набор предикатов, позволяющих выделять подмножества исходного множества (см. срезы данных).

Поставленную выше задачу можно решать различными способами ввиду того, что нет чёткого определения “наибольшего влияния на изменения”, но мы пока остановимся на трёх подходах:

- Использование анализа чувствительности
- Деревья решений и коэффициент влияния факторов
- Сравнение среза с другой подвыборкой

Воспользоваться анализом чувствительности можно следующим образом:

3. Возьмём в качестве аргументов некоторой функции все имеющиеся срезы (0 или 1, принадлежит запрос этому срезу или нет), а затем с помощью регрессии построим такую искусственную функцию, которая будет приближать значение метрики на совокупности срезов.
4. Подсчитаем коэффициенты Соболя (Sobol Indices) для этой функции и на основе их в качестве результата вернём список, отсортированный по их значениям.

Деревья решений являются одним из подходов в машинном обучении, позволяющим решать, среди прочих, задачу регрессии. Полезным побочным эффектом их применения является возможность подсчёта “полезности” отдельного аргумента в деле предсказания значения целевой функции. Можно воспользоваться этими коэффициентами для выделения менее и более “важных” срезов, по аналогии с предыдущим пунктом.

Ещё одним способом решения задачи является следующий:

5. Рассмотрим каждый из срезов данных.
6. Выделим ему срез для сравнения: случайный срез аналогичного размера, всё непопавшее в срез и т.п.
7. Сравним с использованием статистических критериев значения на исходном и парном срезе.
8. Используя результаты сравнения отранжируем срезы.

На основе перечисленных выше методов строится обобщённый инструмент, предоставляющий возможности конфигурирования, предподсчёта данных, запроса списка наиболее повлиявших срезов и визуализации результатов.

Подход, описанный в этом докладе, позволяет автоматизировать анализ и помогает в исследовании изменений большой системы. Благодаря этому можно оперативнее реагировать на проблемы, а также лучше и точнее анализировать результаты вносимых в систему правок.

Список использованных источников:

1. C.D. Manning, P. Raghavan, H. Schütze. Introduction to Information Retrieval, Cambridge UP, 2008
2. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola Global Sensitivity Analysis: The Primer, John Wiley & Sons, 2008

МОДЕЛЬ АКТОРОВ. ПРИМЕНЕНИЕ ДЛЯ СОЗДАНИЯ МНОГОПОТОЧНЫХ ПРИЛОЖЕНИЙ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Селюк М. И., Владыко В. Д.

Сиротко С. И. – канд. физ.-мат. наук, доцент

В настоящее время наращивание вычислительных мощностей и производительности систем достигается в основном благодаря параллельным вычислениям. Это делает особенно актуальным эффективную реализацию параллелизма в программном обеспечении. Современная модель параллелизма, основанная на низкоуровневых потоках, имеет ряд серьёзных проблем. Новая концепция - модель акторов - успешно решает эти проблемы.

Цель данной работы – освоение технологии акторов, её применение на примере создания простейшего многопоточного приложения на языке Scala с использованием библиотеки Akka, а также анализ особенностей и перспектив модели на основе полученного опыта.

Недостатки традиционной модели вычислений, основанной на понятии машины Тьюринга, становятся очевидны при разработке сложных вычислительных систем с использованием параллелизма. Понятие глобального времени, когда в каждый момент выполняется одна неделимая операция, приводит к необходимости использовать низкоуровневые примитивы синхронизации потоков, для разрешения коллизий, связанных с разделяемым состоянием. Высокая сложность создания таких программ, вызванная разрывом между человеческим и машинным представлением, влечёт за собой совершение большого количества ошибок, возникновение гонки за данными, взаимных блокировок, а также плохой масшабируемости. Таким

образом, современная модель параллелизма, основанная на потоках, является слишком низкоуровневой и плохо подходит для программирования человеком.

Решить данные недостатки призвана модель акторов. Разработанная в 1973 году Карлом Хьюиттом, она в качестве главных «персонажей» программы предлагает акторы, подобно тому, как в объектно-ориентированном программировании это место занимают объекты. Актор – это вычислительная сущность, способная выполнять в ответ на полученное сообщение, одно из трёх действий (или несколько - параллельно):

- отправление конечного числа сообщений другим акторам;
- создание конечного числа новых акторов;
- изменение своего поведения для получения следующего сообщения.

Главная особенность и фундаментальное достижение модели – развязка отправителя и сообщений.

Возможный образ думать о традиционной и акторной моделях – представлять первую как о телефонной системе (позволяет дозвониться только одному человеку в данный момент времени), а вторую – как электронную почту (имеется очередь сообщений, на которые получатель реагирует).

В процессе выполнения работы с целью лучшего ознакомления с возможностями и особенностями модели акторов было создано простейшее многопоточное приложение с её использованием. В качестве языка программирования был выбран язык Scala , разработанный в 2003 году под руководством Мартина Одерски и сочетающий преимущества объектно-ориентированного и функционального программирования. Выбор был обусловлен широкими возможностями распределённого программирования на Scala.

Библиотека Akka ещё больше расширяет возможности языка, предлагает средства продвинутого параллельного программирования и на данный момент является стандартом де-факто реализации модели акторов.

Созданное приложение имитирует принцип работы кафе: выполнение барменом и официантами заказов посетителей. Используются базовые приёмы модели – создание акторов, отправление и обработка сообщений, возможность наличия у акторов внутреннего изменяемого состояния, а также более продвинутые средства, например, иерархии акторов и обработка ошибок времени выполнения.

На основании опыта программирования приложения и экспериментов с ним был сделан вывод: программы, разработанные с применением концепции акторов, просты в создании и сопровождении, а также принципиально не могут содержать ошибок, вызванных гонками и взаимными блокировками. Всё это делает модель акторов весьма перспективной для разработки распределённых систем с высокими требованиями к производительности. Среди основных недостатков концепции: отсутствие прямой связи с объектно-ориентированной моделью, сложность определения требований по памяти до запуска, а также проблемы, связанные с асинхронным выполнением. Тем не менее, преимущества модели акторов с избытком компенсируют её немногочисленные недостатки.

Список использованных источников:

1. Таненбаум Э. Современные операционные системы / пер. с англ. СПб.: Питер, 2015. – 1120 стр.: ил.
2. John C. Mitchell. Concepts in programming languages. — Cambridge University Press, 2003. — 529 p.
3. Руководство по работе с языком Scala для занятых разработчиков Java-приложений // IBM developerWorks [Электронный ресурс]. — 2015. Режим доступа : http://www.ibm.com/developerworks/ru/library/j-scala04109/index.html — Дата доступа : 15.03.2015.

ОБЗОР МЕТОДОВ ФАКТОРИЗАЦИИ БОЛЬШИХ ЧИСЕЛ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Аксамит М. В. Былинович В. Н.

Стройникова Е. Д. – ассистент

В математике до сих пор не найден эффективный способ факторизации больших чисел. Существующие алгоритмы обладают рядом недостатков, и ни один из них не может дать стопроцентный результат для любого числа за достаточно короткое время.

Факторизация больших чисел - вычислительно сложная задача, которая лежит в основе широко известного алгоритма шифрования RSA. Этот алгоритм имеет два ключа: открытый, которым информация зашифровывается, и закрытый, который может расшифровать. Один из ключей известен, а он в свою очередь состоит из двух частей, одна из которых – произведение больших простых чисел (обозначается n), с помощью которых генерировались оба ключа. Следовательно, если факторизовать число n, то шифр будет взломан.

В зависимости от сложности алгоритмы факторизации можно разделить на две группы: экспоненциальные и субэкспоненциальные алгоритмы.