УДК 004.415.53

ПАРАЛЛЕЛЬНОЕ ТЕСТИРОВАНИЕ БОЛЬШИХ АВТОМАТНЫХ МОДЕЛЕЙ*

© 2011 г. ***И.Б. Бурдонов, С.Г. Грошев, А.В. Демаков, А.С. Камкин, А.С. Косачев, А.А. Сортов***

Институт системного программирования РАН, Москва

sortov@ispras.ru

Поступила в редакцию 20.01.2011

Технология тестирования UniTESK позволяет создавать качественные тесты сложных систем на соответствие формальным моделям требований к ним. В большинстве случаев выполнение таких тестов требует значительного времени. Представлено расширение технологии UniTESK, позволяющее эффективно распараллеливать выполнение теста на вычислительном кластере.

Ключевые слова: функциональное тестирование, формальные спецификации, модели аппаратуры, распределенные системы, распределенное тестирование, конечные автоматы.

Введение

Статья описывает расширение технологии тестирования UniTESK [1], использующее возможности вычислительных кластеров для сокращения времени тестирования сложных систем на соответствие формальным моделям требований к ним.

В технологии UniTESK тестируемая система (реализация) представляется в виде модельного конечного автомата (МКА), входные символы которого соответствуют тестовым стимулам, а выходные символы – выдаваемым тестируемой системой реакциям. Процесс тестирования заключается в прохождении маршрутов по графу состояний МКА (далее называемому «модельным графом» или просто «графом»). При этом на каждом переходе автоматически осуществляются проверки на соответствие наблюдаемого поведения (реакции) тестируемой системы заданным требованиям, которые описаны в виде спецификации. Спецификация определяет класс удовлетворяющих ей реализаций, модельные графы которых являются различными подграфами графа спецификации. Граф спецификации задается неявно пред- и постусловиями стимулов. Поэтому при тестировании граф реализации строится по мере его обхода; обход графа завершается после прохождения всех его дуг (когда обнаружено, что ни из одной известной вершины не выходит непройденная дуга) или обнаружения ошибки, не позволяющей продолжать тестирование.

Подробнее о способе тестирования с помощью конечных автоматов, используемом в технологии UniTESK, можно прочитать в [1, 2]. Такой способ даёт хорошее покрытие требований к тестируемой системе. Однако размер модельного графа может быть очень большим, даже после применения различных способов его сокращения [3]. В результате обход МКА на одной машине может длиться недопустимо долго.

Чтобы ускорить процесс тестирования, сохраняя при этом достоинства технологии UniTESK (в частности, автоматическое обеспечение полноты обхода модельного графа и автоматическую оценку достигнутого покрытия формальной модели тестируемой системы), предлагается использовать возможности, предоставляемые распределёнными вычислительными системами.

Используемый метод тестирования имеет следующие особенности, которые усложняют решение задачи распараллеливания теста.

1. Тестовая последовательность строится в виде единого маршрута по графу, проходящего все его дуги.

2. Граф не задан заранее, а строится динамически по мере его обхода.

3. Как правило, единственным способом перевода тестируемой системы в требуемое состояние является прохождение в графе маршрута, ведущего из текущего состояния в требуемое.

* Статья рекомендована к печати программным комитетом Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: суперкомпьютерные центры и задачи».

Эти особенности не позволяют распределять работу между узлами вычислительного кластера статически. Существуют методы (например, описанный в работе [4]), позволяющие записывать информацию о графе во время его обхода и использовать её в дальнейшем для статического построения кратких маршрутов, что частично решает указанную задачу. Однако для таких методов необходим первоначальный обход графа, который также может занимать недопустимо долгое время; кроме того, статические методы неустойчивы к неизбежным в процессе разработки изменениям как спецификации, так и реализации.

Для решения задачи распараллеливания тестов был разработан метод параллельного обхода графов и на его основе реализована тестовая система.

Требования к тестовой системе

Для возможности параллельной работы тестовой системы необходимо, чтобы структуры МКА и порождённого им модельного графа удовлетворяли следующим ограничениям.

(A) Модельный граф конечен и сильно связан. Это требование для возможности обхода графа налагается технологией UniTESK и при последовательном режиме работы.

(B) На всех узлах вычислительного кластера граф одинаков.

(C) Для любой вершины графа можно вычислить все стимулы, которыми помечены выходящие из неё дуги.

(D) МКА детерминирован в том смысле, что пресостояние и стимул любого перехода однозначно определяют его постсостояние. Это требование может быть ослаблено в зависимости от используемых алгоритмов обхода [5].

Выполнение указанных выше требований необходимо для *возможности* применения предлагаемого метода параллельного тестирования. Кроме того, для *целесообразности* его применения должно выполняться следующее требование.

(E) Обмены информацией о дугах с другими узлами кластера должны требовать существенно меньше времени, чем проходы дуг графа (включающие в себя применение тестового воздействия, его обработку тестируемой системой и оценку правильности полученных реакций) одним узлом кластера.

Архитектура тестовой системы

Тестовая система состоит из множества процессов, выполняющихся на узлах вычислительного кластера. Процессы могут обмениваться сообщениями, но не имеют общих данных: каждый процесс имеет свою копию тестируемой системы и работает, вообще говоря, по своему алгоритму обхода графа. Процессы связаны односторонними сетевыми соединениями типа точка-точка (двусторонние соединения рассматриваются как пары односторонних соединений), образующими сильно связный ориентированный граф; по этим соединениям они обмениваются информацией об исследованной части модельного графа. Предполагается, что соединения сохраняют порядок сообщений, не теряют их и не производят лишних сообщений.

Каждый процесс осуществляет генерацию своей тестовой последовательности и подачу тестовых воздействий на тестируемую систему. Процесс представляет собой систему, состоящую из нескольких взаимодействующих компонентов. Основные компоненты показаны на рис. 1.

Рис. 1. Устройство процесса тестовой системы

Реализация алгоритма обхода или *обходчик* – библиотечный компонент, реализующий некоторый неизбыточный алгоритм обхода графов [2]. Обходчик является пассивным компонентом, в том смысле, что он не подает воздействия на тестируемую систему. Основной задачей обходчика является вычисление маршрута из заданной вершины графа в какую-либо вершину, из которой выходят еще не пройденные дуги.

Все обходчики реализуют единый интерфейс, предоставляющий следующие функции:

- *Инициализировать обходчик*. Инициализирует хранилище, устанавливает параметры обхода и передает обходчику начальную вершину графа и список допустимых в ней стимулов.
- *Вычислить маршрут в графе и подаваемый стимул*. Вычисляет маршрут из указанной вершины графа в вершину, в которой есть еще не пройденные дуги, а также стимул одной из этих дуг. Если в графе отсутствуют непройденные дуги, функция сообщает о завершении тестирования. Если обнаружено нарушение условия сильной связности графа или других требований на граф, налагаемых обходчиком, функция сообщает об ошибке.
- *Добавить в граф пройденную дугу*. Добавляет дугу в хранилище (при этом указывается, получена ли эта дуга синхронизатором от другого процесса или пройдена локально). Вместе с дугой передается информация о числе стимулов, допустимых в конечной вершине дуги.
- *Получить список дуг, пройденных локально*. Возвращает список дуг, пройденных локальным обходчиком с момента последнего вызова этой функции (или начала работы процесса тестовой системы). Эта функция вызывается синхронизатором для получения дуг, которые нужно передать другим обходчикам.
- *Завершить работу обходчика*. Освобождает ресурсы, занятые во время обхода.

Если обходчики, располагающие одинаковой информацией о пройденных дугах и находящиеся в одном состоянии, выбирают очередную непройденную дугу одинаковым образом, то они будут двигаться по одному и тому же маршруту. Это особенно актуально в начале тестирования, поскольку все процессы стартуют из одной и той же начальной вершины. Для повышения эффективности параллельного обхода графа используется «развод» обходчиков по разным подграфам (идеальной является ситуация, когда разные обходчики работают с попарно непересекающимися подграфами). Развод может быть реализован разными способами. Один из наиболее простых подходов основан на передаче каждому обходчику его номера – числа I из множества $\{0, \ldots, N–1\}$, где N – число процессов тестовой системы. Каждый раз, когда обходчик сталкивается с выбором стимула из списка еще не поданных стимулов L, он выбирает тот из них, который расположен по индексу $I \bmod |L|$, где $|\cdot|$ – размер списка. Другим способом является случайный выбор стимула.

Представление графа состояний или *хранилище* – часть обходчика или независимая подсистема, отвечающая за хранение информации о пройденной части графа. Обновление хранилища осуществляется через интерфейс обходчика. Используемое представление данных может зависеть от выбранного алгоритма обхода и особенностей реализации, однако любое представление содержит следующую информацию:

- известные вершины графа;
- пройденные дуги, для которых известны начальная вершина, стимул перехода и конечная вершина;
- стимулы, допустимые в известных вершинах.

Главный цикл теста – активная часть тестовой системы, которая на основе тестового сценария (неизбыточного описания графа), используя обходчик (обновляя модельный граф и вычисляя пути в вершины с непройденными дугами), подает воздействия на тестируемую систему и проверяет полученные от неё реакции.

Синхронизатор – осуществляет обмен данными с аналогичными компонентами на других узлах и с обходчиком.

Протокол синхронизации

Каждый процесс регулярно выполняет процедуру синхронизации, которая инициируется появлением входящих сообщений или обновлениями в локальном хранилище, или таймером. Алгоритм синхронизации следующий:

I. Синхронизатор принимает все входящие сетевые сообщения и запрашивает у обходчика *обновление* – множество новых дуг, пройденных данным процессом с момента последней синхронизации. Обозначим через R (received) множество дуг, информация о которых содержится в принятых сообщениях, S (sent) множество дуг, которые данный процесс уже посылал в сообщениях по выходящим связям, N (new) обновление.

II. В локальное хранилище (через обходчик) добавляется множество *новых* переходов $R\backslash(S\cup N)$, помеченных как полученные от других процессов.

III. По *всем* исходящим соединениям рассылается сообщение, содержащее множество переходов $(R \cup N) \setminus S$.

IV. Обновляется множество посланных переходов: $S := S \cup R \cup N$.

Задав какой-либо способ упорядочения стимулов, мы в силу требований к тестовой системе (B) и (C) можем перенумеровать для каждой вершины Z все допустимые в ней стимулы числами от 1 до $deg(Z)$, где $deg(Z)$ – количество таких стимулов, и далее рассматривать в качестве стимулов переходов только эти номера. Дуги передаются по сети в виде четвёрки (X, Y, Z, $deg(Z)$), где X – идентификатор начальной вершины дуги, Y – номер стимула, Z – идентификатор конечной вершины дуги. Таким образом, получателям сообщений вместе с информацией о новой вершине графа становится также известен список допустимых в ней стимулов.

Несложно убедиться, что при таком протоколе синхронизации информация о каждой дуге, пройденной хотя бы одним процессом, передаётся по каждому соединению ровно 1 раз. Оценим максимальное время T, через которое информация о новой дуге станет гарантированно известна всем процессам (избавляя их от необходимости проходить её самостоятельно). Пусть $\boldsymbol{t}$ – максимальное время передачи информации о дуге от одного процесса к соседнему процессу в сети, включая время обработки сообщения в процессе. *Расстояние* от процесса p_1 до процесса p_2 – минимальная длина пути из p_1 в p_2 по графу межпроцессных соединений. Пусть d – диаметр графа межпроцессных соединений, то есть максимум из расстояний от процесса до процесса. Тогда $T = dt$.

Отметим также, что, используя этот факт, можно очищать множество S, удаляя из него дуги, которые уже были получены по каждой из входящих связей.

Координация

Процесс тестирования может управляться с использованием координатора. Координатор предоставляет пользователю возможность задать конфигурацию вычислительной среды: узлы, на которых запускаются процессы тестовой системы и связи между ними. В начале работы тестовой системы координатор запускает все процессы и передает им информацию об их входящих и выходящих соединениях. Кроме того, координатор позволяет управлять процессом тестирования и централизованно собирать информацию о состоянии процессов, найденных ими ошибках, достигнутом тестовом покрытии, количестве пройденных дуг графа и т.д.

При обнаружении ошибки дальнейшее поведение тестовой системы определяется уровнем критичности этой ошибки. Выделяются следующие уровни:

1) Локально устранимая ошибка – ошибка, которая не влияет на дальнейший обход графа. При обнаружении такой ошибки процесс оповещает координатора и продолжает тестирование и обмен данными с другими процессами.

2) Локально неустранимая ошибка – ошибка, при которой дальнейший обход графа этим процессом невозможен. При обнаружении такой ошибки процесс передает информацию об этом координатору, прекращает тестирование, однако продолжает обмен данными с другими процессами, сохраняя таким образом сильную связность графа межпроцессных соединений.

3) Глобально неустранимая ошибка – ошибка, при которой невозможен дальнейший обход графа всеми процессами. Процесс оповещает координатора, который передает всем остальным процессам указание о прекращении тестирования.

Завершение работы тестовой системы также возможно при получении соответствующей команды от пользователя. В этом случае координатор передает всем узлам указание завершить работу. Процессы могут сохранить свое состояние, чтобы была возможность в дальнейшем возобновить тестирование.

Сохранение и восстановление состояния

Несмотря на ускорение тестирования за счёт распараллеливания, полный обход модельного графа всё равно может занимать весьма продолжительное время. Сохранение результатов проделанной работы для последующего восстановления и продолжения обхода графа позволяет прерывать тестирование, например, для проведения регламентных работ, а также продолжать тестирование после сбоев, не теряя полученных результатов.

Рассмотрим два протокола такого сохранения и восстановления.

Протокол контрольных точек

Построение контрольной точки. Время от времени (по команде координатора или по некому общему соглашению) все процессы приостанавливают обход графа, продолжая при этом обмены информацией, и ждут, когда каждый процесс завершит прохождение очередной

дуги, а в сети перестанут циркулировать сообщения о пройденных дугах. Как показано выше, это произойдёт, как только информация о каждой пройденной дуге достигнет каждого процесса, после чего в хранилище каждого процесса будет содержаться информация об одном и том же наборе вершин, пройденных и непройденных дуг. После этого каждый процесс сохраняет информацию из своего хранилища на локальный диск, а когда сохранение успешно завершено всеми процессами, продолжает тестирование из той вершины модельного графа, в которой он находится в данный момент (или завершает работу, если поступила соответствующая команда от координатора).

Восстановление происходит следующим образом: после запуска всех процессов и установления между ними соединений каждый процесс загружает из локального файла состояние хранилища, сохранённое при последнем успешном построении контрольной точки, после чего начинает обход графа с его начальной вершины.

Протокол журнализации

Несколько процессов при старте назначаются ведущими журнал (если не нужна защита от таких сбоев узла, при которых теряется содержимое его диска, то достаточно одного процесса). Синхронизаторы этих процессов каждый раз при рассылке по сети сообщений с множеством дуг одновременно записывают эти дуги в том же порядке в файл на локальном диске (*журнал*).

Восстановление происходит следующим образом: после запуска всех процессов и установления соединений один из процессов начинает последовательную загрузку дуг из журнала, в то время как все остальные начинают работать обычным образом. Во время загрузки этот процесс не осуществляет обход графа, но принимает и ретранслирует сообщения о новых дугах, полученные от других процессов. Загружаемые из журнала дуги также рассылаются по сети и добавляются в локальное хранилище вместе с полученными от других процессов. Отметим, что выполнение требования (E) гарантирует, что загрузка и рассылка всем процессам сохранённых данных об известной части графа могут быть выполнены существенно быстрее, чем повторный её обход. После завершения загрузки из журнала этот процесс переходит к тестированию в обычном режиме, при этом он может дописать в конец своего журнала дуги, которые в нём отсутствовали, но были получены во время загрузки от других процессов, и в дальнейшем продолжать дописывать в конец журнала все новые дуги.

Опыт применения подхода

Описанный в статье подход был применен для распараллеливания функционального тестирования моделей цифровой аппаратуры [5]. Получаемые в процессе графы состояний в зависимости от сложности моделей и целей тестирования содержали от тысячи до миллиона вершин и, соответственно, от нескольких тысяч до нескольких миллионов дуг. Для прогона тестов использовалось от 1 до 150 компьютеров (процессор: Intel® Core™2 Quad Q9400, 2.66 GHz; оперативная память: 4 Gb), работающих под управлением ОС Linux и объединенных в сеть Ethernet. В табл. 1 приводится условная классификация сложности тестов в зависимости от числа дуг в графе состояний и указывается количество ресурсов, затрачиваемое на прогон тестов (число компьютеров и время).

Аналитическая оценка эффективности распараллеливания представляется затруднительной, поскольку в ней нужно учесть множество факторов, включая время прохода по дуге графа, время, затрачиваемое на передачу данных между процессами, топологию межпроцессных связей и многие другие. При проведении экспериментов измерялся коэффициент эффективности распараллеливания $K(n) = T(1)/(nT(n))$, где $T(n)$ – время прогона теста на n процессах.

Таблица 1

Классификация тестов по уровню сложности

Категория сложности тестов	Число дуг в графе состояний	Используемое число компьютеров	Время, затрачиваемое на прогон тестов, мин
Простые тесты	< 10000	1	< 30
Тесты средней сложности	10000 – 100000	1–10	< 30
Сложные тесты	100000 – 1000000	10–100	< 30
Очень сложные тесты	> 1000000	> 50	> 60

Проведенные замеры показывают, что на задачах функционального тестирования моделей аппаратуры при правильно подобранной топологии значение коэффициента эффективности распараллеливания превосходит 0.8, однако здесь следует сделать несколько замечаний. Во-первых, эксперименты проводились на компьютерах с двумя и более ядрами, что позволяет синхронизатору не отнимать вычислительные ресурсы у обходчика графов. Во-вторых, для разного числа компьютеров использовались разные топологии межпроцессных связей: для 8 и менее компьютеров, как правило, использовалась топология «кольцо», для 9 и более – «двумерный тор».

Для 100–150 компьютеров этих двух вариантов вполне достаточно, однако для эффективного распараллеливания на большем числе компьютеров могут потребоваться другие топологии («трехмерный тор», «гиперкуб»).

Таблица 2

Результаты прогона теста средней сложности

Используемое число компьютеров	Топология межпроцессных связей	Время, затрачиваемое на прогон теста, мин	Коэффициент эффективности распараллеливания
1	—	95.2	1
9	Кольцо	11.8	0.9
9	Тор 3×3	10.9	0.97
16	Кольцо	6.7	0.89
16	Тор 4×4	6.2	0.96
25	Кольцо	4.4	0.87
25	Тор 5×5	4.0	0.95

Таблица 3

Результаты прогона сложного теста

Используемое число компьютеров	Топология межпроцессных связей	Время, затрачиваемое на прогон теста, мин	Коэффициент эффективности распараллеливания
1	—	803.3	1
81	Кольцо	12.2	0.81
81	Тор 9×9	11.4	0.87
100	Кольцо	10.2	0.79
100	Тор 10×10	9.5	0.85

В табл. 2 и 3 представлены типичные результаты прогона тестов. В них показано время, затрачиваемое на прогон теста средней сложности (18227 состояний, 109362 дуги) и сложного теста (84561 состояние, 338244 дуги) на разном числе компьютеров для топологий «кольцо» и «двумерный тор».

Перспективы

Следующие вопросы требуют дополнительного изучения:

• **Выбор оптимального графа обмена данными между узлами вычислительной системы.** Узлы вычислительной системы должны быть соединены между собой таким образом, чтобы информация между ними распространялась как можно быстрее, но без чрезмерной нагрузки на каналы связи. Полярными вариантами соединений являются схемы «каждый-с-каждым» и «кольцо». При выборе схемы связей следует учитывать возможности конкретной системы.

• **Специализация вычислительных узлов.** Возможно, в некоторых ситуациях окажется полезным разделение функциональности вычислительных узлов. Решением ресурсоемких задач могут заниматься специально выделенные узлы, обслуживая запросы всех остальных. Например, обходчики могут самостоятельно принимать решение, если в текущей вершине модельного графа или в соседних с ней есть непройденные дуги. А в сложных случаях, когда расстояние до непройденных дуг велико, запрашивать выбор пути у специально выделенного узла.

• **Централизованное управление обходом графа.** Выделение узлов, которые управляют поведением множества обходчиков, дает возможность координировать их работу. Таким образом можно повысить эффективность распараллеливания особенно для систем, граф состояний которых имеет определенный вид.

В качестве примера рассмотрим ситуацию, когда из начальной вершины выходит две дуги, но одна из них ведет вершину с большим количеством выходящих дуг, а другая – в длинное кольцо, из каждой вершины которого ведет только одна дуга.

Если каждый обходчик принимает решение самостоятельно, то примерно половина пойдет по кольцу и будет дублировать действия друг друга. Централизованное управление позволяет предварительно исследовать возможные пути на определенную глубину, используя небольшое количество обходчиков, а затем распределить основную массу обходчиков по путям пропорционально количеству найденных непройденных дуг.

Заключение

В статье описано расширение технологии UniTESK средствами распараллеливания работы тестовых систем на распределенных вычислительных кластерах. Важной особенностью является то, что распараллеливание осуществляется динамически, без использования информации о структуре модельного графа. С точки зрения инженеров, работать с распределенной тестовой системой не сложнее, чем с обычной тестовой системой, выполняемой на одном компьютере (дополнительные входные данные связаны со структурой межпроцессных соединений, но они задаются один раз, при конфигурации). Предлагаемый подход обеспечивает существенное сокращение времени выполнения тестов, и, как следствие, уменьшение времени, затрачиваемого на обнаружение ошибок, и ускорение процесса проектирования в целом.

Дальнейшее направление работы мы видим в реализации механизмов, нацеленных на повышение гибкости и эффективности распараллеливания тестов. К ним относятся динамическая реконфигурация топологии сети (для сложных тестов может потребоваться ускорение за счет добавления новых вычислительных узлов в граф межпроцессных соединений) и поддержка систем с общей памятью, в частности, многоядерных процессоров (в этом случае возможна более эффективная реализация синхронизаторов, а также совместное использование одного хранилища несколькими процессами).

Список литературы

1. Кулямин В.В., Петренко А.К., Косачев А.С., Бурдонов И.Б. Подход UniTesK к разработке тестов // Программирование. 2003. № 29(6). С. 25–43.

2. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов: детерминированный случай // Программирование. 2003. № 29(5). С. 59–69.

3. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ // Программирование. 2000. № 26(2). С. 61–73.

4. Грошев С.Г. Локализация ошибок методом построения сокращенных трасс // Программирование. 2009. № 35(3). С. 35–50.

5. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Неизбыточные алгоритмы обхода ориентированных графов: недетерминированный случай. // Программирование. 2004. № 1(30). С. 2–17.

6. Иванников В.П., Камкин А.С., Косачев А.С., Кулямин В.В., Петренко А.К. Использование контрактных спецификаций для представления требований и функционального тестирования моделей аппаратуры // Программирование. 2007. № 33(5). С. 47–61.

PARALLEL TESTING OF LARGE AUTOMATA MODELS

I.B. Burdonov, S.G. Groshev, A.V. Demakov, A.S. Kamkin, A.S. Kosachev, A.A. Sortov

The UniTESK technology allows the development of high-quality tests to check complex systems for compliance with formal models of requirements. However, in most cases such tests take a significant amount of time. This article presents an extension of the UniTESK technology which makes it possible to speed up the testing by its parallel execution on a computer cluster.

Keywords: functional testing, formal specifications, hardware models, distributed systems, distributed testing, finite-state automata.