РАСШИРЕНИЕ ПРОТОКОЛА TCP ДЛЯ СИНХРОНИЗАЦИИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

А.С. Игумнов

ВВЕДЕНИЕ

В данной работе показана возможность сохранения полностью синхронизированного состояния параллельной программы без изменения ее исходного кода. Реализация предполагает модификацию драйвера TCP.

В работе [1] рассматривалась технология сохранения/восстановления состояния параллельной программы в контрольных точках (КТ). Сохранение состояния TCP соединений предполагалось выполнять в контексте пользователя. Данные из сокетов TCP считывались во временные буферы в прикладной программе. Локальное состояние процесса сохранялось средствами программы BLCR - Berkeley Lab Checkpoint/Restart. К сожалению, дальнейшие исследования показали, что на уровне прикладной программы гарантированное сохранение состояния TCP-соединения невозможно.

Предлагаемый в данной статье подход основан на расширении протокола TCP и переносе кода, отвечающего за сохранение КТ, в драйвер TCP.

Для корректного сохранения КТ параллельной программы она должна отвечать следующим требованиям:

- все процессы параллельной программы запущены на узлах кластера с модифицированным драйвером TCP;
- параллельная программа не взаимодействует с другими группами процессов;
- сокеты не разделяются между несколькими процессами;
- граф TCP соединений является связным, т.е из любого процесса сообщение может быть доставлено в любой другой процесс, причем полузакрытые соединения считаются двунаправленными;
- локальное состояние каждого процесса может быть сохранено и в последующем восстановлено средствами BLCR;
- существуют средства, позволяющие получить "слепок" состояния файловой системы;
- выделяется один "управляющий" процесс, который инициирует сохранение КТ.

РАСШИРЕНИЕ ПРОТОКОЛА TCP

В RFC793 [2] описан набор состояний протокола TCP:

CLOSED, LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, LAST-ACK, TIME-WAIT

Предлагается дополнить этот набор еще тремя состояниями: CHECKPOINT-WAIT, PRECHECKPOINT и CHECKPOINTED.

Для управления новыми состояниями протокола можно использовать поле Options в заголовке сегмента TCP.

В соответствии с RFC793 [2] запись в поле Options имеет формат:

1. Номер опции - 1 байт
2. Длина в байтах - 1 байт
3. Данные

Номер опции для экспериментальных расширений определен в RFC4727 [3] и может быть выбран равным 253 или 254. Данные для предлагаемого расширения состоят из:

- Типа сообщения - 1 байт
- Числового поля - 2 байта
- Длина опции равна 5 байтам.
- Тип сообщения может принимать значения:
- checkpoint_request - инициация сохранения КТ;
- checkpoint_ack1 - подтверждение начала сохранения КТ;
- checkpoint_ack2 - подтверждение завершения сохранения КТ;
- checkpoint_failed - сообщение о невозможности сохранения КТ.

Числовое поле служит для передачи номера контрольной точки в сообщениях типа checkpoint_request, количества сохраненных процессов в сообщениях checkpoint_ack1 и checkpoint_ack2 или код ошибки в сообщениях checkpoint_ack2.

КРАТКОЕ ОПИСАНИЕ ПРОТОКОЛА СИНХРОНИЗАЦИИ

Предположим, что у нас есть два процесса, связанные через пару сокетов TCP, находящихся в состоянии ESTABLISHED. Первый процесс является инициатором сохранения КТ, второй находится в роли инициируемого. Будем обозначать сокет первого процесса S1, а сокет второго процесса - S2.

Первый процесс инициирует сохранение контрольной точки, переводя сокет S1 в состояние CHECKPOINT-WAIT.

При переходе в состояние CHECKPOINT-WAIT из S1 в S2 отправляется TCP сегмент C1.

В заголовке сегмента C1 заполнены следующие поля:

- seq=SEQ1 - последний подтвержденный байт в очереди на отправку S1->S2
- ack=ACK1 - последний полученный байт в очереди на прием S1<-S2
- win=0
- Options=checkpoint_request.

Пользовательские данные в этом сегменте не передаются.

По сути дела, это стандартный сегмент с дополнительной опцией.

В состоянии CHECKPOINT-WAIT сокет S1 на любые сегменты без опции checkpoint_ack1, checkpoint_ack2 или checkpoint_failed отвечает повторной отправкой C1.

Драйвер TCP, обслуживающий сокет S2, обнаружив сегмент с опцией checkpoint_request, переводит сокет S2 в состояние PRECHECKPOINT. Из очереди на отправку удаляются подтвержденные байты.

После перехода S2 в состояние PRECHECKPOINT из S2 в S1 отправляется сегмент C2.

В заголовке сегмента C2 заполнены следующие поля:

- seq=SEQ2 . Значение SEQ2 должно быть равно ACK1, так как в момент отправки сегмента C1 это был последний подтвержденный байт, а в состоянии CHECKPOINT-WAIT следующие байты полученные S1 не подтверждаются.
- ack=ACK2 . Значение ACK2 больше или равно SEQ1, так как S2 мог получить байты, для которых у S1 еще нет подтверждения.
- win=0
- Options=checkpoint_ack1.

В состоянии PRECHECKPOINT сокет S2 на любые сегменты отвечает сегментом C2.

Драйвер TCP начинает сохранение КТ процесса, к которому принадлежит сокет S2.

После сохранения КТ сокет S2 переводится в состояние CHECKPOINTED. В этом состоянии из S2 в S1 периодически отправляется сегмент C3.

В заголовке сегмента C3 заполнены следующие поля:

- seq=SEQ2.
- ack=ACK2.
- win=0
- Поле Options2 содержит значение checkpoint_ack2 (если сохранение КТ прошло успешно) или checkpoint_failed (если сохранение КТ не произошло).

В состоянии CHECKPOINTED в ответ на сегмент C1 или сегмент C2 повторно отправляется сегмент C2. При получении сегмента C3 или сегмента без опции синхронизации и seq >= ACK2, сокет переводится в состояние ESTABLISHED.

Драйвер TCP, обслуживающий сокет S1, получив сегмент C2, переводит сокет S1 в состояние PRECHECKPOINT и начинает сохранение КТ. Из очереди на отправку удаляются подтвержденные байты. После сохранения КТ сокет S1 переводится в состояние CHECKPOINTED.

Драйвер TCP, обслуживающий сокет S1, получив сегмент C3 или сегмента без опции синхронизации и seq >= ACK1 переводит сокет S1 в состояние ESTABLISHED.

Инициирующая сторона в сегменте C3 в поле Options всегда указывает значение checkpoint_ack2, независимо от успешности сохранения КТ.

В состоянии ESTABLISHED в ответ на сегмент с опциями checkpoint_ack1 или checkpoint_ack2 посылается подтверждение последнего принятого байта.

Таким образом, после получения сегмента C1 процесс, которому принадлежит сокет S2, знает свое собственное состояние и состояние сокета S1. После получения сегмента C2 полное состояние известно и первому процессу. Сегменты C3 необходимы для уведомления о том, что процесс синхронизации завершился с обеих сторон, и можно переводить TCP-сокет в нормальное состояние.

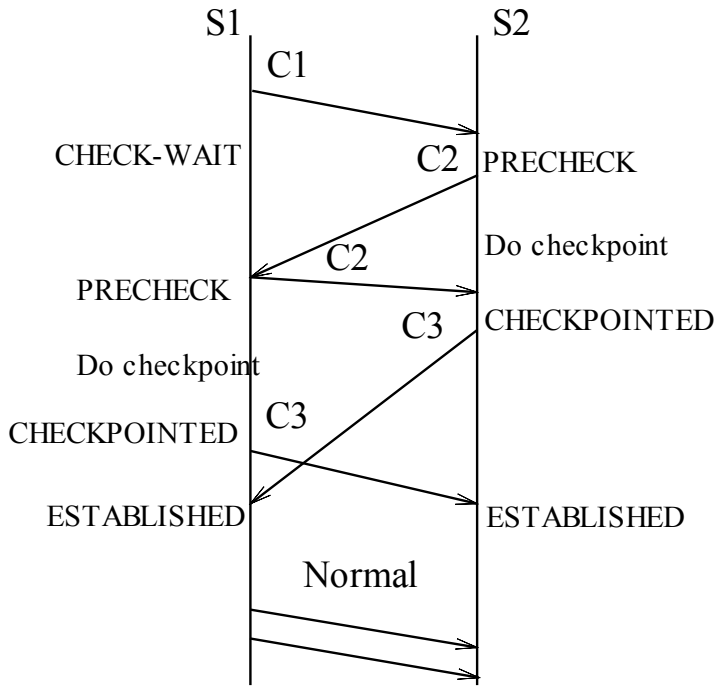

Рис.1 Временная диаграмма протокола синхронизации

ДОПУСТИМЫЕ СОСТОЯНИЯ ПРОТОКОЛА TCP

Инициирующий процесс не делает попыток синхронизации, если сокет находится в состояниях CLOSED,LISTEN,LAST-ACK,TIME-WAIT. В этих состояниях TCP-соединение либо еще не установлено, либо уже разорвано.

Если сокет находится в состояниях SYN-SENT,SYN-RECEIVED, то инициирующий процесс ожидает перехода сокета в состояние ESTABLISHED, после чего начинается синхронизация.

В состояниях FIN-WAIT-1,FIN-WAIT-2 синхронизация происходит так же, как и в состоянии ESTABLISHED. Это технически возможно, хотя с точки зрения прикладного процесса сокет в состояниях FIN-WAIT-1,FIN-WAIT-2 закрыт на запись.

ПОТЕРИ ПАКЕТОВ И ПОВТОРНЫЕ ПОСЫЛКИ

Сокет в состоянии CHECKPOINT-WAIT периодически отправляет сегмент C1. После заданного числа попыток сокет переводится в состояние ESTABLISHED. КТ не сохраняется. Данная ситуация возможна в нескольких случаях:

- обрыв канала связи или крах системы на втором компьютере;
- партнер не поддерживает расширение протокола TCP.

В первом случае, после перехода в состояние ESTABLISHED, связь будет разорвана стандартными таймерами TCP. Во втором передача данных будет возобновлена. Постоянная передача сегментов C1 не даст завершиться TCP-соединению по тайм ауту.

Сокет S2 в состоянии PRECHECKPOINT один раз отправляет сегмент C2 и не ждет его подтверждения. В случае потери сегмента C2 произойдет повторная передача сегмента C1 сокетом S1. Сокет S2 в ответ повторно отправит сегмент C2.

В состоянии CHECKPOINTED периодически отправляется сегмент C3. Время между повторами можно выбирать произвольно, например равным одной секунде. Количество повторов выбирается таким, чтобы суммарное время было достаточным для сохранения КТ.

При отсутствии ответа в этот промежуток времени можно сделать вывод о сбое сохранения КТ. В этом случае партнеру однократно отправляется сегмент с опцией checkpoint_failed.

В завершающей стадии синхронизации считается, что прием сегментов без опций синхронизации, означает успешную доставку партнеру сегмента C3. Это позволяет избежать бесконечного цикла взаимных подтверждений.

СИНХРОНИЗАЦИЯ БОЛЬШЕГО КОЛИЧЕСТВА ПРОЦЕССОВ

При необходимости синхронизации нескольких процессов используется простой алгоритм распространения информации о КТ (Рис. 2).

Управляющий процесс инициирует сохранение КТ с помощью системного вызова, переводящего все сокеты процесса в состояние CHECKPOINT-WAIT. При этом всем соседним процессам, связанным с ним через эти сокеты, отправляется сегмент C1.

Драйвер TCP на узле кластера, обнаружив сегмент C1, переводит сокет S, с которого он получен, в состояние PRECHECKPOINT, после чего начинает выступать в роли инициатора. В таблице процессов ищется процесс, которому принадлежит сокет S. Все остальные сокеты, принадлежащие тому же процессу, переводятся в состояние CHECKPOINT-WAIT с рассылкой сегмента C1 далее по графу соединений.

Сохранение КТ производится в тот момент, когда все инициируемые соседи сообщили о начале сохранения КТ отправкой сегмента C2. Если хотя бы один из соседей сообщил об ошибке сохранения, то сохранение КТ не производится; инициатору сохранения отправляется сегмент с опцией checkpoint_failed.

Инициирующий процесс переходит в состояние CHECKPOINTED в тот момент, когда завершено локальное сохранение КТ и все инициируемые соседи сообщили о сохранении КТ отправкой сегмента C3.

При распространении информации о КТ возможны коллизии, когда два партнера одновременно передают сегмент C1. В этом случае производится процедура выбора на основе IP адреса и номера порта. При коллизии сокет переводится из состояния CHECKPOINT-WAIT в состояние PRECHECKPOINT, если у него меньший IP или меньший номер порта (в случае равенства IP). В знак подтверждения «младший» сокет отправляет сегмент C2.

Номера контрольных точек, передаваемые в сегменте C1, не используются в протоколе синхронизации. Они вводят единую систему именования КТ в распределенной системе. Динамически создаваемые процессы смогут сохранять КТ без знания о ранее сохраненных КТ.

В сегмент C3, отправляемый инициатору в числовое поле опции checkpoint_ack2, может включаться сумма соответствующих значений, полученных в сегментах C3 от инициируемых, плюс 1. Вычисленная сумма позволит управляющей машине получить общее количество процессов, на которых произошло сохранение КТ. В случае сбоя в сохранении КТ, в числовом поле опции checkpoint_failed передается код ошибки.

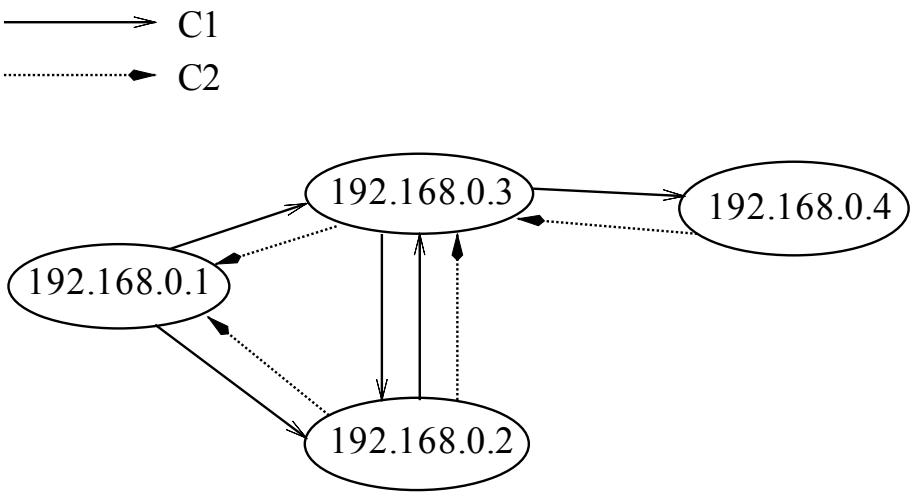

Рис. 2 Распространение информации о КТ

ОЦЕНКА ВРЕМЕНИ СОХРАНЕНИЯ

Благодаря сегменту C2 инициатор получает необходимую информацию до начала реального сохранения КТ. Соответственно, время распространения информации о сохранении КТ достаточно мало. Оно сравнимо со скоростью передачи сегмента C1 по цепочке от инициатора до самого «дальнего» процесса. После распространения информации о КТ происходит параллельное сохранение КТ на всех узлах. Общее время сохранения будет определяться самым медленным узлом.

ЛИТЕРАТУРА:

1. А.С.Игумнов, А.В.Рычков, Сохранение и восстановление состояния параллельной программы, использующей TCP сокеты. Труды всероссийской научной конференции "Научный сервис в сети Интернет: технологии параллельного программирования",с.82-84 издательство Московского университета
2. Information Sciences Institute University of Southern California. RFC793 TRANSMISSION CONTROL PROTOCOL (http://tools.ietf.org/html/rfc793)
3. B. Fenner. RFC4727 Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers (http://tools.ietf.org/html/rfc4727)