы в одной из трасс $S_i$ ($i \neq j$) соответствующее ему $s$, которое его отметит. Иначе трасса $S_j$ не будет продвинута. Если $S_i$ единственная, то при обработке $s$ событие $r$ станет отмеченным, т. е. будет включено в $R_j$, и $mark[j]$ увеличится. Если же $n > 1$, то $r$ может быть отмечено тем $s$, которое будет обработано ранее других. □

**Свойство 3. (Завершение).** *Итерация внешнего цикла будет последней, если и только если при ее завершении не останется ни одного ожидаемого события в трассах $S$.*

**Доказательство.** ⇒ Пусть итерация $\gamma$ является последней. По алгоритмам внешний цикл завершается, если $\forall i\ all\_e[i] = 0$. Допустим, имеется $S_x$ такая, что после завершения $\gamma$ $\forall i \neq x all\_e[i] = 0 \wedge all\_e[x] \neq 0$. Значит, в $S_x$ осталось ожидаемое событие. Допустим, что это событие есть $s$ с $s[pole1] = P_j$. Тогда исходное вычисление закончилось с одним сообщением в транзите от $P_x$ к $P_j$. Это противоречит предположению о полном вычислении. Теперь допустим, что это событие есть неотмеченное $r$. Тогда исходное вычисление зависло на ожидании приема сообщения в процессе с $r$, что также противоречит указанному предположению. Наконец, если это событие есть отмеченное $r$ или признак конца трассы $S_x$, то оно будет обработано на следующей за $\gamma$ итерации, что противоречит прямому утверждению, что $\gamma$ — последняя итерация.

⇐ Пусть при завершении $\gamma$ не осталось ни одного ожидаемого события в трассах $S$. Это значит, что во всех трассах выбрано событие признака конца трассы. Поэтому $\forall i\ all\_e[i] = 0$ и внешний цикл завершается. □

Докажем корректность алгоритмов на основе доказательства его безопасности и живучести.

Безопасность означает, что события $r$ в построенных трассах R соответствуют причинному порядку событий $s$.

**Утверждение 1.** $s < s' \wedge r \sim r' \Rightarrow r < r'$.

**Доказательство.**

(i) $s <_i s' \wedge r \sim r' \Rightarrow r < r'$.

События $s$ или фрагменты F* всегда обрабатываются последовательно. По свойству 3 для любого $s$ всегда найдется неотмеченное $r$. Поскольку $s <_i s'$, то $s$ будет обработано ранее $s'$. Поэтому, если $s[pole1] = s'$ $[pole1] = P_j$, то соответствующее $r$ будет помещено в $R_j$ перед $r'$, т. е. $r < r'$.

(ii) $s <_i s'' \wedge r'' <_x s' \wedge r \sim r' \Rightarrow r < r'$.

По свойству 1 $S_x$ может быть продвинута только после того, как $r''$ будет отмечено соответствующим ему $s$. По свойству 2 таким $s$ может стать или единственное, или самое первое из ожидаемых $s$ с $s[pole1] = P_x$. По условию (ii) соответствующим для $r''$ является $s''$. (Отметим, что именно в этом случае в исследуемом вычислении $s'$ будет причинно зависеть от $s$, так как по определению 1 $s < s'' < r'' < s' \Rightarrow s < s'$.) Поскольку $s[pole1] \neq s''$ $[pole1]$, то независимо от того, последовательно или во фрагменте обрабатываются $s$ и $s''$, $s$

---

* Напомним, что F содержит $s$ с различными значениями $s[pole1]$ и между $s \in F$ могут находиться только отмеченные $r$.

будет обработано раньше $s'$. Следовательно, если $s[pole1] = s'[pole1] = P_j$, то соответствующее $r$ будет помещено в $R_j$ перед $r'$, т. е. $r < r'$.

(iii) Обозначим через $s_{ik}$ событие $s \in S_{ik}$, а соответствующее ему $r$ в $S_j$ через $r_j^{ik}$.

Если в цепочке $s_{i1}, s_{i2}, \ldots, s_{in}$ $\forall k: 1 \le k < N$: каждая пара $(s_{ik}, s_{i(k+1)})$ удовлетворяет условию (ii), то $r_j^{i1} < r_j^{i2} < \ldots < r_j^{in}$.

Рассмотрим последнюю пару $s_{i(n-1)}, s_{in}$. Поскольку она удовлетворяет условию (ii), то $s_{i(n-1)} < s_{in}$. Следовательно, перед $s_{in} \in S_{in}$ должно находиться $r''$, которое может быть отмечено только при обработке соответствующего $s''$ из $S_{i(n-1)}$. Такое $s''$, в свою очередь, может быть обработано только после обработки аналогичного $r'''$, находящегося перед $s_{i(n-2)} \in S_{i(n-2)}$. Таким образом, продвигаясь по цепочке в направлении к $s_{i1}$, получаем первую пару. Согласно (ii), $r_j^{i1} < r_j^{i2}$ и обработка $s'' \in S_{i2}$ отметит $r'' \in S_{i3}$. Продвигаясь по цепочке к $s_{in}$, получаем $r_j^{i1} < r_j^{i2} < \ldots < r_j^{in}$. □

Живучесть означает, что для каждого события $s$ соответствующее ему событие $r$ будет помещено в результирующую трассу процесса-адресата и для каждого события $r$ найдется соответствующее ему событие $s$. Доказательство непосредственно следует из свойства 3.

**5. Особенности циклической отладки.** Построенные $R_i$ могут найти применение как в методах воспроизведения, так и в других методах организации проверки распределенного вычисления.

Традиционные методы воспроизведения [5, 6] для управления повторным вычислением обычно используют трассы $R_i$, полученные при первоначальном вычислении. При воспроизведении события $s$ в каждом из $P_i$ выполняются в порядке появления, а $r$ могут задержать вычисление $P_i$ до прибытия сообщения, у которого номер процесса-источника и порядковый номер соответствующего $s$ совпадают с информацией для очередного $r \in R_i$. Таким образом, воспроизводимое вычисление является детерминированным и эквивалентным первоначальному. Для его анализа используются различные методы циклической отладки [7, 1]. Однако этого может быть недостаточно для установления причины ошибки, поскольку используется лишь один из возможных вариантов первоначального выполнения.

Трассы $R_i \in R$, построенные приведенными алгоритмами, содержат $\sum = \sum_{i=1}^{N} \sigma_i$ потенциальных вариантов первоначального вычисления. Воспроизведение для каждого из $P_i$ может быть реализовано под совместным управлением $R_i$ и $S_i$. При появлении $r$ вычисление $P_i$ задерживается, как и в традиционных методах, до прибытия сообщения, соответствующего требованиям очередного $r \in R_i$.

При появлении $s$ производится сравнение с очередным $s \in S_i$. Если они совпадают, то совпадает и число выполненных перед ними $r$, хотя порядок $r$ и номера процессов-источников могут быть иными. Если воспроизводимое вычисление содержит только такие совпадающие $s$, то его можно назвать синтаксически эквивалентным первоначальному. Среди таких вычислений может быть найдено и эквивалентное первоначальному, если последнее было причинно упорядоченным.

Поскольку содержимое любого сообщения может влиять на локальную последовательность действий $P_i$, то текущее $s$ из воспроизводимого вычисления может не совпасть с очередным $s \in S_i$ или число $r$ перед таким $s$ будет отличаться от числа $r \in R_i$. Это может указывать либо на существование иного варианта первоначального вычисления, которое не отображено в S, либо на ошибку.

Наиболее близкой представленным алгоритмам можно назвать работу [2], в которой рассмотрено управляемое выполнение, допускающее эксперименты с различными порядками событий $r$ для одного контролируемого процесса. Остальные процессы выполняются без контроля. Построенные $R_i$ могут быть

полезны как для систематического формирования тестовых гипотез о порядке $r$ для такого управления, так и для контроля выполнения одного процесса при фиксированном выполнении остальных. Отметим также, что число $\sigma_i$ всевозможных перестановок $r$ для контроля вычисления одного процесса в [2] будет выше, чем аналогичное число трасс $R_i$, поскольку $R_i$ содержат только причинно упорядоченные $r$.

В работе [2] также предложен подход, при котором управляемое выполнение производится по контрольным точкам. Это позволяет снизить $\sigma_i = n^k$ до $\sigma_i = \sum_{p=1}^{c} n^k$, где $c$ есть число контрольных точек; $k_p$ — среднее число таких $r$ между двумя точками, для которых доступными для соответствия являются в среднем $n$ событий $s$, а $k = \sum_{p=1}^{c} k_p$. Планируется представить параллельные алгоритмы построения R, по аналогичным точкам.

Заключение. Представлены два алгоритма разных классов и кратко охарактеризованы возможные их варианты. Искомые трассы R строятся по трассам S, которые могут быть получены при вычислении как с протоколом причинно упорядоченной доставки, так и без него. Однако S могут быть получены и автономно, например, при локальном исследовании вычисления каждого из $P_i$. В таком случае устраняется эффект навязывания, присущий любой мониторизации распределенного вычисления.

Предложенные алгоритмы обеспечивают систематическое построение трасс причинно упорядоченных событий $r$. Определение причинного порядка событий является основой для многочисленных методов анализа и организации причинно упорядоченных вычислений [8—12]. Дальнейшая работа предполагает исследования применений таких $R_i$ в развитии методов управления сложностью параллелизма.

## СПИСОК ЛИТЕРАТУРЫ


1. Clémencon C., Fritscher J., Meehan M. J., Rühl R. An implementation of race detection and deterministic replay with MPI // Lecture Notes in Computer Science. 1995. N 966. P. 155.
2. Damodaran-Kamal S. K., Francioni J. M. Nondeterminacy: testing and debugging in message passing parallel programs // ACM SIGPLAN NOTICES. 1993. 28, N 12. P. 118.
3. Колосова Ю. И. Общая схема алгоритмов сохранения причинного порядка событий и ее использование // Автометрия. 1996. № 3. С. 59.
4. Lamport L. Time, clocks and the ordering of events in a distributed system // Commun. ACM. 1978. 21, N 7. P. 558.
5. Leblanc T. J., Mellor-Crummey J. M. Debugging parallel programs with instant replay // IEEE Trans. on Computers. 1987. C-36, N 4. P. 471.
6. Leu E., Schiper A., Zramdini A. Efficient execution replay technique for distributed memory architectures // Lecture Notes in Computer Science. 1991. N 487. P. 315.
7. Leu E., Schiper A. Execution replay: a mechanism for integrating a visualization tool with a symbolic debugger // Ibid. 1992. N 634. P. 55.
8. Mattern F., Fünfrocken. A non-blocking lightweight implementation of causal order message delivery // Ibid. 1994. N 938. P. 197.
9. Raynal M., Schiper A., Toueg S. The causal ordering abstraction and a simple way to implement it // Inf. Proc. Lett. 1991. 39, N 6. P. 343.
10. Raynal M., Singhal M. Capturing causality in distributed systems // Computer. 1996. N 2. P. 49.
11. Schiper A., Eggli J., Sandoz A. A new algorithm to implement causal ordering // Lecture Notes in Computer Science. 1989. N 392. P. 219.
12. Schwarz R., Mattern F. Detecting causal relationships in distributed computations: in search of the holy Grail // Distributed Computing. 1994. 7, N 3. P. 149.


*Поступила в редакцию 28 февраля 1997 г.*