Теория транзакций

Диспетчеры и протоколы

Уточнение структуры сервера данных

- Менеджер транзакций (EM) получает запросы от клиентов, ведет учет транзакций, генерирует begin transaction, commit, abort, передает запросы планировщику
- Планировщик (Scheduler): получает произвольное расписание на входе, выдает сериализуемое на выходе, передает операции менеджеру данных и возвращает результаты в TM.
- Менеджер данных асинхронно выполняет операции, полученные от планировщика.

JET BRAINS RESEARCH

Функционирование диспетчера

- Для получения сериализуемого выхода диспетчер может немедленно передать операцию DM, (временно) блокировать, или отклонить (в этом случае транзакция обрывается).
- Оптимистический планировщик (почти) никогда не блокирует операции, но обрывает транзакции в случае невозможности сохранить сериал изуемость.
- Пессимистический блокирует операцию, если ее немедленное выполнение может привести к нарушению сериализуемости.
- CSR-безопасность: $\mathrm{Gen}(\mathrm{S}) \subset \mathrm{CSR}$.

Оценка качества протоколов

- Измеряются на эталонных тестах (benchmarks)
  - TPC-C, TPC-E, . . .
- Пропускная способность (транзакций в секунду)
- Степень конкурентности (количество одновременно выполняемых транзакций)
- Частота обрывов (%)
- Время отклика может быть большим

JET BRAINS RESEARCH

Планировщики, использующие блокировки

Пессимистические, генерируют и используют дополнительные операции rl, wl для блокирования шагов транзакций.

Замки $\mathrm{pl}_i(x),\ \mathrm{ql}_j(y)$ находятся в конфликте, если $x = y,\ i \neq j$ и по $p = w \vee q = w$.

Запрос на установку замка, конфликтующего с уже имеющимся, приводит к блокированию операции установки замка.

Правила использования блокировок

LR1: Если t_i содержит $r_i(x)$ ($w_i(x)$), то планировщик выполняет $rl_i(x)$ ($wl_i(x)$) когда-либо до $r(w)$ и $ru_i(x)$ ($wu_i(x)$) когда-либо после нее.

LR2: На каждый элемент данных транзакция может получить не более чем один замок каждого типа.

LR3: Каждый замок должен быть снят один и только один раз.

JET BRAINS RESEARCH

Протокол двухфазного блокирования — 2PL

Для каждой транзакции t_i любая операция установки замка pl_i предшествует любой операции снятия замка qu_i.

Пусть $p, q \in CP(s)$, тогда:

$$pl_i(x) < p_i(x) < pu_i(x)$$

$$p_i(x), q_j(x) \text{ конфликтуют } \Longrightarrow pu_i(x) < ql_j(x) \vee qu_j(x) < pl_i(x)$$

$$p_i(x),\ q_i(y) \Longrightarrow pl_i(x) < qu_i(y)$$

Граф конфликтов 2PL

Пусть $G = G(CPIs))$ - граф конфликтов вывода 2PL-планировщика.

Если (t_i, t_j) - дуга в G, то $pu_i(x) < ql_j(x)$

Если $t_1, t_2, t_3, \ldots t_n$ - путь в G, то $pu_1(x) < ql_n(y)$

G не содержит контуров.

Корректность 2PL

$\mathrm{Gen}(2\mathrm{PL}) \subseteq \mathrm{CSR}$

$\mathrm{Gen}(2\mathrm{PL}) \subset \mathrm{OCSR}$

$w_i(x)r_2(x)r_3(y)r_2(z)w_1(y)c_3c_1c_2 \in \mathrm{OCSR}$,

но не может быть получен как выход $2\mathrm{PL}$.

JET BRAINS RESEARCH

Обработка тупиков

$r_1(x)w_2(y)w_2(x)w_1(y)$

Распознавание тупиков: граф ожиданий WFG.

Вершины - активные транзакции, дуга (t_i, t_j) означает, что t_i запрашивает замок, конфликтующий с замком, который установила t_j .

Контур в WFG необходим и достаточен для существования тупика.

Постоянные или периодические проверки WFG?

Разрешение тупиков

Необходимо оборвать одну из транзакций, попавших в контур WFG.

Стратегии: последнюю заблокированную, случайно выбранную, самую новую, по минимальному числу замков, максимальное число контуров

JET BRAINS RESEARCH

Предотвращение тупиков - timeout

Ограничения по времени ожидания: если время ожидания замка превысило timeout, ждущая транзакция обрывается.

Этот метод использовался во многих промышленных системах

Не требует проверки зависимостей между транзакциями, но может быть неэффективным из-за ложных обрывов транзакций.

JET BRAINS RESEARCH

Голодание - Livelocks

Ситуация, при которой некоторая транзакция становится жертвой разрешения тупика при каждом повторном запуске.

Для предотвращения можно учитывать первоначальную отметку времени поступления транзакции при выборе жертвы.

Предотвращение тупиков

- Возникновение тупика приводит к задержкам
- Все методы предотвращения тупиков приводят к ухудшению abort rate
- Полезны при наличии избыточных вычислительных мощностей и недопустимости задержек:
  - Системы БД реального времени
  - БД в оперативной памяти
  - Транзакционная память

JET BRAINS RESEARCH

Реализация предотвращения тупиков

Транзакция t_i запрашивает замок, несовместимый с замком, уже установленным t_j.

Wait-Die Если t_i стартовала раньше t_j, она ждет, иначе обрывается.

Wound-wait Если t_i стартовала раньше, t_j обрывается, иначе t_i ждет.

Immediate restart Перезапуск t_i.

Running priority Если t_j ждет другой замок, то t_j обрывается, иначе t_i ждет.

JET BRAINS RESEARCH

Варианты протокола 2PL

Консервативный $2\mathrm{PL}$ - все замки устанавливаются сразу. Применимо только для ограниченного класса приложений.

Строгий протокол двухфазного блокирования $(\mathrm{strict\ 2PL, S2PL})$ - все полученные замки на запись сохраняются до завершения транзакции. Строгие расписания важны для восстановления.

Сильный (strong) протокол $(\mathrm{SS2PL})$ - все замки удерживаются до завершения транзакции.

$\mathrm{Gen(SS2PL)} \subset \mathrm{Gen(S2PL)} \subset \mathrm{Gen(2PL)}$

$\mathrm{Gen(SS2PL)} \subset \mathrm{COCSR}$

JET BRAINS RESEARCH

Упорядоченное разделение замков

$w_1(x)r_2(x)r_3(y)c_3r_2(z)c_2w_1(y)c_1 \notin \mathrm{Gen}(2PL), \in \mathrm{OCSR}.$

Правила установки замков $pl_i(x) \longrightarrow ql_j(x)$:

OS1 Замок $pl_i(x) \longrightarrow ql_j(x)$: если pl_i получен раньше, чем ql_j, то должно быть $p_i < q_j$.

OS2 Если $pl_i(x) \longrightarrow ql_j(x)$: то t_j не может снимать никакие замки.

Правила определяют O2PL.

$\mathrm{Gen}(O2PL) = \mathrm{OCSR}.$

JET BRAINS RESEARCH

Недвухфазное блокирование

Предполагается, что элементы данных организованы в виде дерева.

Протокол с монопольным замками (на запись):

WTL1 Если x не корень, то $wl_i(x)$ может быть установлен, только если транзакция уже установила $wl_i(parent(x))$.

WTL2 Снятый замок не может быть установлен снова.

$Gen(WTL) \subset CSR$. Протокол свободен от тупиков.

Доказательство: Сопоставить граф конфликтов с деревом данных.

Неблокирующий диспетчер: Упорядочивание по временным меткам

Транзакции получают $ts(t)$ - временные метки.

Правило TO: $p_i(x)$, $q_j(x)$ - конфликт,

$$ts(t_i) < ts(t_j) \implies p_i(x) < q_j(x)$$

$$Gen(TO) \subset CSR$$

Доказательство. Для каждой дуги в графе сериализуемости $ts(t_i) < ts(t_j)$.

Реализация TO

Для каждого элемента данных нужно хранить max-r-ts(x) и max-w-ts(x).

Операция отклоняется, если она поступила слишком поздно (и транзакция обрывается).

Необходимо гарантировать, что обработчик данных будет выполнять конфликтующие операции в том же порядке, в котором их посылает диспетчер.

Протокол Snapshot Isolation (SI)

- Практика опередила теорию
- Для каждой транзакции определяются:
  - Интервал времени I(t)=[sts(t), cts(t)]
  - Write set W(t)
- Ограничение протокола:
  - $I(t1) \cap I(t2) = \varnothing \vee W(t1) \cap W(t2) = \varnothing$
- При наружении ограничения транзакция обрывается
- Чтение: по состоянию на момент начала транзакции

JET BRAINS RESEARCH

SI и сериализуемость

- Gen(SI) $\not\subseteq$ FSR

  $r_1(x)\, r_2(x)\, r_1(y)\, r_2(y)\, w_1(x)\, w_2(y)\, c_1\, c_2$

- CSR) $\not\subseteq$ Gen(SI)

  $r_1(x)\, w_1(x)\, r_2(x)\, w_2(x)\, r_1(y)\, w_1(y)\, c_1\, c_2$

- Gen(SI) префиксно замкнут
- Монотонность
- Строгость

JET BRAINS RESEARCH

Реализация SI на основе блокировок

- Блокировки устанавливаются только на запись
- При невозможности установить блокировку транзакция обрывается
- Чтение зафиксированных (committed) состояний

Аномалии, связанные с SI

- Несогласованная запись

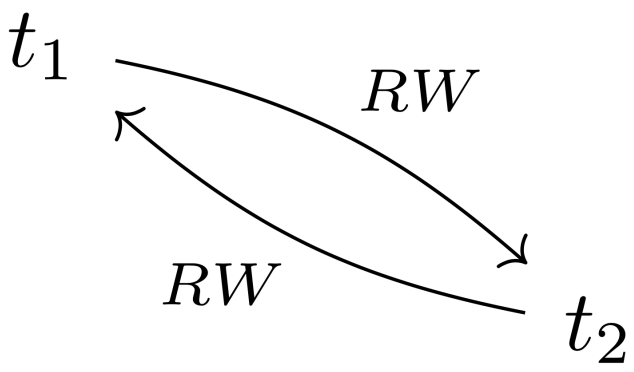
  $r_1(x)\, r_2(x)\, r_1(y)\, r_2(y)\, w_1(x)\, w_2(y)\, c_1\, c_2$

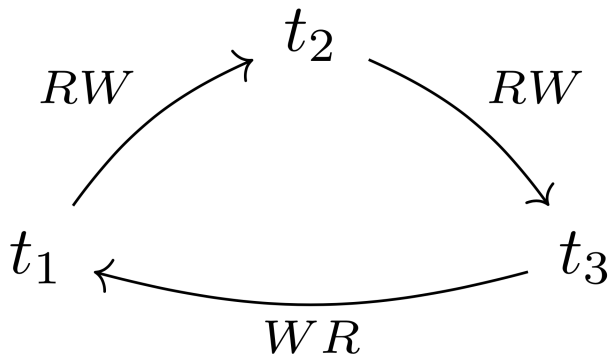
- Аномалия только читающей транзакции

  $r_1(x)\, r_2(x)\, r_2(y)\, w_2(x)\, c_2\, r_3(y)\, w_3(z)\, c_3\, r_1(z)\, c_1$

Serializable SI (SSI)

- Зависимости RW, WW, RW между транзакциями
- WW исключается протоколом SI
- Граф сериализуемости на основе зависимостей
- Запрещенные подграфы:





Неблокирующий протокол: Проверка SG

- При планировании каждой операции проверяется, не появился ли контур в графе сериализуемости.
- $\mathrm{Gen}(\mathrm{SGT}) = \mathrm{CSR}$.
- Необходима синхронизация с обработчиком данных, как и для TO.
- Проблема удаления завершенных транзакций из графа.

JET BRAINS RESEARCH

Оптимистическое управление конкурентным выполнением (OCC)

Фазы выполнения транзакции:

read: выполняются все операции над данными, но результаты операций записи хранятся в локальной памяти транзакции.

validation: По полученным RS(t) и WS(t) проверяется возможность сериализовать транзакцию

write: результаты операций записи записываются в базу данных.

Две последних фазы выполняются как непрерываемые.

OCC: проверка назад или вперед

BOCC: Проверяется совместимость конфликтов со всеми уже зафиксированными транзакциями. При несовместимости проверяемая транзакция обрывается.

FOCC: Проверяются конфликты с активными транзакциями. При конфликте обрывается другая (активная) транзакция.

Корректность OCC

Если в граф без контуров добавлять вершины таким образом, что из новых вершин дуги не выходят, то получившийся граф не содержит контуров

$$\mathrm{Gen}(\mathrm{BOCC}) \subset \mathrm{CSR}$$

$$\mathrm{Gen}(\mathrm{FOCC}) \subset \mathrm{CSR}$$

JET BRAINS RESEARCH

Конкурентность на основе множественных версий

В этом контексте версии абсолютно прозрачны вне базы данных.

Расписания с мультиверсиями

Сериализуемость для мультиверсий

Ограничение числа версий

Протоколы управления конкурентностью на основе мультиверсий

JET BRAINS RESEARCH

Расписания с мультиверсиями

Каждая операция записи создает версию, x_i - версия, записанная $w_i(x)$.

Функция версий: $h(r_i(x)) = w_j(x)$ - последняя запись, предшествовавшая чтению, $h(w_i(x)) = w_i(x)$.

Мультиверсионная история:

$m = (op(m), <_m),\ op(m) = h(\bigcup op(t_i))$

(1) $p, q \in op(t),\ p <_t q \Rightarrow h(p) <_m h(q)$

(2) $r_j(x) = r_j(x_i) \wedge c_j \in m \Rightarrow c_i \in m \wedge c_i <_m c_j$

Мультиверсионное расписание: префикс м-истории.

Монорасписания

Мультиверсионное расписание называется монорасписанием, если функция версий отображает каждую операцию чтения на последнюю предшествующую операцию записи.

$s = r_1(x)w_1(x)r_2(x)w_2(y)r_1(y)\ w_1(z)c_1c_2$

$m = r_1(x_0)w_1(x_1)r_2(x_1)\ w_2(y_2)r_1(y_2)\ w_1(z_1)c_1c_2$ – моноверсионное расписание.

$m = r_1(x_0)w_1(x_1)r_2(x_1)\ w_2(y_2)r_1(y_0)\ w_1(z_1)c_1c_2$ – не моноверсионное.

Операции над разными версиями x не находятся в конфликте.

Мультиверсионная эквивалентность по видимому остоянию

Отношение "читает из":

$\mathrm{RF(m)} = \{(\mathrm{t_i}, \mathrm{x}, \mathrm{t_j}) : \mathrm{r_j(x_i)} \in \mathrm{op(m)}\}$

Мультирасписания m, m’ $\vee$ -эквивалентны,

$$m \approx_v m' \Leftrightarrow RF(m) = RF(m')$$

Такая эквивалентность недостаточна для определения сериализуемости:

$m = w_0(x_0)w_0(y_0)c_0r_1(x_0)r_1(y_0)w_1(x_1)w_1(y_1)c_1r_2(x_0)r_2(y_1)c_2$

MV-сериализуемость

Мультиверсионная история m

MV-сериализуема ($m \in MVSR$), если она

V - эквивалентна серийной моноверсионной истории

$VSR \subset MVSR.$

$(\neq): m = w_0(x_0)w_0(y_0)c_0r_1(x_0)w_2(x_2)w_2(y_2)c_1r_1(x_0)c_1$

Конфликты в мультирасписаниях

Единственный вид конфликтов: $r_j(x_i)$, $w_i(x_i)$

$$m \approx_v m' \Longrightarrow G(m) = G(m')$$

Упорядочение версий: для любого x любое полное отношение полной упорядоченности $\ll$

MV-граф сериализуемости для m, $\ll$, $MVSG(m)$:
- граф конфликтов, пополненный дугами, представляющими отношение $\ll$.

$(m \in MVSR) \Leftrightarrow$ существует $\ll$ такое, что $MVSG(m)$ не содержит контуров.

MV-сериализуемость по коммутативности

MV-конфликт: $r_i(x_j)$, $w_k(x_k)$ такие, что $r_i(x_j) <_m w_k(x_k)$

Полностью упорядоченная мультиистория называется редуцируемой, если ее можно трансформировать в моноверсионную серийную историю перестановками соседних операций, не нарушающими MV-конфликты.

История m MV- сериализуема по конфликтам ($m \in MVSR$), если существует серийная моноверсионная история с теми же MV-конфликтами.

Редуцируемость $\Leftrightarrow MCSR$.

Характеристика MCSR

$\mathrm{MCSR} \subset \mathrm{MVSR}$

$(\neq)$: $r_2(y_0)r_3(z_0)w_3(x_3)\ c_3r_1(x_3)w_1(y_1)c_1w_2(x_2)c_2$

Граф MV-конфликтов: вершины=транзакции, дуги представляют MV-конфликты.

$m \in \mathrm{MCSR} \Leftrightarrow$ граф MV-конфликтов не содержит контуров.

Ограничение количества версий

$w_0(x_0)c_0r_1(x_0)w_3(x_3)c_3w_1(x_1)c_1r_2(x_1)w_2(x_2)c_2 \quad x_0 \ll x_1 \ll x_2 \ll x_3$

$VSR = 1VSR \subset 2VSR \subset \ldots \subset MVSR$

Мультиверсионный протокол с метками времени - MVTO

Операции чтения читают последнюю версию, метка которой меньше метки текущей транзакции.

Операция записи отвергается, если более поздняя транзакция уже прочитала более ранюю версию x, чем та, которая была бы записана.

Фиксация задерживается до фиксации всех транзакций с предшествующими метками, которые записывали данные, прочитанные данной транзакцией.

Многоверсионный SI

- Во всех реализациях SI используется 2V (зафиксированная и текущая в обновляющей транзакции)
- MVSI – использует метки времени фиксации версий
- Выполнение только читающих транзакций всегда возможно

Протокол MV2PL

Использование замков гарантирует, что существует только одна незафиксированная версия любого элемента данных.

Операция чтения читает последнюю зафиксированную версию.

Завершение транзакции задерживается до тех пор, пока не завершатся все транзакции, из которых она читала.

JET BRAINS RESEARCH

Протокол 2V2PL

rl устанавливаются всегда и не задерживают выполнение транзакции.

wl несовместим с другими замками на запись.

cl (certificate lock) устанавливается для всех элементов данных перед фиксацией. Этот замок несовместим с остальными.

Корректность: Сертификационные замки обеспечивают полное упорядочение версий.

Протокол MVSGT

Поздние транзакции: в графе есть путь из текущей

Ранние: есть путь в текущую.

Для операции чтения выбирается версия, не являющаяся ни поздней, ни ранней.

При планировании операции записи проверяется отсутствие контуров в графе.

JET BRAINS RESEARCH

MV-протокол для только читающих транзакций

- Обычный S2PL или SS2PL для пишущих транзакций, однако образуются новые версии.
- Для только читающих - MVTO.
- Удаление устаревших версий нетривиально.