Метод для согласованного выполнения семейства распределенных асинхронно взаимосвязанных транзакций

Данилов И. Г.

ООО “НИЦ супер-ЭВМ и нейрокомпьютеров”, Таганрог

3 апреля, 2014 г.

Подходы

Передача сообщений (MPI)

- **двусторонние** коммуникации;
- неявная синхронизация (“по готовности”) данных;
- барьерная синхронизация.

Подходы

Передача сообщений (MPI)

- **двусторонние** коммуникации;
- неявная синхронизация (“по готовности”) данных;
- барьерная синхронизация.

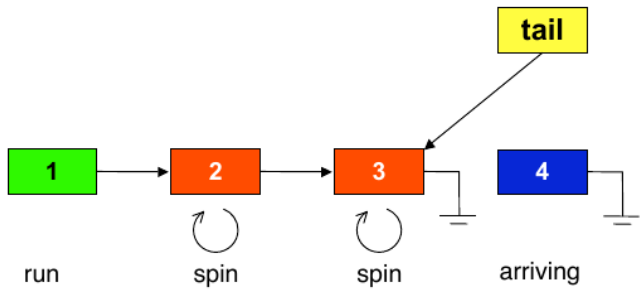
PGAS

- **односторонние** коммуникации (RDMA);
- определяемые программистом паттерны/правила распределения данных + операции редукции;
- общий случай распределения данных + распределённые блокировки/мьютексы;
- барьерная синхронизация.

UPC

```
1   upc_lock_t *list_lock =
2     upc_all_lock_alloc();
3   ...
4   void work()
5   {
6     upc_lock(list_lock);
7     // ...
8     upc_unlock(list_lock);
9   }
```

Распределенная версия MCS list-based queue lock algorithm[1]



[1] John M. Mellor-Crummey и Michael L. Scott. “Algorithms for Scalable Synchronization on Shared-memory Multiprocessors”. В: *ACM Trans. Comput. Syst.* 9.1 (февр. 1991), с. 21—65

Алгоритмы распределенного взаимного исключения

1. алгоритмы на **основе разрешений** (англ. *permission-based*) — процессы получают право на вход в КС в результате опроса и набора достаточного количества *разрешений/голосов* от других процессов;
2. алгоритмы использующие **токены** или *маркеры* (англ. *token-based*) — доступ к КС получает только процесс, владеющий *токеном*.

Проблемы

Время реакции — временной интервал между посылкой процессом сообщений для входа в КС и концом выполнения КС.

Нижняя оценка: $k \cdot (T + E)$

- в случае *грубо-гранулярной* (англ. *coarse-grained locking*) реализации блокировок — ухудшение масштабируемости$\downarrow$;
- в случае *мелко-гранулярной* (англ. *fine-grained locking*) реализации блокировок — улучшение масштабируемости$\uparrow$, но усложнение реализации$\downarrow$и повышение накладных расходов$\uparrow$на блокировки;

Альтернативные подходы

Алгоритмы планирования **транзакций** в БД, которые могут базироваться на следующих принципах:

- пессимистичные:
  - на принципе **ожидания**;
  - на принципе упорядочивания с использованием **временных меток** или **версий**;
- оптимистичные:
  - с использованием механизма **отката транзакций** (обычно на основе процедуры **валидации**).
- гибридные.

Цель алгоритмов планирования

Основная цель алгоритмов планирования транзакций — соблюдение заявленного **критерия согласованности**. Самым строгий критерием является **критерий сериализуемости**.

Определение

*Выполнение E' **сериализуемо** если оно вычислительно эквивалентно, т.е. производит тот же самый результат, некоторому последовательному выполнению E*[2].

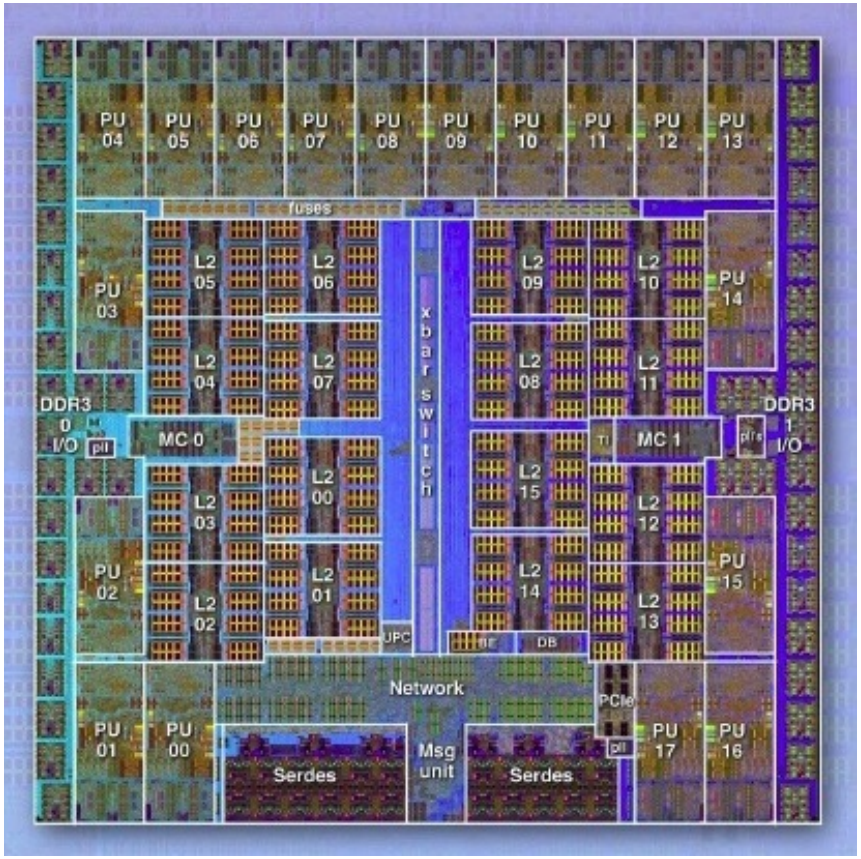
[2] Christos H. Papadimitriou. "The serializability of concurrent database updates". В: *J. ACM* 26.4 (1979), с. 631—653

Транзакции в вычислительных процессах

Подход, в котором используются принципы транзакций БД с целью конкурентного доступа вычислительных процессов к разделяемой памяти, называется **транзакционной памятью**[3].

[3] Maurice Herlihy и J. Eliot B. Moss. "Transactional memory: architectural support for lock-free data structures". В: *SIGARCH Comput. Archit. News* 21.2 (1993), с. 289—300

IBM Blue Gene/Q

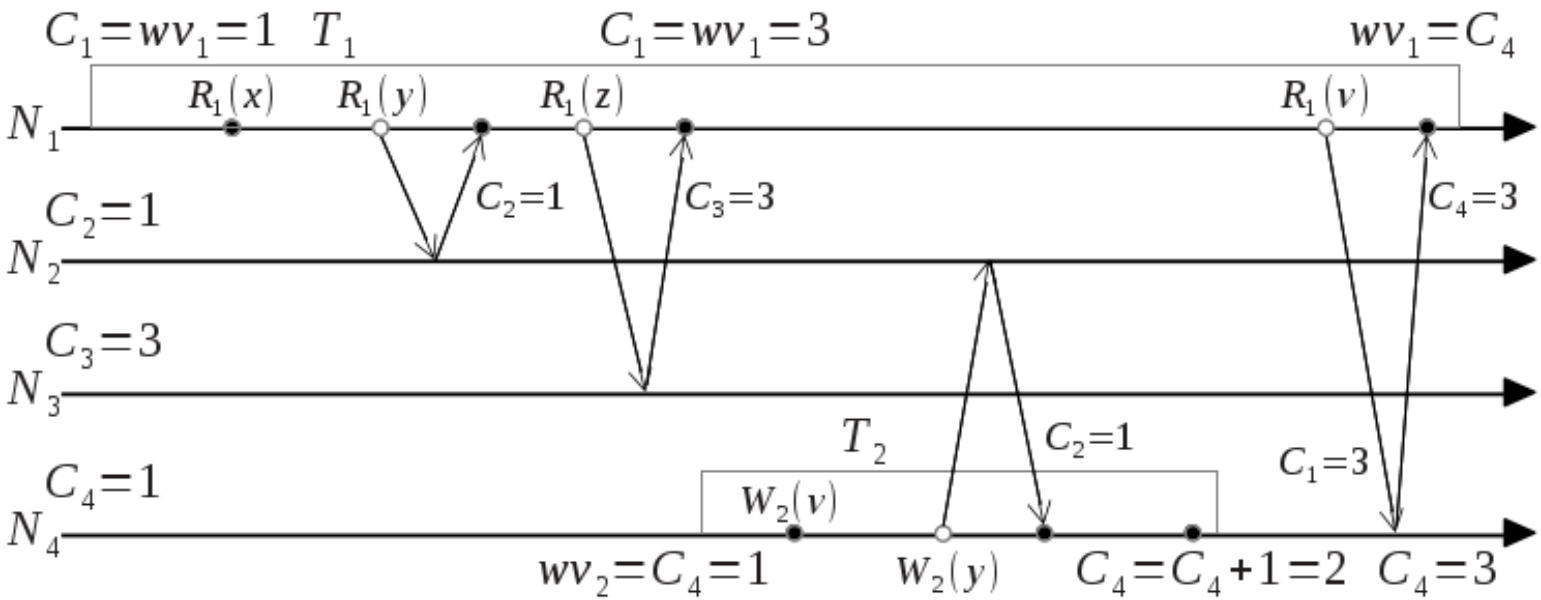


Транзакции памяти и БД

Критерий сериализуемости рассматривает только **зафиксированные** транзакции. БД выступает в роли “**песочницы**” для транзакций, которые могут выполнятся с аномалиями.

Для транзакций памяти это в общем случае недопустимо. Был предложен более строгий в плане корректности выполнения критерий **скрытности** (англ. *opacity*). Он является вариантом критерия сериализуемости и предполагает, что каждая транзакция в любой момент времени наблюдает **согласованное** состояние системы (невозможны гонки по данным).

Алгоритм TFA



Предлагаемый метод. Логические часы

1. если C_k — успешное событие фиксации транзакции $T_k \in \mathbb{T}$, выполняемой на узле системы n_m, то

$$CLK_m(C_k) \leftarrow CLK_m \leftarrow CLK_m + 1 \tag{1}$$

2. пусть e — событие чтения ячейки $x \in G$ транзакцией $T_k \in \mathbb{T}$, выполняемой на узле системы n_m; при этом если ячейке назначена версия $ts_x : TS(ts_x) = \langle j, tsn \rangle$, тогда после события e часы CLK_m устанавливаются в значение большее среди текущего значения часов CLK_m и отметки tsn версии ts_x:

$$CLK_m \leftarrow \max(CLK_m, tsn) \tag{2}$$

Предлагаемый метод. Локальные векторные часы

Определим для каждой транзакции T_k вектор VC_k так, что $j-$й элемент вектора равен $VC_k[j] = tsn$ — некоторой отметке времени логических часов узла с номером равным j.

Предлагаемый метод

- при старте транзакции ее вектор VC_k инициализируется нулевыми значениями;
- при чтении значения ячейки $x \in G$ с соответствующей версией $ts_x \in TSset : TS(ts_x) = \langle j, tsn \rangle$ проверяется, если $VC_k[j] < tsn$, то производится валидация объектов $Rset_k$ и присваивание $VC_k[j] \leftarrow tsn$, а также изменяются часы CLK_m;
- валидация производится для всех объектов x чьи версии сохранены ранее в $Rset_k$ путем сравнения текущей версии объекта ts'_x относительно сохраненной в $Rset_k$ версии ts_x: $RS_k(x) = ts_x$ и если $ts'_x > ts_x$, то валидация заканчивается *неуспешно*;

Предлагаемый метод. Продолжение

- при записи ячейки $x \in G$ производится определение ее текущей версии и при необходимости изменяются часы CLK_m, а новое значение v' сохраняется в буфере для последующей записи во время фиксации (отложенная стратегия обновления данных);
- в момент события C_k фиксации транзакции T_k, выполняемой на узле n_m, наращиваются часы CLK_m, а все объекты *атомарно* перезаписываются новым значением v' с новой версией равной $ts_x^k = \langle m, CLK_m(C_k) \rangle$.

Предлагаемый метод. Особенности

- соблюдение критерия скрытности;
- возможность реализации на основе метода алгоритмов транзакционной памяти с использованием **только односторонних операций**.

Вопросы?

Спасибо за внимание!

vainamon@gmail.com