УДК: 20.53.19, 28.23.13

И. В. Петухов
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Использование модели акторов для параллельного выполнения алгоритмов интеллектуального анализа данных, основанных на блоковой структуре

Описывается способ применения модели акторов для параллельных вычислений алгоритмов интеллектуального анализа данных. Рассматривается способ сопоставления актора и блока из блоковой структуры алгоритма. Также рассматриваются 2 подхода к построению блоковой структуры с использованием модели акторов.

Интеллектуальный анализ, параллельные алгоритмы, интеллектуальный анализ распределенных данных, модель акторов

Модель акторов представляет собой математическую модель параллельных вычислений, которая трактует понятие «актор» как универсальный примитив параллельного численного расчета: в ответ на сообщения, которые он получает, актор может принимать локальные решения, создавать новые акторы, посылать свои сообщения, а также устанавливать, как следует реагировать на последующие сообщения [1].

Актор – независимый вычислитель. Актор принимает на вход сообщения от различных адресатов – других акторов, производит определенные вычисления в зависимости от отправителя сообщения, типа пришедших данных и самих данных, а затем формирует новое сообщение и отправляет его соответствующему адресату.

Актор можно рассматривать как функцию – последовательность операций, которая на вход принимает некоторый набор параметров. При этом актор может начать выполнять операции либо синхронно, т. е. когда все параметры получены, либо асинхронно, когда получены необходимые для начала вычислений параметры. При этом выполнение приостанавливается до момента получения требуемых для продолжения параметров.

В декомпозированном на блоки алгоритме может быть несколько типов блоков:

– обычная операция;

– условное ветвление;

– блок цикла (по сути – условное ветвление, в котором, в зависимости от результата, происходит переход на начало последовательности операций либо на следующую за циклом операцию) [2].

В условном блоке происходит проверка условия и, в зависимости от результата, переход на одну из веток выполнения операций. В цикле, в дополнение к проверке условия, производятся дополнительные операции, например увеличение счетчика.

Актор можно представить как некоторую последовательность блоков операций. Актор принимает на вход сообщение, в котором содержится промежуточная модель знаний и некоторые параметры, необходимые для вычислений, обрабатывает набор векторов, который хранится в базе данных (или на диске), используя эту информацию и последовательность блоков, заложенную в актор, и отправляет полученную модель знаний дальше.

Довольно частыми в процессе вычислений могут быть независимые операции над элементами набора данных (набора векторов). Для таких операций можно использовать набор однотипных акторов. Другими словами, вместо одного актора, который обрабатывает весь набор данных, сообщения отправляются на несколько акторов, которые будут обрабатывать только часть набора данных. Однако при таком подходе необходимо где-то разделять и рассылать данные на акторы, а также собирать полученные частичные модели в одну целую.

Таким образом, можно выделить 2 подхода к построению блоковой структуры алгоритма с использованием модели акторов:

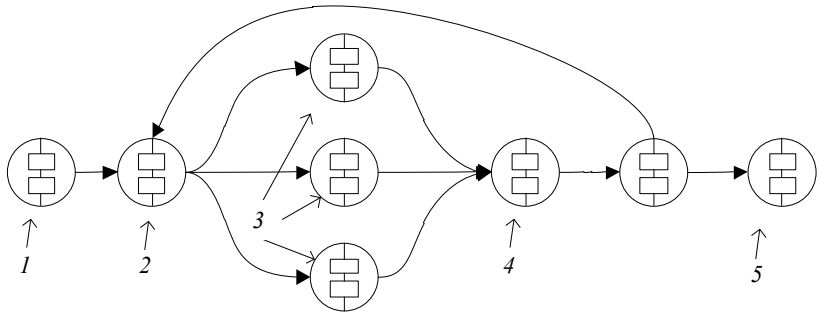

Рис. 1

1. Явная параллелизация. Каждый блок или последовательность блоков является отдельным актором.

2. Неявная параллелизация. Блоковая структура алгоритма едина и выполняется последовательно в одном месте одним исполнителем, а некоторые блоки алгоритма, помеченные как параллельные, выполняются на акторах.

В первом случае общая идея работы системы выглядит следующим образом (рис. 1):

1. Первый актор (*1*) инициируется из внешней среды, и ему передаются начальные настройки.

2. После выполнения содержащихся в нем блоков актор пересылает сообщение с полученным результатом следующему актору или акторам (например, если идет обработка однотипных данных), пока последний актор (*5*) не выполнит свои блоки и не вернет результат обратно.

Стоит отметить, что предварительно все блоки алгоритма должны быть распределены между акторами. Кроме того, каждый актор сам регулирует, куда и когда отправлять сообщения на основе указаний разработчика алгоритма. Если актор (*2*) рассылает сообщения нескольким акторам (*3*), а после завершения обработки они передают результаты одному актору (*4*), то этот актор-приемник должен сам позаботиться о слиянии полученных результатов, т. е. это должен сделать разработчик.

Во втором случае схема работы будет следующей (рис. 2):

1. Блоки начинают выполняться последовательно.

2. При достижении блока, помеченного для параллельной обработки, управление передается специальному адаптеру.

3. Адаптер рассылает акторам задания на выполнение в виде последовательности блоков, которые должен выполнить актор, и результатов вычислений, полученных на предыдущих шагах работы алгоритма.

4. Актор производит необходимые вычисления и отправляет результат обратно адаптеру.

5. Адаптер дожидается завершения всех заданий, объединяет полученные от акторов результаты в один и передает их обратно в блоковую структуру.

6. Выполнение продолжается со следующего блока пока не завершится алгоритм или не встретится очередной помеченный блок.

Стоит отметить, что алгоритм объединения данных должен быть реализован разработчиком.

Для реализации параллельного алгоритма анализа данных в обоих подходах разработчик уже должен иметь последовательную реализацию алгоритма в виде блоковой структуры. Для преобразования последовательной реализации в параллельную в первом подходе разработчику необходимо:

1. Найти блоки или последовательности блоков в алгоритме, результаты выполнения которых не зависят друг от друга. Например, циклы по данным или какие-либо промежуточные вычисления.

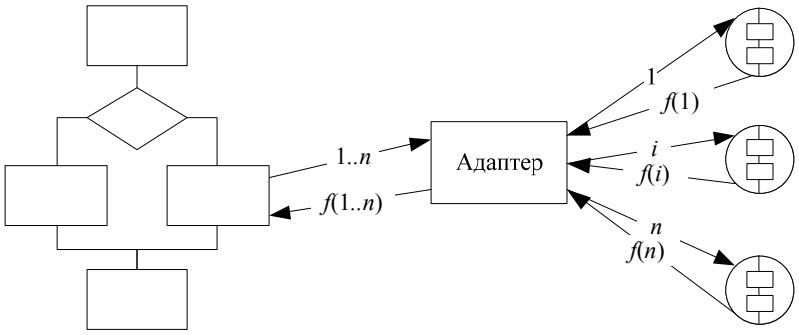

Рис. 2

2. Создать для каждой такой последовательности блоков актор и поместить туда эту последовательность.

3. Настроить правила отправки и приема сообщений между акторами для каждого актора.

В случае использования второго подхода от разработчика требуется:

1. Найти блоки или последовательности блоков в алгоритме, результаты выполнения которых не зависят друг от друга.

2. Поместить такие последовательности в один блок и пометить его как блок для параллельного выполнения.

Рассмотрим подробнее, как будет производиться параллельное выполнение алгоритма в обоих случаях. При использовании обоих подходов возможны две ситуации:

1. Последовательность блоков одна, и требуется обработать с помощью нее некоторый набор данных. При этом результат выполнения последовательности для одних данных не влияет на результат для остальных в данной последовательности. Такую ситуацию можно назвать параллелизацией по данным.

2. Имеется несколько независимых друг от друга последовательностей блоков, т. е. таких, что результат выполнения одной последовательности не повлияет на результаты других последовательностей из данного набора. Такую ситуацию можно назвать параллелизацией по задачам.

В первом подходе используется следующая последовательность действий:

1. Актор, после которого должна производиться параллелизация, рассылает сообщения с результатами своих вычислений акторам, на которых должны производиться вычисления. В случае параллелизации по данным дополнительно указывается диапазон данных, которые нужно обработать актору.

2. Эти акторы выполняют назначенные им задания и отсылают результаты своих вычислений следующему актору.

3. Актор, которому пересылаются сообщения от «параллельных» акторов, ожидает результаты от всех акторов, работавших на предыдущем шаге.

4. Получив все результаты, он объединяет их в общий результат и использует его для дальнейших вычислений.

При использовании второго подхода последовательность действий будет следующей:

1. Как только выполнение алгоритма доходит до блока, помеченного как параллельный, управление передается адаптеру.

2. Адаптер рассылает акторам сообщения с последовательностями блоков, которые должны быть выполнены на них, и результаты предыдущего блока алгоритма. В случае параллелизации по данным дополнительно указывается диапазон данных, которые нужно обработать.

3. Акторы получают сообщения от адаптера и начинают выполнять отправленные им задания. После завершения обработки результат отправляется обратно адаптеру.

4. Адаптер дожидается получения всех результатов от акторов, а затем объединяет их в один результат, который будет передан следующему блоку алгоритма.

Реализация данных подходов возможна на основе библиотеки построения распределенных алгоритмов с помощью блоковой структуры DXelopes. [3] В обоих случаях можно использовать базовые блоки для построения алгоритма: Step, DecisionStep, CyclicStep.

При первом подходе реализация блока операции (ActorStep) будет основана на базовом классе блока (Step). Блок ActorSrep (рис. 3) содержит набор блоков операций (*2*), представляющих собой часть алгоритма, которую должен обработать данный блок, приемник (*1*) и отправитель (*3*) сообщений. Принимая сообщения, блок подготавливает параметры, необходимые для вычислений, затем передает управление внутреннему набору блоков, ожидает его завершения и отправляет сообщения другим акторам, в зависимости от настроек.

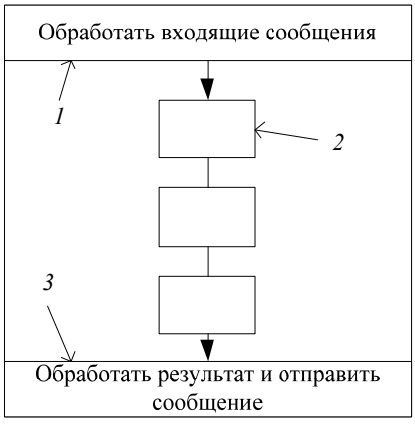

Рис.3

На основе базового блока ActorStep можно реализовать блоки для параллельных вычислений: блок рассылки сообщений ActorSenderStep и блок слияния результатов ActorRecieverStep. Основное различие между этими блоками кроется на этапах приема и отправки сообщений.

Блок ActorSenderStep после завершения выполнения внутренних блоков рассылает сообщения всем указанным адресатам. Разработчик алгоритма должен явным образом указать, куда отправлять сообщения.

Блок ActorRecieverStep ожидает прихода сообщений от всех указанных ему адресатов, затем объединяет полученные из сообщений результаты в один и передает их во внутренние блоки, после чего отправляет полученный результат далее.

Для второго подхода не надо реализовывать базовые блоки, а можно использовать уже существующие в библиотеке. Однако при этом необходимо реализовать адаптер для параллельной обработки данных. В библиотеке предусмотрена возможность использования блоков для параллельной обработки независимых наборов однородных векторов. При выполнении такого блока управление передается специальному адаптеру, который управляет распределением данных для их параллельной обработки. В случае использования модели акторов такой адаптер должен будет формировать сообщения с диапазонами векторов и блоками, с помощью которых будет производиться обработка, отсылать эти сообщения акторам-обработчикам, а затем ожидать завершения выполнения всех вычислений, получать результаты и правильным образом их объединять.

Рассмотрим достоинства и недостатки каждого из подходов.

Особенностью первого подхода является возможность создания сильно распределенного алгоритма, в котором последовательные и циклические операции, не зависимые друг от друга, выполняются параллельно, а затем все данные собираются вместе и вычисление продолжается дальше. Это же является и недостатком, так как разработчику необходимо достаточно четко указывать последовательность действий и точки взаимодействия отдельных акторов при реализации алгоритма интеллектуального анализа данных.

Второй подход более прост в реализации для разработчика алгоритма, так как в этом случае ему необходимо реализовать только саму блоковую структуру алгоритма и отметить в ней блоки, предназначенные для параллельной обработки данных, например блоки, которые работают с однородными наборами данных (векторами). Однако простота подхода ограничивает его применение в основном параллельным вычислением независимых наборов однородных данных.

Таким образом, основными достоинствами для первого подхода являются:

1) гибкость при построении блоковой структуры;

2) высокая степень параллелизации.

Недостатки:

1) сложность построения блоковой структуры (для ее распределенной работы);

2) необходимость реализации маршрутизации (кому отправлять сообщения) в каждом блоке;

3) необходимость реализации алгоритма слияния данных после каждого этапа параллелизации.

Достоинства второго подхода:

1) простота построения блоковой структуры;

2) разработчику не надо разбираться в работе системы акторов.

Недостатки:

1) ограниченность параллелизации (в основном, только циклические операции над векторами);

2) необходимость реализации алгоритма слияния данных;

3) сложность слияния данных, так как оно происходит в одной точке.

В дальнейшем планируется более углубленное исследование второго подхода, так как, несмотря на большую универсальность первого подхода, на разработчиков алгоритмов накладывается дополнительная ответственность за реализацию взаимодействия между акторами. Второй подход хоть и является менее гибким по сравнению с первым, но при этом он перекладывает заботу о реализации взаимодействия между акторами на адаптер. В дополнение к этому существует возможность дальнейшего улучшения алгоритма взаимодействия акторов через адаптер независимо от разработчиков алгоритмов.

СПИСОК ЛИТЕРАТУРЫ

1. Clinger W. D. Foundations of actors semantics / MIT, MA, 1981.

2. Холод И. И. Унифицированная модель Data Mining // Сб. докл. XV Междунар. конф. по мягким вычислениям и измерениям SCM`2012, СПб., 25–27 июня, 2012. Т. 1. С. 237–240.

3. Холод И. И., Каршиев З. А. Метод построения параллельных алгоритмов интеллектуального анализа данных из потоконезависимых функциональных блоков // Изв. СПбГЭТУ «ЛЭТИ». 2013. № 8. С. 38–45.

I. V. Petukhov
Saint-Petersburg state electrotechnical university «LETI»

USING ACTORS MODEL FOR BLOCK BASED DATA MINING ALGORITHM PARALLEL EXECUTION

This paper describes method of actors model application for data mining algorithm parallal execution. The paper reviews a way to combine actors model and algorithm's block structure. The paper also reviews two ways of building algorithm's block structure using the actors.

Data mining, parallel algorithms, distributed data mining, actors model