Санкт-Петербургский государственный университет

Кафедра системного программирования

Лавров Владимир Александрович

Реализация компилятора для сетевого вычислителя

Курсовая работа

Научный руководитель:
ст. преп. Баклановский М. В.

Санкт-Петербург
2019

Оглавление

Введение

В большинстве современных приложений используется архитектура “клиент-сервер”. Такое решение имеет ряд преимуществ по сравнению с пиринговыми сетями. В клиент-серверной архитектуре не возникает проблем с обеспечением согласованности данных, организацией системы хранения, так как не нужно обеспечивать синхронизацию. Отсутствует дублирование кода программы-сервера программами-клиентами. Не возникает проблем с системой безопасности, так как все данные хранятся на сервере. Снижаются требования к компьютерам-клиентам и так далее. Но у такой архитектуры есть существенный недостаток — плохая масштабируемость. Каким бы мощным ни был сервер, у него все равно будет ограниченное количество подключений. Увеличение этого параметра влечет за собой высокую стоимость оборудования, а иногда и вовсе недостижимо. Например, в отечественных ERP-системах наблюдается резкое ухудшение производительности при подключении тысячи устройств [8]. Этого недостатка лишены пиринговые (децентрализованные) сети.

Пиринговые сети основаны на равноправии участников. В децентрализованной сети нет выделенных серверов, то есть нет привязки к их мощности. В современном мире суммарная мощность компьютеров, для которых выделяется какой-либо сервер, сравнима с мощностью этого сервера, так что, с этой точки зрения, проблемы переноса вычислений с сервера на эти устройства не возникает. Но в таких сетях возникает проблема синхронизации. Ее можно частично решить, поставив дополнительные сервера для координации работы системы. В итоге задача синхронизации всех устройств сводится к задаче синхронизации серверов. Эти сервера не создают угрозу масштабируемости, так как не будут выполнять функций сервера из клиент-серверной архитектуры. Таким образом возникает понятие гибридных сетей. Такие сети сочетают скорость централизованных сетей и масштабируемость децентрализованных.

Все перечисленные системы работают на стандартных протоколах

транспортного уровня модели OSI: TCP и UDP, которые, в свою очередь, базируются на протоколе сетевого уровня IP, который обеспечивает маршрутизацию. Эти протоколы работают по принципу точка-точка (один отправитель, один получатель). То есть их задача — переслать пакет из точки A в точку B. Такой подход удобен для клиент-серверной архитектуры, так как любое действие в ней сводится либо к посылке запроса на сервер, либо получению ответа от него. Но в децентрализованных сетях схема взаимодействия может отличаться от схемы "запрос-ответ". Рассмотрим пример такого взаимодействия. Пусть в распределенной сети программа должна пройти маршрут из нескольких компьютеров (не обязательно линейный). Такое взаимодействие часто встречается в ERP-системах, когда несколько человек должны подписать некоторый документ в определенном порядке. При использовании протоколов TCP и UDP вся логика движения программы (маршрут) реализуется на прикладном уровне модели OSI. Но не будет ли лучше, если переложить эту логику на транспортный или сетевой уровень? Ведь именно там решаются проблемы маршрутизации и контроля доставки. За счет этого может повыситься скорость, надежность и устойчивость таких систем.

Сетевой вычислитель — проект, в котором предлагается переместить информацию, связанную с маршрутами программ, ниже по сетевому стеку, и предоставить пользователю интерфейс для создания и редактирования этих маршрутов. Компилятор в этой системе отвечает за преобразование маршрутов и связанных с ними данных из пользовательского представления в конфигурации для устройств сети. В данной работе будет представлена реализация компилятора для тестовой версии протокола многих точек.

1. Постановка задач

Цель данной работы — реализация компилятора для тестовой версии протокола многих точек. Для ее достижения необходимо выполнить следующие задачи:

- Изучить предметную область
- Спроектировать протокол
- Реализовать компилятор для тестовой версии протокола
- Протестировать полученное решение

2. Описание предметной области

2.1. Сетевой вычислитель

Сетевой вычислитель — проект, в котором будет реализована новая концепция организации последовательных вычислений в распределенной среде. По сути мы отказываемся от сервера (P2P) и добавляем возможность прохождения программами определенных маршрутов. Вся логика, связанная с маршрутами, хранится на устройствах. Маршруты составляются про помощи некоторого графического интерфейса, который генерирует промежуточный файл в определенном формате. Этот файл поступает на вход компилятору, который выдает конфигурацию для устройств, на которых будет исполняться программа. Эта конфигурация загружается на устройства при помощи нашей программы-загрузчика. После этого программа может исполняться.

Каким образом исполняется программа? На каждом устройстве есть программы на прикладном уровне модели OSI, назовем их обработчиками. Сеть используется для организации движения потока данных между этими программами на разных устройствах. Узел, с которого начинается маршрут, инициирует выполнение программы, запуская первый обработчик, который отправляет в сокет данные, необходимые следующему по маршруту узлу. Обработчик на следующем узле получает данные из сокета, обрабатывает их и отправляет назад в сокет, ничего не зная о маршруте. Таким образом, исходная задача программиста, реализующего систему, содержащую программы, исполняющиеся по определенным маршрутам, сводится к составлению маршрута и описанию способа обработки данных на каждом узле, то есть написанию обработчиков. Для увеличения скорости данные можно обрабатывать в пространстве ядра, не выгружая их через сокет в пользовательский уровень.

2.2. Стек протоколов сетевого вычислителя

На данный момент в системах, в которых используются рассматриваемые нами программы, вся логика, связанная с маршрутами, описывается на прикладном уровне модели OSI. Но для повышения эффективности работы таких систем предлагается опустить эту логику ниже по сетевому стеку. На каких уровнях внедрять протоколы? Рассмотрим следующие варианты:

- Реализовать протоколы сетевого и транспортного уровней

  Можно начинать работу на сетевом уровне и хранить информацию о маршрутах там. Это решение выглядит наиболее логичным, так как именно сетевой уровень отвечает за маршрутизацию, но оно сложно в реализации.

- Использовать ipv4 (поле options) / ipv6 (расширения)

  В протоколе ipv4 уже есть возможность задания пакету определенного маршрута, то есть множества хостов, которые должны быть пройдены перед доставкой этого пакета в пункт назначения. Этот вариант не подходит, так как планируется поддерживать возможность задания нелинейных маршрутов.

  Также в спецификации протокола ipv4 [4] есть поле options, которое можно использовать для хранения информации о маршрутах.

  В спецификации протокола ipv6 [6] предусмотрены расширения, предназначенные для внедрения дополнительной функциональности. Их можно использовать для хранения информации о маршрутах.

- Реализовать свой протокол транспортного уровня и использовать ip как базовый протокол

Для тестовой реализации системы был выбран вариант с началом работы на транспортном уровне как наиболее простой в реализации. В

дальнейшем планируется опустить протокол до сетевого уровня и реализовывать функциональность ip, либо использовать расширения ipv6 для увеличения скорости работы и соответствия логике сетевого стека.

2.3. Сравнение протоколов транспортного уровня

Итак, реализуем протокол транспортного уровня. Рассмотрим аналоги.

- UDP протокол [3]

  UDP — промежуточный протокол между IP и прикладным уровнем. С помощью него реализуется интерфейс сокетов. Он не реализует никакую дополнительную функциональность и используется там, где важна скорость работы и не важен надежный сервис (датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа). UDP подразумевает, что проверка ошибок и их исправление либо не нужны, либо должны исполняться в приложении.

- TCP протокол [5]

  TCP, в отличие от UDP, обеспечивает надежную доставку сообщений, но при этом сильно замедляет ее. Этот протокол используется, когда важен надежный сервис (побайтовое совпадение исходного и переданного сообщений).

- SCTP протокол [7]

  SCTP протокол работает аналогично TCP. Будучи более новым протоколом, SCTP имеет несколько нововведений, таких как многопоточность, защита от DDoS атак, синхронное соединение между двумя хостами по двум и более независимым физическим каналам (multi-homing) [1].

Таблица 1: Сравнение протоколов транспортного уровня

Параметр	UDP	TCP	SCTP
Установка соединения	нет	да	да
Надежная передача	нет	да	да
Упорядоченная доставка	нет	да	да
Контрольная сумма данных	да	да	да
Поддержка многопоточности	нет	нет	да

2.4. Протокол сетевого вычислителя

Какими параметрами должен обладать протокол транспортного уровня сетевого вычислителя?

- Установка соединения

  Установка соединения в протоколе многих точек не нужна, так как соединение придется устанавливать между всеми парами узлов на пути следования пакетов (маршрут может быть очень длинным), что приведет к избыточному захвату ресурсов.

- Надежная передача

  Контроль доставки сообщений — задача транспортного уровня. Протокол сетевого вычислителя работает с данными, поступающими на вход обработчикам. Передача обработчику некорректных данных повлечет сбой выполнения всей программы, либо сильно усложнит написание обработчика пользователем. Следовательно надежная доставка сообщений нужна. Но вариант TCP не подходит, так как контроль доставки будет осуществляться на каждом этапе исполнения, то есть между каждыми двумя устройствами, что может сильно замедлить работу. К тому же протокол сетевого вычислителя не устанавливает соединение, как это делает TCP. Решение — контроль доставки на основе контрольных точек. Помечаем некоторые (определенные) узлы флагом isCheckpoint. Подтверждение доставки будет отправляться только между двумя соседними узлами, помеченными флагом isCheckpoint. Задача компилятора установить эти флаги оптимальным образом.

Узлы	1	2	3	4	5
isCheckpoint	нет	да	нет	да	нет

- Упорядоченная доставка

  Упорядоченная доставка в протоколе сетевого вычислителя нужна, так как он работает с данными, поступающими на вход обработчикам. Нарушение порядка следования пакетов приведет к описанным в предыдущем пункте проблемам.

- Контрольная сумма данных

  Добавление контрольной суммы данных — обычная практика, позволяющая повысить надежность доставки. В протоколе сетевого вычислителя, как и в большинстве других сетевых протоколов, она должна быть.

- Поддержка многопоточности

  Из трех рассмотренных в таблице 1 протоколов многопоточность поддерживает только SCTP. В протоколе сетевого вычислителя многопоточность, как она реализована в SCTP, не имеет смысла. Про многопоточность в смысле программы сетевого вычислителя будет рассказано в главе 3.

3. Описание решения

Для начала рассмотрим разницу между понятиями программа и процесс в сетевом вычислителе. Программа — маршрут выполнения со всей дополнительной информацией (например, набор контрольных точек на этом маршруте) и программами обработчиками. Процесс — экземпляр программы во время ее выполнения. Каждый процесс имеет свой уникальный идентификатор, который находится в заголовке сообщения. В рамках одного процесса одновременно могут посылаться несколько сообщений. Например, какому-то узлу на пути следования сообщения потребовались данные из системы хранения. В этом случае формируется новое сообщение в рамках одного процесса (с уникальным идентификатором этого процесса), которое запрашивает данные у системы хранения, которая, в свою очередь, отправляет их сообщениями этого же процесса. Здесь мы можем выделить понятие потока исполнения. Поток — сообщение, которое обрабатывается или пересылается в рамках одного процесса, со всеми необходимыми ему ресурсами (например, checkpoint может сохранять данные сообщения, чтобы заново отправить его в случае потери).

3.1. Компилятор

Задача компилятора — преобразование данных, полученных от пользователя при помощи, например, графического интерфейса, в программу сетевого вычислителя. Как уже упоминалось ранее, маршруты программ могут быть нелинейными. В сетевом вычислителе поддерживается возможность ветвления маршрута, а также циклы. Ветвление реализуется с помощью анализа возвращаемого обработчиком результата. Таким образом, в общем случае программа сетевого вычислителя — граф, вершины которого — узлы сети. На вход компилятору подается сериализация этого графа. Рассмотрим подробнее формат входных данных.

3.1.1. Формат входных данных

Программа, предоставляющая пользователю возможность составления маршрутов при помощи графического интерфейса, генерирует файл в формате txt. Альтернативой этому формату может служить формат xml [2], который намного удобнее для чтения, но этот файл в любом случае генерируется автоматически, а с подробной спецификацией разницы в работе с ними нет, поэтому был выбран формат файла txt.

Формат данных:

program id	number of hosts

- program id

  Идентификатор программы сетевого вычислителя.

- number of hosts

  Количество устройств, по которым проходит маршрут.

host id	state	handler id
...	...	...

- host id

  Идентификатор устройства. Пользователь составляет маршрут из id устройств. По id можно получить ip адрес устройства с помощью таблицы разрешения id в ip.

- state

  Состояние программы сетевого вычислителя, то есть этап ее выполнения. На каждом следующем устройстве этап выполнения увеличивается на 1.

- handler id

  Идентификатор обработчика, который нужно вызвать на устройстве host на этапе выполнения state. На данный момент все обработчики загружаются на устройства вручную.

host id	state1	return value	state2	args
...	...	...	...	...

- host id

  Идентификатор устройства, на котором запускается обработчик на этапе выполнения, указанном в state1.

- state1

  Состояние программы сетевого вычислителя, то есть этап ее выполнения. На каждом следующем устройстве этап выполнения увеличивается на 1.

- return value/state2

  state2 — состояние, в которое нужно перейти программе, если обработчик с идентификатором handler id на этапе выполнения state вернул return value. Строк с одинаковым state1 может быть много. Таким образом реализуется ветвление.

- args

  Дополнительные параметры:
  -checkpoint — индикатор того, что state1 является контрольной точкой.
  -data filename hostid state — указание компилятору, что на этапе state файл с именем filename должен быть доставлен на устройство hostid.

3.1.2. Хранение маршрутов на устройствах

Все устройства хранят информацию о программах в следующем формате:

prog id	state 1	state 2	state 3	...
...	struct stateInfo	struct stateInfo	struct stateInfo	...
...	...	...	...	...

- prog id

  Идентификатор программы сетевого вычислителя.

- state

  Состояние программы сетевого вычислителя, то есть этап ее выполнения.

- struct stateInfo

  Структура, которая содержит:

  - Идентификатор обработчика, запускаемого на этом этапе выполнения
  - Массив соответствия возвращаемого обработчиком результата и состояния, в которое нужно перейти, в случае возврата обработчиком этого значения
  - Индикатор того, что устройство на этом этапе выполнения — checkpoint
  - Структуру, содержащую информацию для запросов к системе хранения данных (filename hostid state, аналогично дополнительному параметру из args в пункте 3.1.1)

3.1.3. Формат выходных данных

Выходные данные компилятора — конфигурации для устройств, которые будут рассылаться по ним при помощи программы загрузчика. Очевидно, что нам не нужно хранить всю таблицу с информацией о программах 3.1.2 на каждом устройстве. Для каждого устройства оставляем в этой таблице только то, что нужно ему, то есть генерируем таблицу с набором только тех состояний (state), на которых идет обработка данных на этом устройстве. Получаем:

state	сериализация stateInfo
...	...

Соответственно на вход загрузчику подаем идентификатор устройства, на которое нужно отправить конфигурацию, идентификатор добавляемой программы и таблицу, содержащую state и соответствующий ему stateInfo.

3.2. Загрузчик

Задача загрузчика — разослать конфигурации, полученные от компилятора, на устройства в сети. Рассылка происходит с помощью протокола сетевого вычислителя, так как компилятор работает с идентификаторами устройств, а посылать нужно по ip адресу (доступ к таблице разрешения hostid в ip адрес есть только у загруженного в ядро ос модуля). После того как сообщение с конфигурацией пришло на нужное устройство, нужно распарсить его и модифицировать таблицу, хранящую информацию о маршрутах, добавив туда новый полученный маршрут.

4. Особенности реализации

На данный момент реализован конфигуратор, позволяющий загружать все необходимые для работы данные на узлы, не проводя никаких оптимизаций. Обработчики загружаются на устройства вручную.

Программа написана под Linux, так как тестовая версия протокола многих точек реализована в Linux. В качестве языка программирования для этой работы подходит быстрый высокоуровневый язык, в котором реализован интерфейс для работы с сокетами, поэтому был выбран язык C++. Для загрузчика, соответственно, язык C, так как загрузчик — часть модуля ядра.

4.1. Преобразование таблицы маршрутов

Каким образом преобразовать таблицу 3.1.2 в конфигурации для разных устройств? В произвольном случае программа сетевого вычислителя — граф. Каждому устройству необходимо отправить только нужные ему данные о маршруте. Эта задача решается обычным обходом графа, например, в глубину. На устройство с идентификатором hostid (если на каком-то state встретился выбранный hostid) отправляем:

- stateInfo, соответствующий этому state
- для всех состояний, следующих за state (достижимых из state), идентификаторы устройств, на которых выполняется программа в этих состояниях

4.2. Расстановка контрольных точек

Расстановка флагов checkpoint на устройства — тоже задача компилятора. Каким образом расставить контрольные точки, ничего не зная о маршруте, кроме его структуры? На данный момент реализовано следующее: в графе состояний выделяются компоненты сильной связности.

- Если в компонентах больше одной вершины, контрольной точкой помечаются входные и выходные вершины этих компонент
- Если есть участок из четырех последовательных компонент, состоящих из одной вершины, то только одна из них помечается контрольной точкой

В дальнейшем планируется ввести набор интерфейсов для создания программ сетевого вычислителя, что позволит намного лучше ориентироваться в этих программах и, соответственно, расставлять контрольные точки более обдуманно. Например, компилятор будет явно знать, что на каком-то этапе идет работа с человеком. Это означает, что этот этап нужно отметить контрольной точкой.

5. Тестирование

Во всех тестах маршруты состояли из 2000 узлов. Каждый тест включал в себя прохождение 100 маршрутов. Для тестирования были выбраны следующие сценарии:

- Сравнение скорости работы протокола многих точек (без контроля доставки) с UDP протоколом на одном маршруте на локальной машине

  Этот тест направлен на то, чтобы оценить выгоду от перемещения логики (маршрутов) с прикладного на транспортный уровень.

Характеристика выборки	Протокол сет. выч.	UDP протокол
Минимальное время, мс	<1	1
Максимальное время, мс	4.5	5.8
Среднее время, мс	1.9	1.9
Дисперсия, мс^2	0.3	0.3

  Протокол сетевого вычислителя показывает схожие с UDP протоколом результаты по скорости работы на задачах без осуществления контроля доставки.

- Сравнение скорости работы протокола многих точек (с контролем доставки на основе контрольных точек) с TCP протоколом на одном маршруте на локальной машине

  TCP протокол обеспечивает надежную доставку данных с подтверждением на каждом узле. Здесь ожидается существенное различие в скорости работы. Тестирование производится на локальной машине для того, чтобы исключить влияние среды на результаты.

Характеристика выборки	Протокол сет. выч.	TCP протокол
Минимальное время, мс	2.1	826
Максимальное время, мс	6.1	1013
Среднее время, мс	3.3	890
Дисперсия, мс^2	3.2	1814

  Протокол сетевого вычислителя на специфических ему задачах существенно быстрее, чем TCP.

6. Заключение

В ходе работы были достигнуты следующие результаты:

- Проведено исследование предметной области
- Написана спецификация для протокола многих точек
- Реализованы компилятор и загрузчик для тестовой версии протокола
- Полученное решение протестировано

Список литературы

[1] Jorge Mena Ryan Rusich. SCTP // www.sctp.ro. — 2006. — URL: http://www.sctp.ro/sctp_repo/papers/jmena_rusich_204_SCTP.pdf (online; accessed: 28.05.2019).

[2] Phillips A. xml // www.ietf.org. — 2006. — URL: https://www.ietf.org/rfc/rfc4646.txt (online; accessed: 29.05.2019).

[3] Postel J. UDP // tools.ietf.org. — 1980. — URL: https://tools.ietf.org/html/rfc768 (online; accessed: 15.05.2019).

[4] Postel Jon. IPv4 // tools.ietf.org. — 1981. — URL: https://tools.ietf.org/html/rfc791 (online; accessed: 15.05.2019).

[5] Postel J. TCP // tools.ietf.org. — 1981. — URL: https://tools.ietf.org/html/rfc768 (online; accessed: 15.05.2019).

[6] S. Deering R. Hinden. IPv6 // tools.ietf.org. — 1998. — URL: https://tools.ietf.org/html/rfc2460 (online; accessed: 15.05.2019).

[7] Stewart R. SCTP // tools.ietf.org. — 2007. — URL: https://tools.ietf.org/html/rfc4960 (online; accessed: 28.05.2019).

[8] Рудычева Наталья. Первое настоящее сравнение «1С:ERP» и SAP ERP // www.cnews.ru. — 2018. — URL: http://www.cnews.ru/articles/2016-12-09_pervoe_nastoyashchee_sravnenie_1serp_i_sap_erp (дата обращения: 21.12.2018).