ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

На правах рукописи

КАМИЛЬ Висам Абдуладим Камиль

УПРАВЛЕНИЕ ПРОЦЕССАМИ ОБРАБОТКИ ДАННЫХ В МЕГАСЕТЯХ НА ОСНОВЕ ГРАФОВЫХ МОДЕЛЕЙ И СИСТЕМЫ РАЗРАБОТКИ ПОТОКОВЫХ ПРИЛОЖЕНИЙ

Специальность: 2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель:
д.т.н., доцент Мутин Денис Игоревич

Воронеж – 2025

Содержание

Введение 4

1. Особенности разработки потоковых приложений в синтезируемых мегасетях 15

1.1. Проблема оптимального проектирования мегасети из множества изолированных подсетей 15

1.2. Исследования в области кластеризации сенсорной сети 19

1.3. Особенности разработки потоковых приложений 24

1.4. Цель работы и задачи исследования 31

2. Оптимизация согласованности подсистем при обработке данных в мегасетях 33

2.1. Графовое проектирование мегасети из изолированных подсетей 33

2.2. Графовая модель мегасети 34

2.3. Расстояние по сопротивлению и отрицательные ребра 37

2.4. Согласовательный анализ в объединении сетей 41

2.5. Согласованность с дополнительными узлами моста 46

2.6. Численные примеры 54

2.7. Выводы 56

3. Кластерные алгоритмы сетевой оптимизации данных в специальных мегасетях на основе нечеткой логики 58

3.1. Оптимизация сети и кластеризация 58

3.2. Алгоритм кластеризации 61

3.3. Оценка эффективности 71

3.4. Сравнительное исследование 74

3.5. Выводы 79

4. Разработка распределенных потоковых приложений в мегасетях на основе UML-моделей 81
4.1. Инструмент разработки распределенных потоковых приложений в мегасетях, управляемых моделями 81
4.2. Общие сведения об особенностях разработки потоковых приложений 82
4.3. Обзор инструмента разработки потоковых приложений 84
4.4. Язык моделирования 86
4.5. Операторы потоковой передачи и трансформеры 101
4.6. Генерация кода 107
4.7. Оценка 110
4.8. Выводы 130

Заключение 132

Список использованных источников 134

ВВЕДЕНИЕ

Актуальность темы. Информационно-вычислительные сети являются магистральным хребтом современного цифровизированного общества. Особенно важной является задача обеспечения согласованного функционирования таких распределенных систем. Одной из подзадач является проблема реализации устойчивости образованных мегасистем к штатным или несанкционированным внешним возмущениям. В конечном счете взаимозависимость между устойчивостью сети и ее структурой позволяет рационализировать интегрированную производительность именно через структуру мегасети. Большой вклад в развитие методов создания и управления внесли Зиганурова Р.А., Ковалев И.В., Кравец О.Я., Олескин А.В., Antsaklis P.J., Jovanovic M.R., Murray R.M. В теоретическом плане ряд методов создания мегасетей сводится к оптимизации выбора главных кластеров и мостов каждого кластера. Разумной идеей кажется модифицировать архитектуру системы управления облачными средами, расширив ее на случай модели объединенной сети, в которой узлы управляются зашумленной динамикой согласования, а веса ребер могут быть положительными или отрицательными.

С развитием таких технологий, как эффективные алгоритмы оптимизации сети, агрегирование данных и маршрутизация стали ключевыми подходами к оптимизации. Информация для сенсорных узлов, широко распространенных в современных информационно-вычислительных системах, становится высокодоступной. Каждый раз вместо того, чтобы принимать во внимание отдельные сенсорные узлы, система может распределять свои ограниченные ресурсы в определенных кластерах для оптимизации затрат и ресурсов. Важные знания могут быть извлечены с использованием соответствующих и эффективных алгоритмов интеллектуального анализа данных. Необходимы алгоритмы

кластеризации групп сенсорных узлов мегасети, обеспечивающие инвариантность к большому объему сетевых данных и динамичности сетевого местоположения узлов.

За последние несколько лет резко выросла значимость технологий и приложений, требующим обработки больших объемов данных, порождаемых мегасетями. Сложность извлечения такой информации обычно связана с объемом, скоростью и разнообразием анализируемых данных. Конкретные сценарии работы с большими данными могут обладать одной или несколькими из этих характеристик. В частности, для тех сценариев, в которых объем и скорость имеют первостепенное значение, в игру вступают распределенные потоковые платформы. Они позволяют разрабатывать потоковые приложения, т.е. приложения, которые обрабатывают потенциально бесконечные потоки данных, непрерывно и быстро обновляя полезные выходные данные (статистику, прогнозы и т.д.). Отсюда понятна необходимость создания архитектуры системы разработки потоковых приложений для обработки данных на основе моделей потоков данных

Таким образом, актуальность темы диссертационного исследования продиктована необходимостью дальнейшего развития средств математического и программного обеспечения управления процессами обработки данных в мегасетях на ове графовых моделей и системы создания потоковых приложений.

Тематика диссертационной работы соответствует научному направлению ФГБОУ ВО «Воронежский государственный технический университет» «Вычислительные комплексы и проблемно-ориентированные системы управления».

Целью работы является разработка математического и программного обеспечения управления процессами обработки данных в мегасетях на основе графовых моделей и системы создания потоковых

приложений для обработки данных с использованием специальной метрики согласованности и интеграции логики приложений.

Задачи исследования. Для достижения поставленной цели необходимо решить следующие задачи:

1. Провести анализ проблем создания математического и программного обеспечения управления процессами обработки данных в мегасетях на основе графовых моделей и системы проектирования потоковых приложений с использованием специальной метрики согласованности и интеграции логики приложений.

2. Создать графовую модель синтеза мегасети из набора непересекающихся подграфов с согласованностью в виде нормы H_2, обеспечивающую соединение подграфов с обеспечения оптимальной согласованности финальной мегасети.

3. Сформулировать оптимизационную задачу выбора мостов подграфов мегасети с построением соединительных ребер, обеспечивающую получение оценок минимальной и максимальной согласованности с весами ребер на положительной полуоси.

4. Разработать алгоритм кластеризации данных групп сенсорных узлов мегасети, обеспечивающий инвариантность к большому объему сетевых данных и динамичности сетевого местоположения узлов.

5. Создать архитектуру системы разработки потоковых приложений для обработки данных на основе моделей потоков данных, обеспечивающую интеграцию в среду IDE поверх UML.

Объект исследования: процессы обработки данных в мегасетях с потоковыми приложениями в их составе.

Предмет исследования: структура математического и программного управления процессами обработки данных в мегасетях на основе алгоритмов кластеризации данных и архитектуры системы создания потоковых приложений для обработки данных.

Методы исследования. При решении поставленных в диссертации задач использовались теория графов, теория вероятностей, теория принятия решений, а также методы объектно-ориентированного программирования.

Тематика работы соответствует следующим пунктам паспорта специальности 2.3.5 «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»: п.3 «Модели, методы, архитектуры, алгоритмы, языки и программные инструменты организации взаимодействия программ и программных систем»; п.9 «Модели, методы, алгоритмы, облачные технологии и программная инфраструктура организации глобально распределенной обработки данных».

Научная новизна работы. В диссертации получены следующие результаты, характеризующиеся научной новизной:

1. Графовая модель синтеза мегасети из набора непересекающихся подграфов с согласованностью в виде нормы H_2, отличающаяся непрерывностью весов ребер на всей числовой оси и зашумленной динамикой согласования узлов, реализующая соединение подграфов с обеспечением оптимальной согласованности финальной мегасети.

2. Оптимизационная задача выбора мостов подграфов мегасети с построением соединительных ребер, отличающаяся параллелизмом процесса решения и обеспечивающая получение оценок минимальной и максимальной согласованности с весами ребер на положительной полуоси.

3. Алгоритм кластеризации данных групп сенсорных узлов мегасети, отличающийся использованием нечеткой логики для учета гетерогенных параметров сенсорной сети и гетерогенного управления сетью, обеспечивающий инвариантность к большому объему сетевых данных и динамичности сетевого местоположения узлов.

4. Архитектура системы создания потоковых приложений для

обработки данных на основе моделей потоков данных, отличающаяся интеграцией логики приложения в модель и обеспечивающая интеграцию в среду IDE поверх UML.

Теоретическая и практическая значимость исследования заключается в разработке математического и программного обеспечения управления процессами обработки данных в мегасетях на основе графовых моделей и системы проектирования потоковых приложений с использованием специальной метрики согласованности и интеграции логики приложений.

Теоретические результаты работы могут быть использованы в проектных и научно-исследовательских организациях, занимающихся проектированием платформенно-инвариантных систем управления мегасетями с потоковыми приложениями обработки данных.

Положения, выносимые на защиту

1. Графовая модель синтеза мегасети из набора непересекающихся подграфов с согласованностью в виде нормы H_2 реализует соединение подграфов с обеспечением оптимальной согласованности финальной мегасети.

2. Оптимизационная задача выбора мостов подграфов мегасети с построением соединительных ребер обеспечивает получение оценок минимальной и максимальной согласованности с весами ребер на положительной полуоси.

3. Алгоритм кластеризации данных групп сенсорных узлов мегасети обеспечивает инвариантность к большому объему сетевых данных и динамичности сетевого местоположения узлов.

4. Архитектура системы создания потоковых приложений для обработки данных на основе моделей потоков данных обеспечивает интеграцию в среду IDE поверх UML.

Результаты внедрения. Основные результаты внедрены в Научно-

исследовательском институте вычислительных комплексов им. М. А. Карцева» (г. Москва) при проектировании распределенной информационно-вычислительной системы, в учебный процесс Воронежского государственного технического университета в рамках дисциплин: «Вычислительные машины, системы и сети», «Информационные сети и телекоммуникационные технологии», а также в рамках курсового и дипломного проектирования.

Апробация работы. Основные положения диссертационной работы докладывались и обсуждались на следующих конференциях: XXIX International Open Science Conference «Modern informatization problems in the technological and telecommunication systems analysis and synthesis» (Yelm, WA,, USA, 2024); VII Международной НПК «Наука и технологии: перспективы развития и применения» (Петрозаводск, 2024); VI Всероссийской НПК «Информационные технологии в экономике и управлении» (Махачкала, 2024); XXX International Open Science Conference «Modern informatization problems in the technological and telecommunication systems analysis and synthesis» (Yelm, WA,, USA, 2025), а также на научных семинарах кафедры автоматизированных и вычислительных систем ВГТУ (2023-2025 гг.).

Достоверность результатов обусловлена корректным использованием теоретических методов исследования и подтверждена результатами сравнительного анализа данных вычислительных и натурных экспериментов.

Публикации. По результатам диссертационного исследования опубликовано 12 научных работ (2 – без соавторов), в том числе 5 – в изданиях, рекомендованных ВАК РФ (из них 1 – в издании Wos и одно свидетельство о регистрации программы для ЭВМ). В работах, опубликованных в соавторстве и приведенных в конце автореферата, лично автором получены следующие результаты: [1, 7] - графовая модель

синтеза мегасети из набора непересекающихся подграфов с согласованностью в виде нормы H_2; [2, 8] - оптимизационная задача выбора мостов подграфов мегасети с построением соединительных ребер; [4, 9, 10] - алгоритм кластеризации данных групп сенсорных узлов мегасети с использованием нечеткой логики для учета гетерогенных параметров сенсорной сети и гетерогенного управления сетью; [12] - архитектура системы создания потоковых приложений для обработки данных на основе моделей потоков данных с интеграцией в среду IDE поверх UML; [3, 5] - информационное и программное обеспечение для экспериментальной оценки качества разработанных методов и алгоритмов.

Структура и объем работы. Диссертационная работа состоит из введения, четырех глав, заключения, списка литературы из 159 наименований. Работа изложена на 150 страницах.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

В первой главе исследуются особенности управление процессами обработки данных в мегасетях на основе графовых моделей и системы разработки потоковых приложений. Отмечено, что повысить эффективность управления можно путем применения графовой модели синтеза мегасети из набора непересекающихся подграфов с согласованностью в виде нормы H_2, обеспечивающей соединение подграфов с обеспечением оптимальной согласованности финальной мегасети, оптимизационной задачи выбора мостов подграфов мегасети с построением соединительных ребер, обеспечивающей получение оценок минимальной и максимальной согласованности с весами ребер на положительной полуоси, разработаткиь алгоритма кластеризации данных групп сенсорных узлов мегасети, обеспечивающего инвариантность к большому объему сетевых данных и динамичности сетевого

местоположения узлов, создания архитектуры системы разработки потоковых приложений для обработки данных на основе моделей потоков данных, обеспечивающей интеграцию в среду IDE поверх UML. Результат анализа потребовал формализации данных задач, а также алгоритмизации их решения. Сформулирована цель и задачи исследования.

Вторая глава посвящена исследованию оптимизации согласованности подсистем при обработке данных в мегасетях.

Представлена графовая модель синтеза мегасети из набора непересекающихся подграфов с согласованностью в виде нормы H_2, отличающаяся непрерывностью весов ребер на всей числовой оси и зашумленной динамикой согласования узлов, реализующая соединение подграфов с обеспечением оптимальной согласованности финальной мегасети.

Исследована проблема оптимизации согласованности сети сетей (КоС) с произвольным знаком весов ребер. Решается задача выбора узлов-мостов между всеми подсистемами КоС, узлы-мосты связываются межсистемными ребрами. Проблема оптимизации решается с учетом связи согласованности и эффективного сопротивления. Минимизация осуществляется путем поиска узлов-мостов в графах ограниченного сопротивления. Эти узлы-мосты могут быть идентифицированы независимо от других подграфов и соединительной топологии. Доказано, что для КоС с древовидным опорным графом решение является оптимальным даже в рамках более общей модели, допускающей наличие нескольких узлов моста на подграф. Представлены границы согласованности, а также аналитические примеры оптимальных КоС. Представлены численные результаты, иллюстрирующие производительность различных топологий КоС.

Представлена оптимизационная задача выбора мостов подграфов мегасети с построением соединительных ребер, отличающаяся

параллелизмом процесса решения и обеспечивающая получение оценок минимальной и максимальной согласованности с весами ребер на положительной полуоси.

Третья глава посвящена исследованию и разработке алгоритма кластеризации данных групп сенсорных узлов мегасети, отличающегося использованием нечеткой логики для учета гетерогенных параметров сенсорной сети и гетерогенного управления сетью, обеспечивающего инвариантность к большому объему сетевых данных и динамичности сетевого местоположения узлов.

Кластеризация беспроводных сенсорных сетей для сложной многомерной сети имеет решающее значение. Это важно для оптимизации многомерной сети, так как учитывает многочисленные атрибуты, дающие благоприятные результаты. В работе представлен подход к узлам кластерных датчиков с аналогичными характеристиками в рамках иерархической структуры. Структура может определять и классифицировать атрибуты каждого сенсорного узла по основным и второстепенным критериям. Лингвистические переменные используются для представления каждого критерия. Каждая лингвистическая переменная затем преобразуется в трапециевидное нечеткое число для повышения точности и стандартизации. При этом предложен кластерный подход для перечисления воздействия всех критериев. Точность кластеризации определяется для измерения межкластерного и внутрикластерного расстояния. Предлагаемый подход превосходит существуюие подходы. Он может обрабатывать несколько атрибутов для сенсорного узла в подводной беспроводной сенсорной сети. По сравнению с предыдущими подходами, это помогает значительно отличить разницу между сенсорными узлами.

В главе 4 проанализированы особенности архитектуры системы разработки потоковых приложений для обработки данных на основе

моделей потоков данных, отличающейся интеграцией логики приложения в модель и обеспечивающая интеграцию в среду IDE поверх UML.

Растущий спрос на распределенные потоковые приложения создает потребность в соответствующих абстракциях и связанных языках и инструментах, позволяющих справиться с их сложностью и неоднородностью.

Предложен инструмент разработки потоковых приложений, основанный на моделях, состоящий из нового профиля UML, который позволяет определять модель приложения, используя нотацию UML.

Рассмотрена архитектура системы, управляемой моделями, которая опирается на концептуальные сходства между различными потоковыми платформами для упрощения и ускорения проектирования, разработки и эксплуатации распределенных потоковых приложений, одновременно преодолевая проблему привязки запуска к конкретному исполнению движка. Система основана на предметно-ориентированный язык моделирования в форме профиля UML, который позволяет разработчикам моделировать потоковое приложение в виде графа потока данных, используя стандартное моделирование UML и, в частности, составные структурные диаграммы.

Основная идея заключается в преобразовании приложений в топологии или прямые ациклические графы (DAG), в которых операторы являются компонентами, которые можно развертывать по отдельности, и обеспечивают механизмы отказоустойчивости. Кроме того, операторы обычно распараллеливаются, что означает наличие нескольких запущенных экземпляров оператора, каждый из которых можно рассматривать как отдельный процесс, который потенциально может выполняться в своей собственной среде.

Согласно стратегиям развертывания, одно потоковое приложение на самом деле представляет собой совокупность независимых и

индивидуально развертываемых компонентов. Платформа распределенной потоковой передачи отвечает за принятие решений о том, как распределять параллельные вычисления, с конкретными указаниями, в соответствии с которыми должны быть разделены входные данные, в то время как планировщик затем приступает к планированию заданий и обеспечению их выполнения.

1. ОСОБЕННОСТИ РАЗРАБОТКИ ПОТОКОВЫХ ПРИЛОЖЕНИЙ В СИНТЕЗИРУЕМЫХ МЕГАСЕТЯХ

1.1. Проблема оптимального проектирования мегасети из множества изолированных подсетей

Информационно-вычислительные сети являются магистральным хребтом современного цифровизированного общества. Назовем только распределенные электрические системы [2.1], сети мобильных транспортных объектов [2.2, 2.3], сети нестационарных сенсоров [2.4] и т.д. Особенно важной является задача обеспечения согласованного функционирования таких распределенных систем. Одной из подзадач является проблема реализации устойчивости образованных мегасистем к штатным или несанкционированным внешним возмущениям. В конечном счете взаимозависимость между устойчивостью сети и ее структурой позволяет рационализировать интегрированную производительность именно через структуру мегасети.

Рассматриваются мегасети, которые в сущности представляют собой конгломерат сетей (сеть сетей, КоС). Задача – создание топологической структуры мегасетей как КоС. Такая топологическая структура представляет собой множество подграфов, индуцированных сетями-компонентами, и сети межграфовой связи в виде множества соединяющих ребер.

Примеры мегасети – множество подсистем связанных беспроводными сетевыми структурами датчиков. Задача информационной интеграции в мегасети решается путем установления «длинной» беспроводной или проводной связи между подсистемами датчиков. Иной пример – мегасеть из локальных наборов автономных подвижных объектов. При этом в каждом наборе автономные подвижные объекты

автоматически управляют собственным движением на основе локальной информации о движении других объектов набора. При этом обобщенное движение наборов в пределах одного вектора движения также необходимо управление для реализации безаварийного движения. Следующий пример КоС – социальные сети. В некоторых из них возможна стратификация на несколько сообществ, которые «слабо общаются» друг с другом. Отдельный класс примеров КоС – это функционирование больших электрических и иных инфраструктурных сетях [2.5]. Создание связей между подграфами может быть дорогостоящим из-за затрат на инфраструктуру или связь. Поэтому важно, чтобы эти соединительные элементы были продуманно спроектированы для оптимизации производительности полного КоС.

Сначала представлена модель объединенной сети, в которой узлы управляются зашумленной динамикой согласования, а веса ребер могут быть положительными или отрицательными [2.6]. Задача оценки производительности мегасети может быть сведена к задаче оценки надежности мегасети по метрике H_2, которую назовем согласованностью, как в [2.7]. Оценка надежности мегасети по метрике H_2 дает оценку точности и корректности взаимодействия узлов при штатных или нерегламентированных внешних взаимодействиях. Также такая оценка является косвенной для понимания переходных процессов в электрических сетях с децентрализованным управлением [2.1].

Рассматривается графовая модель КоС. В рамках модели находится единственный узел подграфа, который будет играть роль моста. Объединительные для подграфов ребра могут соединять только узлы моста. Оптимизационная задача фактически состоит в, прежде всего, выборе узлов моста мегасети с построением соединительных ребер. Это – не последовательная, а параллельная задача, что и определяет ее сложность.

Несмотря на параллельную формулировку, далее установлено, что задача глобальной оптимизации решается множеством решений задач локальной оптимизации выбора узлов моста по мегасети. Если рассматривать сущность оптимальной информационной центральности [2.8], то оказывается, что такая оптимальная информационная центральность присуща именно решениям задач локальной оптимизации выбора узлов моста по мегасети. В результате оценка вычислительной сложности задачи оптимального проектирования КоС связана с мощностью максимального подграфа полиномиально.

Затем рассматривается древовидная топология КоС, в которой узлами являются локальные подсети. Показано, что оптимальная согласованность для такой древовидной сети КоС обеспечивается путем единственного узла-моста. Это неверно для структуры с циклами. Наконец, представлены численные результаты, которые дополнительно иллюстрируют производительность модели КоС.

Согласованность была проанализирована во множестве работ для различных сетевых топологий с положительными весами ребер. Этот анализ включает ориентированные графы [2.10], кольца [2.7], фрактальные сети [2.11], а также деревья и двудольные графы [2.12].

Оптимизация согласованности сети также изучалась в системах с положительными весами ребер. Задача проектирования топологии путем добавления ребер в общую сеть рассматривалась в [2.9], где утверждена жадная эвристика, и в [2.13], где используется эвристика, основанная на выпуклой оптимизации. В [2.14] приведены аналитические результаты по оптимальному добавлению ребер для согласованности в специальных сетях. В [2.15] продемонстрированы два алгоритма аппроксимации для добавления ребер.

В [2.16, 2.17] анализировалась устойчивость КоС к каскадным отказам каналов и случайным отказам каналов [2.18]. В этих работах

производительность определяется количеством подключений к сети после сбоев. В [2.19] изучались условия устойчивости в рамках модели КоС с различными временными шкалами внутри и между подграфами, а в [2.20] определены критерии наблюдаемости и управляемости для КоС, определяемых декартовыми произведениями графов. В [2.21] исследуется скорость сходимости согласования в стохастической КоС с положительными весами ребер, когда ссылки между подграфами используются нечасто.

В [2.22] изучается производительность КоС с точки зрения нормы H_2. Однако рассматривается иная динамика. В частности, авторы требуют, чтобы добавление ребер между подграфами не изменяло динамику внутри отдельных подграфов. Вместо этого цель - изучить координацию всего КоС, где объединение нескольких подграфов влияет на динамику каждого из них.

Важным отличительным моментом исследования является расширение изучения согласованности на случай сетей с отрицательными весами ребер. Собственно отрицательные веса ребер не являются абсолютно новым компонентом модели КоС, они применены для исследования социальных сетей с обоюдным согласованием [2.6], процессов кластеризации [2.23], сходимости решений задач усреднения [2.24].

Определение «РПС» (РПС) в сетях с отрицательными весами ребер представлено в [2.25, 2.26, 2.27]. Ни одна из предыдущих работ не изучала отрицательные веса ребер в КоС с согласовательной динамикой.

В [2.28] изучалась согласованность в невзвешенных КоС. В настоящем исследовании модель КоС расширяется на взвешенные графы и допускаются отрицательные веса ребер. Также получены новые результаты о РПС в сетях с отрицательными ребрами. Кроме того, результаты об оптимальности модели КоС с одним мостовым узлом на

подграф являются новыми.

Изучается проблема оптимального проектирования сети в объединении сетей, графе, состоящем из набора непересекающихся подграфов и набора добавленных ребер между ними. Узлы подчиняются зашумленной согласовательной динамике, разрешены ребра с весами на всей числовой оси. Вводится понятие согласованности в качестве нормы H_2. Суть нормы состоит в том, что она представляет собой оценку расстояния до консенсуса, которое рассчитывается в виде дисперсии. Ставится задача о том, как соединить подграфы, выбрав один соединительный узел в каждом подграфе, чтобы полученная объединенная сеть имела оптимальную согласованность. Затем показано, что эту проблему можно решить, идентифицируя соединительный узел в каждом подграфе независимо от других узлов и подграфов. Таким образом, задача может быть решена за полиномиальное по размеру наибольшего подграфа время. Если даже в подграфе существует не единственный соединительный узел, то полученное для древовидной топологии решение также оптимально. Отдельно рассмотрены интегрированные сети с весами ребер на положительной полуоси. Для таких сетей установлены оценки минимальной и максимальной согласованности. Решения даны в аналитическом виде и проиллюстрированы примерами. Последние представляют расширение изучения величины нормы H_2 для интегрированных сетей.

1.2. Исследования в области кластеризации сенсорной сети

С развитием таких технологий, как эффективные алгоритмы оптимизации сети, агрегирование данных и маршрутизация стали ключевыми подходами к оптимизации сети. Информация для сенсорных узлов становится высокодоступной в системах на основе сенсорного узла.

Важные знания могут быть извлечены с использованием соответствующих и эффективных алгоритмов интеллектуального анализа данных, таких как подходы к росту шаблонов, такие как PSP, предложенные в [3.27], используемые для идентификации последовательных шаблонов. Алгоритм кластеризации сенсорной сети подразделяет сенсорную сеть на несколько кластеров. Внутри каждого кластера сенсорные узлы имеют общее поведение. Таким образом, конкретная система может разработать соответствующую стратегию оптимизации данных сети для сохранения существующих сенсорных узлов, каждый раз вместо того, чтобы принимать во внимание отдельные сенсорные узлы, система может распределять свои ограниченные ресурсы в определенных кластерах для оптимизации затрат и ресурсов. Ранее был разработан ряд методов кластеризации. Например, в [3.12] предложена низкоэнергетическая адаптивная иерархия кластеризации, в которой узлам датчиков разрешается оставаться в состоянии сна в течение длительного периода времени для экономии энергии, тогда как глава кластера всегда остается активным для приема и передачи данных. В [3.37] предложен алгоритм кластеризации для разнородных данных, который вводит нечеткую функцию расстояния и размерность фрактальной корреляции для идентификации кластера среди произвольной группы узлов датчиков. Рассматривая кластеризацию разнородных потоков данных, в [3.4] предложен алгоритм UK-means, минимизирующий расстояние между главным кластером и узлами датчика. Однако не удалось объединить данные с категориальными атрибутами. Метод LuMicro из [3.39] является одним из методов кластеризации, основанных на двустороннем механизме выбора для повышения качества кластера. В [3.29] предложен алгоритм SCOLPE, который принимает микрокластер и вводит кластерную гистограмму, но может кластеризовать только данные с категориальными атрибутами. Улучшая этот подход, в [3.11] предложена стратегия

«разделяй и властвуй» для кластеризации потока данных с категориальными и числовыми атрибутами.

В многомерной сети кластеризация сенсорных узлов на основе их характеристик в крупномасштабной сети является сложной задачей. На сходство узла датчика влияют различные параметры, такие как подводная среда, подвижность узла, затухание узла и другие подобные параметры. Количественное измерение большинства из этих параметров - непростая задача. Это связано с тем, что большинство вышеперечисленных атрибутов являются произвольными и дискретными, и, как правило, получены человеческими предположениями. Нечеткая теория считается наиболее подходящей мерой для решения общности и неоднозначности. Многочисленные исследователи ввели и реализовали нечеткую теорию для решения нечетких задач, такую как [3.38], основанную на функции принадлежности, [3.2] для кластерного анализа, [3.32] для распознавания образов. В нечеткой теории используются основные термины для оценки атрибутов, такие как «Хорошо», «Плохо» и другие. Существует также множество типичных форм нечетких чисел, таких как интервальные нечеткие числа, треугольные нечеткие числа, трапециевидные нечеткие числа, которые рассматриваются как общая форма нечетких чисел и которые легко обрабатываются основными оценочными терминами. Многие нечеткие систематические методы применяются в сетевых операциях и процессе сетевой кластеризации в различных областях исследований. В [3.33] предложен централизованный подход к решению проблем кластеризации. В [3.16] предложен соответствующий генетический алгоритм. В [3.21] представлен алгоритм маршрутизации максимального времени жизни UCLF. В [3.13] предложен алгоритм иерархической маршрутизации, основанный на эволюционных алгоритмах, использующих нечеткую логику для обработки неопределенностей в WSN. В [3.18] использован метод, известный как

метод Mamdani, который применен в качестве метода нечеткого вывода.

В настоящее время трудно найти соответствующие исследования, посвященные схемам кластеризации сенсорных узлов для оптимизации сети. Как уже говорилось ранее, кластеризация сенсорных узлов является промежуточным этапом в процессе оптимизации беспроводной сенсорной сети. Схема распределенной кластеризации для гетерогенных беспроводных сенсорных сетей, предложенная в [3.10], где выбирают центры кластеров на основе отношения между остаточной энергией каждого узла и средней энергией сети, а затем кластеризуют аналогичные узлы датчиков с соответствующими головками кластеров. В [3.22, 3.30] предложен метод, который состоит из двух типов узлов, основанных на начальной энергии. Оба исследования ориентированы на энергию, поэтому подходят для небольших сетей. Но оба исследования могут страдать от многочисленных проблем, таких как сложность современной беспроводной сенсорной сети, больше ресурсов будет использоваться для измерения сходства между сенсорными узлами. Следовательно, уменьшается фактор экономии затрат. Кроме того, классические методы кластеризации и алгоритмы агрегации данных не смогут обрабатывать многомерную сеть с высокой концентрацией сенсорных узлов. В дополнение к этому, мобильность узла является параметром, который приводит к смещению узлов и попаданию в другой кластер, приобретая неопределенные атрибуты. Следовательно, гетерогенность приписанных сенсорных узлов должна быть включена в процесс кластеризации. В этом обзоре литературы не представлено подходящее решение для оптимизации и кластеризации мобильной сети. В работе разрабатывается эффективный алгоритм, основанный на теории нечетких множеств, чтобы обеспечить эффективное решение, чтобы минимизировать пробел в исследованиях. Методология предложенного алгоритма разбита на разделы. Анализ структуры на основе определенной иерархии и оценка каждого уровня в

соответствии с его значимостью. После этого разработать нечеткий метод и предложить алгоритм нечеткой кластеризации для оценки атрибутов сенсорного узла. Далее оценивают точность кластеризации для определения эффективности предложенного алгоритма. Наконец, сравнение предложенного алгоритма с существующим алгоритмом кластеризации.

Частота появления многомерных данных достаточно высока, и классические методы кластеризации не подходят для многомерных мегасетей произвольной формы в беспроводной сенсорной сети. В основном это связано с тем, что методам кластеризации присуща высокая степень параметризации. Агрегирование данных, используемое традиционными методами кластеризации, неприменимо для сетей высокой размерности, имеющих произвольные формы. Более, существующие алгоритмы кластеризации не могут быть использованы для решения проблем и увеличения производительности кластеризации. В прошлом многие первоклассные ученые исследовали схемы кластеризации, основанные на нечеткой логике. В главе производится анализ этих методов. Далее подробно будет рассмотрен алгоритм на основе нечеткой логики, предназначенный для кластеризации группы, он предлагает характеризовать сенсорные узлы, основываясь на основных и второстепенных критериях. Такие алгоритмы были разработаны для группирования сенсорных узлов в многочисленные кластеры. Точность кластеризации была исследована для оценки эффективности алгоритма. Этот алгоритм также конкурирует в способности выделять различие между всеми узлами сенсоров представленной в сети. Другие алгоритмы кластеризации, существующие для агрегации данных, также рассмотрены и сравнены с предлагаемыми методами.

1.3. Особенности разработки потоковых приложений

За последние несколько лет мы стали свидетелями большого интереса к технологиям и приложениям, требующим больших объемов данных, для использования больших объемов данных и извлечения скрытой бизнес-аналитики. Сложность извлечения такой информации обычно связана с так называемыми тремя параметрами, то есть объемом, скоростью и разнообразием анализируемых данных. Конкретные сценарии работы с большими данными могут обладать одной или несколькими из этих характеристик. В частности, для тех сценариев, в которых объем и скорость имеют первостепенное значение, в игру вступают распределенные потоковые платформы. Они позволяют разрабатывать потоковые приложения, т.е. приложения, которые обрабатывают потенциально бесконечные потоки данных, непрерывно и быстро обновляя полезные выходные данные (статистику, прогнозы и т.д.). Такие приложения затем надежно выполняются распределенным образом на больших кластерах вычислительных узлов для обработки огромного объема поступающих данных. Потоковые приложения набирают обороты благодаря своему высокому бизнес-потенциалу.

К сожалению, согласно исследованию [4.9], только 14% организаций достигли полномасштабного производства своих приложений, использующих Большие данные.

У этой проблемы есть несколько причин. Среди прочих,

1) нетривиальные парадигмы проектирования на основе потоков данных, которые необходимо использовать для разработки распределенных потоковых приложений [4.1, 4.2],

2) технологическая неоднородность из-за множества существующих распределенных потоковых платформ (например, Apache Storm, Apache Spark Streaming и Apache Flink), каждый из них имеет свои собственные характеристики и API-интерфейсы [4.19].

Например, Flink предлагает механизм потоковой передачи данных, в котором входящие данные обрабатываются сразу же после их поступления, что является оптимальным выбором, когда задержка вычислений является основной проблемой производительности. Spark, напротив, обычно обеспечивает более высокую пропускную способность за счет снижения задержки, поскольку использует механизм микропакетной обработки, т.е. поток данных дискретизируется путем группировки входящих элементов данных в пакеты перед обработкой. Более того, Spark позволяет различным типам рабочих нагрузок (пакетным, потоковым, интерактивным) сосуществовать в одном приложении, чего нет в Flink. Например, Apache Kafka и Apache Storm основаны на других принципах и целях проектирования. Такая неоднородность в настоящее время создает серьезные трудности на протяжении всего жизненного цикла приложения (например, требует много времени для анализа компромиссов, реинжиниринга приложения в случае перехода на разные версии платформы).

Исследователи и практики подходят к проблеме с разных сторон, начиная от изучения и определения семантики механизмов потоковой обработки и заканчивая концептуальными и программными моделями, которые отражают основные абстракции, лежащие в основе распределенных потоковых приложений [4.1, 4.2]. В частности, модель Google Dataflow обеспечивает семантическую основу для потоковых приложений и действует как эталонная модель, отражающая возможности современных потоковых процессоров. Действительно, многие доступные движки можно сравнить с моделью Google Dataflow и Apache Beam, которая предлагает реализацию этой модели и позволяет нам программировать потоковые приложения, которые затем могут выполняться на нескольких платформах (например, Spark, Flink и т.д.). Хотя модель Google Dataflow и ее реализация в Apache Beam представляют

собой важные шаги на пути к определению общей семантики для современных потоковых приложений и решению проблемы неоднородности технологий, быстро создавать прототипы потоковых приложений на нескольких платформах с использованием Beam по-прежнему непросто, поскольку для этого по-прежнему требуются нетривиальные знания в области программирования.

Исследовательское предположение заключается в том, что Model-Driven Engineering (MDE) обладает определенным потенциалом в решении проблем, которые мы до сих пор обсуждали при проектировании и разработке современных потоковых приложений. Фактически, среди обещаний MDE можно выделить следующие:

1) упрощение понимания системы с помощью абстрактного и графического моделирования,

2) устранение технологической неоднородности,

3) повышение производительности и быстрое создание прототипов [4.7, 4.16].

Более того, потоковое приложение может быть легко представлено в виде графа потока данных, который поддается моделированию с тех пор были разработаны более продвинутые фреймворки, предлагающие больше готовых функций потоковой передачи (например, управление окнами) наряду с абстракциями более высокого уровня, направленными на упрощение разработки приложений. В частности, большинство современных фреймворков для разработки распределенных потоковых приложений предлагают функциональные модели программирования, которые значительно упрощают процесс разработки (см. раздел 4.1). Примерами таких фреймворков являются Apache Flink, Apache Spark и Apache Kafka.

В конечном счете, модель потока данных Google [4.2] представляет собой попытку объединить различные, похожие абстракции,

предоставляемые многими доступными фреймворками, во всеобъемлющую модель и соответствующие API (например, Apache Beam).

MDE был применен в нескольких прикладных контекстах для создания различных типов приложений. Например, предложен подход к разработке на основе моделей для распределенных встраиваемых автомобильных приложений [4.6], представлена методологию проектирования распределенных приложений на основе моделей [4.3], MODAClouds [4.5] - это интегрированное решение для проектирования и эксплуатации мультиоблачных приложений на основе моделей.

MDE уже применялся в контексте больших данных. Авторы [4.23] сосредоточены на моделировании и генерации кода для Hadoop MapReduce. В мире потоковой передачи аналогичную поддержку предлагает Stormgen [4.27], DSL для топологий, основанных на Storm. Что касается этих усилий, то мы стремимся предоставить язык моделирования более высокого уровня, основанный на конкретной платформе. Более того, мы реализовали этот язык как профиль UML, со всеми преимуществами, вытекающими из использования хорошо известной, стандартной и широко используемой нотации моделирования. Другим подходом MDE, ориентированным конкретно на потоковые приложения, является Juniper [4.10]. Он ориентирован на системы реального времени, работающие на платформах HPC и FPGA, но, насколько нам известно, не ориентирован на распределенные потоковые платформы.

Попытка использовать UML для моделирования приложений с большими данными была предпринята компанией DICE [4.8, 4.13].

В данном случае, однако, упор делается на качественный анализ на уровне моделирования и на генерацию инфраструктурного кода, пригодного для автоматического развертывания. Вместо этого генерация кода на уровне приложения не рассматривается. По этой причине мы

рассматриваем StreamGen и DICE как взаимодополняющие, хотя их интеграция заслуживает дальнейшего изучения и разработки.

TOREADOR - это исследовательский проект, который, как и мы, решает проблему содействия широкому внедрению решений для работы с большими данными с помощью подхода MDE [4.4, 4.11]. TOREADOR использует предварительно определенную модель приложения для работы с большими данными (аналогично шаблону), которая настраивается в соответствии с шагами предлагаемой методологии. TOREADOR также предлагает некоторую степень автоматизации для создания и развертывания DIA. Что касается StreamGen, то TOREADOR уделяет больше внимания архитектурным/структурным аспектам приложения и шагам, которым необходимо следовать на этапе проектирования. Например, это позволяет учитывать нефункциональные требования (например, производительность, защита данных) и определять варианты развертывания (например, целевого облачного провайдера для развертывания приложения). В то же время, это не дает многого с точки зрения моделирования логики приложения.

StreamGen, напротив, позволяет разработчику интегрировать логику приложения в разработанную модель и не ограничен принятием предопределенной модели для архитектуры приложения. Более того, TOREADOR развертывается как сервис и не основан на UML, в то время как StreamGen интегрирован в широко используемую среду IDE общего назначения и, как один из ее вкладов, построен поверх UML.

Apache Beam похож на StreamGen, поскольку позволяет запускать потоковые приложения на нескольких платформах, но при этом используется совершенно другой подход. Beam - это реализация модели Google Dataflow [4.2]. Он предлагает набор абстракций программирования для разработки конвейеров передачи данных, API, который реализует эти абстракции, и набор программных продуктов. Каждый программный

продукт запускает конвейеры Beam на определенных исполнительных механизмах (например, Spark, Flink, Gearpump). В то время как StreamGen предлагает подход к моделированию, основанный на UML, Beam предоставляет свои собственные API, которые, по крайней мере, по нашему опыту, нетривиальны и требуют довольно хороших знаний в области программирования. Действительно, цель StreamGen - упростить процесс вхождения в мир потоковых приложений, а также упростить и ускорить их внедрение, и мы считаем, что моделирование может сыграть ключевую роль в этом направлении.

Более того, StreamGen работает путем генерации чистого кода, в то время как Beam runners не генерирует код, а скорее интерпретирует конвейеры, закодированные Beam. Таким образом, StreamGen по-прежнему предоставляет возможность настраивать сгенерированный код, например, если кто-то хочет использовать некоторые специфические API, предоставляемые целевой платформой, что в настоящее время невозможно с Beam.

Из-за характера кампании по оценке, проводимой StreamGen, мы разработали угрозы валидности в соответствии с типичными схемами оценки, принятыми для экспериментов с конкретными случаями и исследованиями пользователей, в первую очередь с использованием фреймворка Фельдта и др. [4.12].

Возможность передачи. Возможной стратегией смягчения последствий [4.12] может быть разработка поддержки для других потоковых платформ, но с опорой на сторонних разработчиков. Это помогло бы лучше оценить не только шаги, необходимые для расширения StreamGen, но и его сложность и общее время, необходимое для добавления поддержки новой технологии.

Обоснованность надежности. Тематическое исследование TempTrack является внутренним. Чтобы уменьшить возможные ошибки,

эксперимент дополнен сравнением с моделью Google Dataflow и использован тест Yahoo Benchmark.

Что касается эксперимента с тестом Yahoo Streaming Benchmark, то одна из угроз заключается в том, что его можно считать довольно старым тестом, поскольку последнее обновление было выпущено достаточно давно. Тем не менее, он по-прежнему считается эталонным тестом, когда речь заходит о потоковых приложениях. Более того, насколько нам известно, в настоящее время не существует более нового или надежного стандарта потоковой передачи.

Распределенные потоковые приложения, то есть приложения, которые обрабатывают огромные потоки данных распределенным образом, становятся все более популярными для управления скоростью и объемом больших данных. Тем не менее, широкое внедрение интенсивной обработки данных по-прежнему ограничено используемыми нетривиальными парадигмами проектирования, которые имеют дело с неограниченностью и объемом задействованных потоков данных, а также множеством распределенных потоковых платформ, каждая из которых обладает своими собственными характеристиками и API. В работе представлен StreamGen, инструмент для проектирования на основе моделей, который упрощает разработку таких потоковых приложений и автоматически генерирует соответствующий код. StreamGen способен автоматически генерировать полностью рабочий и готовый к обработке код для различных целевых платформ (например, Apache Spark, Apache Flink). Оценка показывает, что:

1) StreamGen является достаточно общим для моделирования и генерации кода, обеспечивая сопоставимую производительность по сравнению с ранее существовавшими аналогичными и хорошо известными приложениями;

2) инструмент полностью соответствует концепциям потоковой передачи, определенным в рамках модели Google Dataflow;

3) пользователи с небольшим образованием в области компьютерных наук и ограниченным опытом работы с большими данными смогли работать с StreamGen и создавать/рефакторинговать приложение за считанные минуты.

1.4. Цель работы и задачи исследования

Целью работы является разработка математического и программного обеспечения управления процессами обработки данных в мегасетях на основе графовых моделей и системы создания потоковых приложений для обработки данных с использованием специальной метрики согласованности и интеграции логики приложений.

Для достижения цели необходимо решить следующие задачи:

- создать графовую модель синтеза мегасети из набора непересекающихся подграфов с согласованностью в качестве нормы H_2, обеспечивающую соединение подграфов с обеспечения оптимальной согласованности финальной мегасети.

- сформулировать оптимизационную задачу выбора мостов подграфов мегасети с построением соединительных ребер, обеспечивающую получение оценок минимальной и максимальной согласованности с весами ребер на положительной полуоси.

- разработать алгоритм кластеризации данных групп сенсорных узлов мегасети, обеспечивающий инвариантность к большому объему сетевых данных и динамичности сетевого местоположения узлов.

- создать архитектуру системы создания потоковых приложений для обработки данных на основе моделей потоков данных, обеспечивающую интеграцию в среду IDE поверх UML.

Дизайн исследования представлен на рис. 1.1.

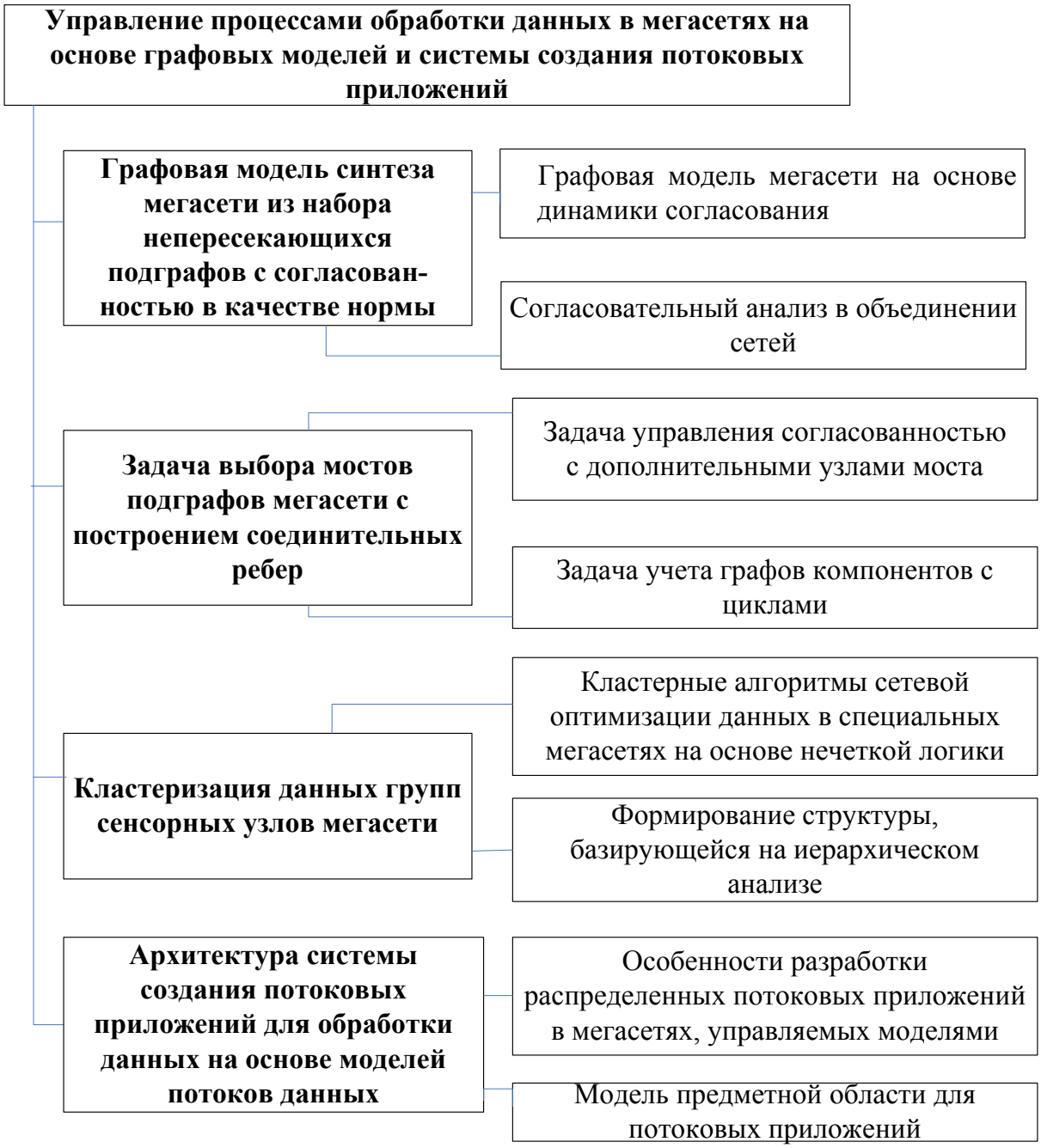


Рис. 1.1. Дизайн исследования

2. ОПТИМИЗАЦИЯ СОГЛАСОВАННОСТИ ПОДСИСТЕМ ПРИ ОБРАБОТКЕ ДАННЫХ В МЕГАСЕТЯХ

2.1. Графовое проектирование мегасети из изолированных подсетей

Изучается проблема оптимального проектирования сети в объединении сетей, графе, состоящем из набора непересекающихся подграфов и набора добавленных ребер между ними. Узлы подчиняются зашумленной согласовательной динамике, разрешены ребра с весами на всей числовой оси. Вводится понятие согласованности в качестве нормы H_2. Суть нормы состоит в том, что она представляет собой оценку расстояния до консенсуса, которое рассчитывается в виде дисперсии. Ставится задача о том, как соединить подграфы, выбрав один соединительный узел в каждом подграфе, чтобы полученная объединенная сеть имела оптимальную согласованность. Затем показано, что эту проблему можно решить, идентифицируя соединительный узел в каждом подграфе независимо от других узлов и подграфов. Таким образом, задача может быть решена за полиномиальное по размеру наибольшего подграфа время. Если даже в подграфе существует не единственный соединительный узел, то полученное для древовидной топологии решение также оптимально. Отдельно рассмотрены интегрированные сети с весами ребер на положительной полуоси. Для таких сетей установлены оценки минимальной и максимальной согласованности. Решения даны в аналитическом виде и проиллюстрированы примерами. Последние представляют расширение изучения величины нормы H_2 для интегрированных сетей.

2.2. Графовая модель мегасети

Исследуется мегасеть как множество непересекающихся подсетей $\{G_1,...,G_p\}$. Каждая подсеть $G_i=(V_i,E_i,w_i)$ представляет собой связный неориентированный граф, где:

n_i - количество узлов,

V_i - набор узлов,

E_i - набор ребер,

$w_i: E_i \to R$; $w_i \in \{-,+\}$ - функция знака веса ребер в E_i.

Примем единственность узла-моста $l_i \in V_i$ в любой подсети G_i. КоС проектируется путем интеграции подсетей через поиск узлов моста и соединение их между собой посредством ребер. Цель – соединить ребрами узлы моста $l_i \in V_i$ с узлами $l_j \in V_j$ подсети G_j. На рис. 1 показан пример пары узлов моста, соединяющих два подграфа для формирования КоС (NoN в англоязычной нотации). Магистральный граф $B=(V_B, E_B, w_B)$ - это граф, определяемый узлами моста, $V_B=\{l_1,. . . ,l_p\}$ - ребра между ними, $E_B \subseteq \{(l_i,l_j) \mid l_i,l_j \in V_B\}$ и весовой функции $w_B:E_B \to R$. Граф B может содержать как положительные, так и отрицательные ребра. Сформулируем следующее предположение о магистральном графе.

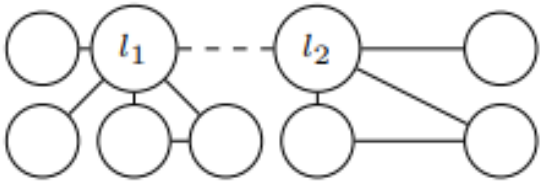


Рис. 1. Узлы моста в КоС

Предположение 2.1. Магистральный граф B сети КоС G является взвешенным связным неориентированным графом.

Объединенная сеть формально определяется как

$G=(V,E,\omega)$,

$$c\ |V| = N = \sum_{i=1}^{p} n_i\ ,$$

где

$$V = V_1 \cup ... \cup V_p$$
$$E = E_1 \cup ... \cup E_p \cup E_B$$

$$w(j,k) = \begin{cases} w_i(j,k), (j,k) \in E_i \\ w_B(j,k), (j,k) \in E_B \\ 0, \text{иначе} \end{cases}$$

Для компактности далее будем писать w(j,k) как w_{jk}.

2.2.1. Динамика согласования

Каждый узел $j \in V$ имеет скалярное состояние x_j, которое определено как

$$\dot{x}_j(t) = u_j(t) + \nu_j(t) \qquad (2.1)$$

где u_j - управляющий вход, а ν_j - белый шум с нулевым средним значением и единичной дисперсией.

Рассмотрим динамику согласования, когда каждый узел обновляет свое состояние на основе относительных состояний своих соседей в КоС. Управляющий вход задается так:

$$u_j(t) = -\sum_{k \in N_i} w_{jk} \left(x_j(t) - x_k(t) \right) \qquad (2.2)$$

где N_j - множество соседей узла j. Пусть x будет вектором состояний узла. Динамику сети G можно записать как

$$\dot{x}(t) = -Lx(t) + \nu(t) \qquad (2.3)$$

где ν - вектор возмущений, L - лапласиан группы G:

$$L_{jk} = \begin{cases} -w_{jk}, (j,k) \in E \\ \sum_{i \in N_j} w_{ji} \\ 0, \text{иначе} \end{cases}$$

Оценим количественно производительность сети по ее согласованности, которая определяется следующим образом:

$$H_C(G) = \lim_{t\to\infty} \sum_{j=1}^{N} \mathrm{var}\left(x_j(t) - \frac{1}{N} \sum_{k=1}^{N} x_k(t) \right)$$

Согласованность представляет собой общую дисперсию отклонений от среднего текущего состояния узла. Эта величина ограничена [2.7] при условии, что КоС является связной. Малая величина дисперсии свидетельствует о сильной согласованности мегасети. Если дисперсия велика, то это указывает на значительную «энтропию» в системе.

Определим согласованность как следующее выражение с использованием нормы H_2:

$$H_C(G) = \mathrm{tr}\left(\int_0^{\infty} e^{-Lt} P e^{-Lt} dt \right)$$

где $P = I - \frac{1}{N} \mathbb{Z}\mathbb{Z}^T$,

$\mathbb{Z}$ - единичный вектор.

Для псевдообратной матрицы L^+ к положительно полуопределенной матрице L с нулевым простым собственным значением имеет место следующее равенство [2.7, 2.25, 2.29]

$$H_C(G) = \frac{1}{2} \mathrm{trace}\left(L^+ \right).$$

2.2.2. Постановка проблемы

Для любого связного графа величина $H_C(G)$ конечна. Эта величина зависит от топологии мегасети. Если минимизировать $H_C(G)$, то в полученной КоС будет минимальная «сетевая энтропия» и наилучшая согласованность. Рассмотрим эту оптимизационную задачу в ситуации, когда множество подграфов задано. Далее предположим, что топология магистрального графа определена априори, т.е. заранее определено, какие

подграфы должны быть связаны с какими другими подграфами. Задача по сути состоит в поиске для каждой подсети (подграфе) единственного узла-моста, через который будет происходить взаимодействие с другими подсистемами. Оптимизационная задача сводится к оптимизации согласованности финальной мегасети. Представим лапласиан $L(l_1,...,l_p)$ финальной мегасети как функцию от набора оптимальных узлов-мостов. Таким образом, запишем эту матрицу Лапласа как $L(l_1,...,l_p)$. Задача оптимизации узлов моста может быть формализована следующим образом:

$$\min_{l_i \in V_i,\, i=1,...,p} H_C(G) = \frac{1}{2}\,\mathrm{trace}\left(L\left(l_1,...l_p\right)^{+}\right) \qquad (2.5)$$

2.3. Расстояние по сопротивлению и отрицательные ребра

Известно, что существует взаимозависимость между между и РПС (Resistance distance – расстояние по сопротивлению) согласованностью в электротехнике. Применим эту взаимозависимость при решении задачи (2.5).

Пусть

$G_g=(V_g,E_g,w_g)$ - произвольный взвешенный граф,

$|V_g|=n$.

Пусть также

E_g^+ - подмножество ребер E_g с положительными весами,

E_g^- - подмножество ребер с отрицательными весами,

$G_g^+ = (V_g,\ E_g^+, w_g^+)$,

$G_g^- = (V_g,\ E_g^-, w_g^-)$,

$G_g^+ \cup G_g^- = G_g$,

w_g^+, w_g^- - веса для редукции сужением w_g на ребра с соответствующим знаком веса.

Пусть L_g - взвешенная матрица Лапласа группы G_g.

2.3.1. Концепция эффективного сопротивления (ЭфС)

Для ненаправленных графов G_g с положительными весами ребер исследуем электрическую сеть с топологией G. Важно, что $\frac{1}{w_g(u,v)}$ - сопротивление любого ребра. Потенциальная близость $r(u,v)$ для ребер u и v при проходящем токе в 1 А ($u, v \in V_g$) определяется через L_g^+ как [2.31]:

$$r(u,v) = \left(L_g^+\right)_{uu} - \left(L_g^+\right)_{vv} + 2\left(L_g^+\right)_{uv} \tag{2.6}$$

где

$\left(L_g^+\right)_{ij}$ - (i,j)-й элемент L_g^+.

Полное ЭфС G_g [2.30]:

$$\Omega_{G_g} = \frac{1}{2}\sum_{u,v\in V_g} r(u,v) = \frac{1}{2}\sum_{u<v\in V_g} r(u,u)$$

$$\Omega_{G_g} = n\sum_{u\in V_g}\left(L_g^+\right)_{uu} - \mathbb{Z}^T L_g^+ \mathbb{Z} = n \bullet \mathrm{trace}\left(L_g^+\right)$$

поскольку

$$H_C(G_g) = \frac{\Omega_{G_g}}{2n} \tag{2.7}$$

Предположение 2.2 (E [2.30]). Пусть $G_g=(V_g,E_g,w_g)$ можно разделить на подграфы A и B с положительными весами ребер. При этом существует единственная вершина x, принадлежащая подграфам A и B. РПС между любыми $a \in A$ и $b \in B$:

$$r(a,b)=r(a,x)+r(x,b). \tag{2.8}$$

Предположение 2.3 (D [2.30]). Пусть для графа $G_g=(V_g,E_g,w_g)$ (у ребер – положительные веса) $\forall u,v \in V_g$, $d(u,v) \geq r(u,v)$ (здесь взвешенная

сумма d(u,v) весов ребер вдоль кратчайшего пути между u и v). При этом строгое равенство достигается только для единственного пути.

2.3.2. Неположительные веса ребер

Концепция ЭфС в [2.25, 2.26] распространена на сети с отрицательными весами ребер. Назовем графы с неположительными весами, в которых РПС всегда конечны, графами ограниченного сопротивления (ОГС).

Определение 2.1. $G_g=(V_g,E_g,w_g)$ будем называть ограниченным графом сопротивлений (ОГС), если:

- G_g - связен;

- $\forall(u,v)\in E_g^-$ ребро в составе цикла;

- в G_g нет циклов с более чем одним $(u,v)\in E_g^-$;

- $\forall(u,v)\in E_g^-: \left|w_g(u,v)\right| < \dfrac{1}{r_{G_g^+}(u,v)}$, $r_{G_g^+}(u,v)$ РПС для u и v в G_g^+.

Пример ОГС: граф с положительными весами ребер.

Промежуточное утверждение [2.26]. (2.6) дает РПС между любой парой узлов, причем оно всегда положительно для ОГС. Кроме того, матрица Лапласа является положительно полуопределенной с простым собственным значением 0 и собственным вектором 1 [2.26]. Таким образом, в силу (2.4) соотношение (2.7) выполняется и для ОГС.

Теперь покажем, что предположение 2.1 верно для ОГС.

Предположение 2.4. Пусть граф G_g - ОГС. Пусть можно разделить на подграфы A и B с положительными весами ребер. При этом существует единственная вершина x, принадлежащая подграфам A и B. РПС между любыми $a\in A$ и $b\in B$ снова вычисляется как:

$$r(a,b)=r(a,x)+r(x,b). \qquad (2.9)$$

Доказательство. G_g^+ - связный граф (следует из определения ОГС). Тогда по предположению 2.2 верно:

$$\begin{aligned} &r(a,b)-r(a,x)-r(x,b)=\bar{L}^+_{aa}+\bar{L}^+_{bb}-2\bar{L}^+_{ab}- \\ &-\left(\bar{L}^+_{aa}+\bar{L}^+_{xx}-2\bar{L}^+_{ax}\right)-\left(\bar{L}^+_{xx}+\bar{L}^+_{bb}-\bar{L}^+_{xb}\right)= \\ &=-2\bar{L}^+_{ab}-2\bar{L}^+_{xx}+2\bar{L}^+_{ax}+2\bar{L}^+_{bx}=0 \end{aligned} \qquad (2.10)$$

Здесь L - лапласиан от G_g^+.

Важно, предположение 2.2 верно и сохранением возможности разбиения сети на подсети A и B ($A\cap B=\{x\}$), существует такое соответствующее ребро e^+ для добавления.

Пусть $e^+=(j,k)$ с весом $w_{jk}=\alpha$. Пусть u - вектор с $u(j)=\sqrt{\alpha}$, $u(k)=-\sqrt{\alpha}$, $u(r)=0(r\neq j, r\neq k)$.

Пусть для $G'=G_g^+\cup\left\{e^+\right\}$ $\left(\bar{L}+uu^T\right)$ - его лапласиан. Пусть $M=\bar{L}^+uu^T\bar{L}^+$ и $r'(u,v)$ как РПС над графом G''.

Далее к (2.10) применим формулу Шермана-Моррисона [2.37]:

$$\left(\bar{L}+uu^T\right)^+=\bar{L}^+-\frac{\bar{L}^+uu^T\bar{L}^+}{1+u^T\bar{L}^+u}$$

чтобы установить, что

$$\begin{aligned} &r'(a,b)-r'(a,x)-r'(x,b)=-2\bar{L}^+_{ab}-2\bar{L}^+_{ax}+2\bar{L}^+_{ax}+ \\ &+2\bar{L}^+_{bx}-\frac{-2M_{ab}-2M_{xx}+2M_{ax}+2M_{bx}}{1+\alpha\left(\bar{L}^+_{jj}+\bar{L}^+_{kk}-2\bar{L}^+_{jk}\right)}=0 \end{aligned} \qquad (2.11)$$

Из (2.10) и (2.11) получаем:

$-2M_{ab}-2M_{xx}+2M_{ax}+2M_{bx}=0$.

Пусть $e^-=(j,k)$ - ребро в E_g^- с весом ребра $w_{jk}=-\alpha$. Пусть G'' будет графом, образованным добавлением e^- к G_g^+.

Для u: $u(j)=\sqrt{\alpha}$, $u(k)=-\sqrt{\alpha}$, $u(r)=0(r\neq j, r\neq k)$. Пусть для G''

$\left(\bar{L}+uu^T\right)$ - его лапласиан. Пусть r″(u,v) - РПС над G″. При этом верно равенство

$$\left(\bar{L}-uu^T\right)^+=\bar{L}^+ +\frac{\bar{L}^+uu^T\bar{L}^+}{1-u^T\bar{L}^+u}$$

После учета этого равенства и повторного применения формулы Шермана-Моррисона получаем

$$r''(a,b)-r''(a,x)-r''(x,b)=-2\bar{L}^+_{ab}-2\bar{L}^+_{xx}+2\bar{L}^+_{ax}+$$
$$+2\bar{L}^+_{bx}+\frac{-2M_{ab}-2M_{xx}+2M_{ax}+2M_{bx}}{1+\alpha\left(\bar{L}^+_{jj}+\bar{L}^+_{kk}-2\bar{L}^+_{jk}\right)}$$

Из (2.10)

$$-2\bar{L}^+_{ab}-2\bar{L}^+_{xx}+2\bar{L}^+_{ax}+2\bar{L}^+_{bx}=0$$

Из (2.11)

$$-2M_{ab}-2M_{xx}+2M_{ax}+2M_{bx}=0$$

Тогда $r''(a,b)-r''(a,x)-r''(x,b)=0$, поэтому (2.9) выполняется для G″. Отсюда следует, что если (2.9) выполняется для C_g^+, оно также выполняется при добавлении каждого ребра из E^-. Таким образом, (2.9) выполняется для любого ОГС.

2.4. Согласовательный анализ в объединении сетей

В этом разделе описано некомбинаторное решение задачи (2.5) для формирования КоС по предопределенному множеству подграфов с выполнением оптимальности согласованности.

2.4.1. КоС с оптимальной согласованностью

Определение 2.2. Рассмотрим граф. Индекс сопротивления узла $v\in V$ графа $G=(V,E,w)$ определим как

$$C(v)=\sum_{u\in V,\,u\neq v} r(u,v) \qquad (2.12)$$

Обозначим индекс сопротивления для узлов в подграфе G_i как $C_i(v_i)$ (узел v_i подграфа G_i).

Утверждение 2.1. Для графа G из p подграфов $G_i=(V_i,E_i,w_i)$, i=1,...,p, пусть G будет КоС, сформированным из этих подграфов с использованием заданной топологии опорного графа с одним мостовым узлом $l_i \in V_i$ на подграф. Предположим, что опорный граф и все его подграфы являются ОГС. Тогда условие $l_i = \underset{v \in V_i}{\operatorname{argmin}}\, C_i(v) \;\; \forall G_i$ достаточно для минимальности согласованности G.

Достаточным условием того, что G является ОГС, является выполнение следующих двух условий:

- опорный граф есть ОГС;

- $\{G_i\}_i$ есть ОГС.

Утверждение 2.2 (следует из предположения 2.4). Пусть G=(V,E,w) - КоС как ОГС. Для узлов $u,v \in V,\ u \in V_i,\ v \in V_j,\ i \neq j$, РПС равно:

$$r(u,v)= r(u,l_i)+r(l_i, l_j)+ r(l_j,v).$$

Утверждение 2.3. Пусть G=(V,E,w) - КоС как ОГС. Тогда выражение (13) определяет согласованность G:

$$H_C(G) = \frac{1}{2N}\left(\sum_{i=1}^{p} 2n_i H_C(G_i) + \right.$$
$$\left. + \sum_{i=1}^{p}\sum_{j=i+1}^{p} |V_i||V_j| r\left(l_i,l_j\right) + \sum_{i=1}^{p} |V - V_i| C_i(l_i) \right) \qquad (2.13)$$

Доказательство: Пусть G=(V,E,w) - КоС, состоящий из p подграфов. Как следствие из предположения 2.4:

$$\Omega_G = \frac{1}{2}\sum_{u,v\in V} r(u,v) = \sum_{i=1}^{p}\frac{1}{2}\sum_{u,v\in V_i} r(u,v) + \\ +\sum_{i=1}^{p}\sum_{g=i+1}^{p}\left(\sum_{u\in V_i}\sum_{v\in V_j} r(u,v)\right) = \sum_{i=1}^{p}\Omega_i + \sum_{i=1}^{p}\sum_{g=i+1}^{p}\left(\sum_{u\in V_i}\sum_{v\in V_j} r(u,v)\right) \quad (2.14)$$

Каждая компонента $\sum_{i=1}^{p}\sum_{g=i+1}^{p}\left(\sum_{u\in V_i}\sum_{v\in V_j} r(u,v)\right)$ в (14) может быть переписана в следующем виде:

$$\sum_{u\in V_i}\sum_{v\in V_j} r(u,l_i)+r(l_i, l_j)+ r(l_j,v) \quad (2.15)$$

как отмечено в утверждении 2.2. Полученная двойная сумма может быть дополнительно упрощена с использованием $|V_j|C_i(l_i)+|V_i||V_j|r(l_i,l_j)+|V_i|C_j(l_j)$. Выражение для Ω_G становится следующим:

$$\Omega_G = \sum_{i=1}^{p}\Omega_i + \sum_{i=1}^{p}\sum_{j=i+1}^{p}\left(|V_j|C_i(l_i)+|V_i||V_j|r(l_i,l_j)+|V_i|C_j(l_j)\right)$$

$$= \sum_{i=1}^{p}\Omega_i + \sum_{i=1}^{p}\sum_{j=i+1}^{p}\left(|V_i||V_j|r(l_i,l_j)\right) + \sum_{i=1}^{p}\sum_{j=i+1}^{p}\left(|V_j|C_i(l_i)+|V_i|C_j(l_j)\right)$$

При увеличении $\sum_{i=1}^{p}\sum_{j=i+1}^{p}\left(|V_j|C_i(l_i)+|V_i|C_j(l_j)\right)$ упрощается $\sum_{i=1}^{p}\left|V - V_i\right|C_i(l_i)$. Следовательно,

$$\Omega_G = \sum_{i=1}^{p}\Omega_i + \sum_{i=1}^{p}\sum_{j=i+1}^{p}\left(|V_i||V_j|r(l_i,l_j)\right) + \sum_{i=1}^{p}\left|V - V_i\right|C_i(l_i)$$

Наконец, используем (2.7), чтобы заменить Ω_i на $2n_iH_C(G_i)$ и разделить на 2N, чтобы получить (2.13).

Теперь закончим доказательство Утверждения 2.2.

Доказательство. По утверждению 2.3 для заданного набора подграфов и архитектуры объединенной сети значение $H_C(G)$ зависит от узлов моста только в компоненте $C_i(l_i)$. Отсюда $l_i = \underset{v\in V_i}{\operatorname{argmin}}\, C_i(v)$

независимо по каждому $C_i(l_i)$ дает минимизацию согласованности G.

Вычислительная сложность поиска единственного лидера в графе с n узлами в системах типа «лидер-последователи» равна $O(n^3)$ [2.33]. В случае (2.5) таких задач – p. Тогда вычислительная сложность решения (2.5) имеет оценку $O(p(\max\limits_{i=1,...,p} n_i))^3$.

2.4.2. Границы для $H_C(G)$ для топологий с единичными весами ребер

Следствие 2.1. Для топологий КоС G с единичными весами ребер, $|V_i|=n$; $i=1,...p$, $\min H_C(G)$ дается формулой:

$$H_C(G) \geq \frac{1}{2N}(2n^2(p-1)+p(n-1)+2p(p-1)(n-1)) \qquad (2.16)$$

Доказательство. В (2.13) слагаемое $\sum\limits_{i=1}^{p} 2n_i H_C(G_i)$ минимизируется, когда G_i является полным графом, и в этом случае $\Omega_{Gi}=n-1$ [2.34]. Слагаемое $\sum\limits_{i=1}^{p}\sum\limits_{j=i+1}^{p}\left(|V_i||V_j|r(l_i,l_j)\right)$ минимизируется, когда граф является полным, и, следовательно, $r(l_i,l_j)=\frac{2}{p}$ для всех $l_i \in V_i$, $l_j \in V_j$. Наконец, для полного G_i, $C_i(l_i)=(n-1)*(2/n)$ для любого $l_i \in V_i$. Комбинируя эти наблюдения и отмечая, что $|V-V_i|=n(p-1)$, получаем согласованность для КоС, сформированного из множества полных подграфов, связанных с полным основным графом:

$$H_C(G)=\frac{1}{2N}\left(\sum_{i=1}^{p}(n-1)+\frac{2}{p}\sum_{i=1}^{p}\sum_{j=i+1}^{p}n^2+\sum_{i=1}^{p}n(p-1)(n-1)\frac{2}{n}\right)=$$

$$=\frac{1}{2N}\left(p(n-1)+2n^2(p-1)+2p(p-1)(n-1)\right)$$

Поскольку G определен как имеющий минимальную

согласованность, любой другой КоС G′ с $|V_i|=n$, $|V_B|=p$ должен иметь $H_C(G')>H_C(G)$.

Отметим, что нижняя граница согласованности составляет O(n+p), что означает, что она линейно возрастает как по количеству узлов в подграфе, так и по количеству подграфов. Сравниваем это с согласованностью полного графа с N=np узлами, что обеспечивает нижнюю границу согласованности всех возможных графов размера N как O(1).

Следствие 2.2. Верхняя граница согласованности любого КоС G со всеми весами ребер, равными 1, состоящего из p подграфов с $|V_i| =n$ для i=1,...,p:

$$H_C(G) \le \frac{1}{2N}(2n^2(p-1)+p(n-1)+2p(p-1)(n-1))$$

Доказательство. Следуя рассуждениям, аналогичным доказательству следствия 2.1, рассмотрим выражение в (2.13). Слагаемое $\sum_{i=1}^{p} 2n_i H_C(G_i)$ максимально, когда подграфы имеют максимальное суммарное ЭфС. Это соответствует случаю, когда все подграфы являются графами путей [2.31], так что

$$\Omega_{G_i} = \sum_{i=1}^{p}\sum_{j=i+1}^{p}(j-i) = \frac{n^3-n}{6} \qquad (2.17)$$

Далее, имеем выражение:

$$\sum_{i=1}^{p}\sum_{j=i+1}^{p}\left(|V_i||V_j|r(l_i,l_j)\right) = n^2\sum_{i=1}^{p}\sum_{j=i+1}^{p} r(l_i,l_j) \qquad (2.18)$$

которое максимизируется путем выбора магистрального графа с максимальным общим ЭфС, т.е. графа пути. Окончательно, заметим, что $C_i(l_i)$ максимально, когда каждый узел моста выбран в качестве конечной точки своего подграфа пути, так что $C_i(l_i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$.

Исходя из этого, вычислим $H_C(G)$ для КоС, состоящего из подграфов пути с графом магистрали пути.

$$H_C(G) = \frac{1}{2N}\left(\sum_{i=1}^{p} 2n_i \frac{1}{12n} n(n^2 - 1 + \sum_{i=1}^{p} \sum_{j=i+1}^{p} (j-i)n^2 + \right.$$

$$\left. + \sum_{i=1}^{p} n(p-1)\frac{n(n-1)}{2} \right) = \frac{pn(n^2-1)}{12N} + \frac{pn^2(p^2-1)}{12N} + \frac{pn^2(n-1)(p-1)}{4N}$$

Поскольку согласованность G максимизирована, любая другая КоС G′ с $|V_i|=n$ и $|V_B|=p$ должна иметь $H_C(G)>H_C(G')$.

Верхняя граница согласованности для всех КоС составляет $O(np^2+n^2p)$. Сравним это с согласованностью графа путей размера $N=np$, который обеспечивает верхнюю границу согласованности всех возможных графов размера N, и равна $(N^2-1)/6$. Таким образом, по мере добавления большего количества узлов согласованность КоС растет медленнее, чем у графа путей.

Учитывая значительную разницу между верхней и нижней границами согласованности КоС, можно видеть, что оптимизация КоС, как с точки зрения топологии магистрали, так и узлов моста, может оказать существенное влияние на ее согласованность.

2.5. Согласованность с дополнительными узлами моста

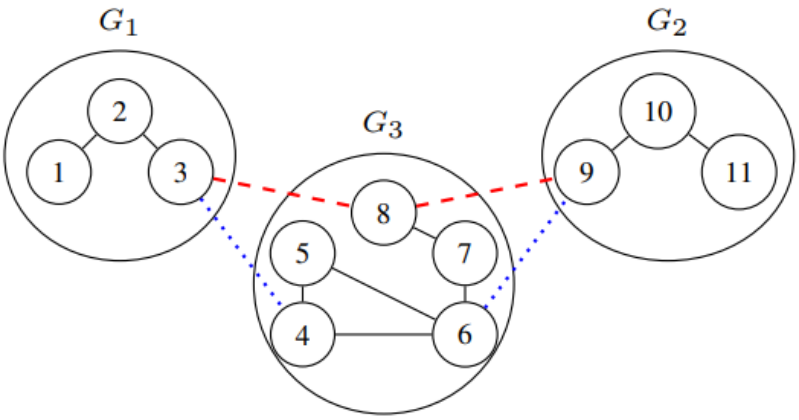
В модели, описанной в разделе 2.2, подграфы КоС соединяются через один узел моста в каждом подграфе. Теперь исследуем влияние нескольких узлов моста в одном подграфе на согласованность КоС. В частности, рассмотрим вопрос о том, улучшит ли согласованность использование нескольких узлов моста на подграф.

Когда на подграф приходится несколько узлов моста, опорный граф может не удовлетворять предположению 2.

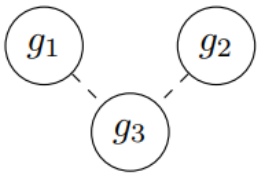
Чтобы сравнить КоС со связными и несвязными опорными графами, введем концепцию компонента графа. Пусть G - КоС, образованный из

подграфов $G_1,...,G_p$, с опорным графом B. Компонентный граф графа G определяется как $C=(V_C,E_C)$, где $V_C=\{1,...,p\}$ с каждым узлом, соответствующим подграфу в G, и $E_C=\{(i_1, j_1)...(i_R, j_R)\}$ - множество компонентов рёбер графа с $|E_C|=R$. Ребро существует тогда и только тогда, когда $(u,v)\in E_B$, где $u \in V_{i_r}$ и $v \in V_{j_r}$. Существует ребро $(u,v)\in E_B$, где $u \in V_{i_r}$ и $v \in V_{j_r}$.

На рис. 2.2 представлены два КоС на базе одного графа компонентов. Рис. 2.2а - КоС G_A и G_B, из одних и тех же трех подграфов. Магистральный граф для G обозначен штриховыми линиями, а магистральный граф для G_B обозначен пунктирными линиями; рис. 2.2б - граф компонентов для G_A и G_B.



а)



б)

Рис. 2.2. Различные варианты формирования КоС

Рассмотрим модифицированный вариант задачи (2.5), в котором задан граф компонентов, т.е. известно, какие подграфы должны быть связаны друг с другом. Цель состоит в том, чтобы определить, какой узел (узлы) моста в каждом подграфе следует использовать для соединения подграфов, чтобы оптимизировать согласованность G. Необходимо, чтобы

каждое ребро в графе компонентов соответствовало ровно одному ребру в КоС. В этом случае матрица Лапласа КоС G, которую необходимо построить, может быть определена как функция узлов моста, соответствующих R ребрам в опорном графе, т.е. $L = L\left(\left(l_i^1, l_j^1\right),...,\left(l_i^R, l_j^R\right)\right)$. Эта обновленная задача может быть выражена следующим образом:

$$\min_{\substack{(i_r,j_r)\in E_C \\ l_i^r\in V_{i_r},\, l_j^r\in V_{i_j},\, r=1,...R}} H_C(G) = \frac{1}{2}\operatorname{trace}\left(\mathbb{F}\right) \tag{2.19}$$

где $\mathbb{F} = \left(L\left(\left(l_i^1, l_j^1\right),...,\left(l_i^R, l_j^R\right)\right)\right)^+$

2.5.1. Древовидная структура

Путь граф компонентов C имеет древовидную структуру и у ребер – единичный вес. Покажем, что нахождение единственного оптимального узла-моста для каждого подграфа дает оптимальную производительность вследствие того, что решения (2.5) и (2.19) идентичны.

Утверждение 2.4. Если:

- $B=(V_B, E_B, w_B)$ - древовидный остовный граф;

- G – КоС, в котором узлы-мосты l_i;

- $l_i = \underset{v\in V_i}{\operatorname{argmin}} C_i(v)$;

- G' отличается от G тем, что существует подграф с двумя узлами-мостами;

- у ребер G и G' – единичный вес,

тогда $H_C(G) < H_C(G')$.

Доказательство. Пусть для маршрута $l_{i\to j}$ от G_i к G_j есть узел-мост в V_i (конец маршрута). Например, на рис. 2.2а узел 8 является узлом моста в G_3, который является конечной точкой путей от G_3 до G_1 и от G_3 до G_2. Таким образом, узел 8 можно обозначить как $l_{3\to1}$ и $l_{3\to2}$. С помощью этих

обозначений теперь можно обобщить (2.13), чтобы учесть несколько узлов моста в каждом подграфе, как:

$$H_C(G_m) = \frac{1}{2N}\left(\sum_{i=1}^{p} 2n_i H_C(G_i) + \sum_{i=1}^{p}\sum_{j=i+1}^{p} \left(\left|V_j\right| C_i\left(l_{i\to j}\right) + \right.\right.$$
$$\left.\left. + \left|V_i\right|\left|V_j\right| r(l_{i\to j}, l_{j\to i}) + \left|V_i\right| C_j\left(l_{j\to i}\right) \right) \right) \qquad (2.20)$$

где G_m - КоС с опорным графом, который может не удовлетворять предположению 2.2.

Пусть множество узлов мостов и опорных ребер в G' есть $B'=(V_B', E_B', w_B)$. Рассмотрим два подграфа G_i и G_j в G'. Поскольку граф компонентов является деревом, существует ровно один путь между G_i и G_j. Если в каждом подграфе на пути из G_i в G_j в G' имеется единственный узел моста, т.е. если $(l_i, l_j) \in E_B'$, то по предположению 2.2 $r(l_{i\to j}, l_{j\to i}) = d_C(l_{i\to j}, l_{j\to i})$, где $d_C(l_{i\to j}, lj\to i)$ обозначает графовое расстояние над компонентным графом C. Если же на этом пути есть хотя бы один подграф с кратным мостом узлов, то $r(l_{i\to j}, l_{j\to i}) > d_C(l_{i\to j}, l_{j\to i})$. Таким образом, $\sum_{i=1}^{p}\sum_{j=i+1}^{p} \left|V_i\right|\left|V_j\right| r(l_{i\to j}, l_{j\to i})$ строго больше в G', чем в G.

Далее рассмотрим сумму

$$\sum_{i=1}^{p}\sum_{j=i+1}^{p} \left(\left|V_j\right| C_i\left(l_{i\to j}\right) + \left|V_i\right| C_j\left(l_{j\to i}\right) \right).$$

В G для i,j=1,…,p, выполнено

$$C_i(l_{i\to j}) = \arg\min_{v\in V_i} C_i(v) \text{ и } C_j(l_{j\to i}) = \arg\min_{v\in V_j} C_j(v).$$

Следовательно, в G' для i,j=1,...,p $C_i'(l_{i\to j}) \ge C_i(l_{i\to j})$ и $C_j'(l_{j\to i}) \ge C_j(l_{j\to i})$, и, что самое главное, $\sum_{G'} \ge \sum_{G}$. Далее, $\sum_{i=1}^{p} 2n_i H_C(G_i)$ одинакова для G и G'. Отсюда следует, что $H_C(G) < H_C(G')$.

Утверждение 2.4 доказано.

Хотя утверждение 2.4 верно, если единственный узел моста в

каждом подграфе имеет минимальный индекс сопротивления, если это не так, то КоС с несколькими узлами моста на подграф может иметь лучшую согласованность. Пример с тремя подграфами показан на рис. 2.2a, где G имеет один неоптимальный мостовой узел в каждом подграфе, а $H_C(G)=6,2727$. G′ имеет тот же граф компонентов, что и G, но имеет два моста в подграфе G_3.

2.5.2. Графы компонентов с циклами

Из рис. 2.3 видно, что когда граф компонентов включает циклы, КоС с несколькими узлами моста на подграф может превзойти оптимальный КоС с одним узлом моста на подграф.

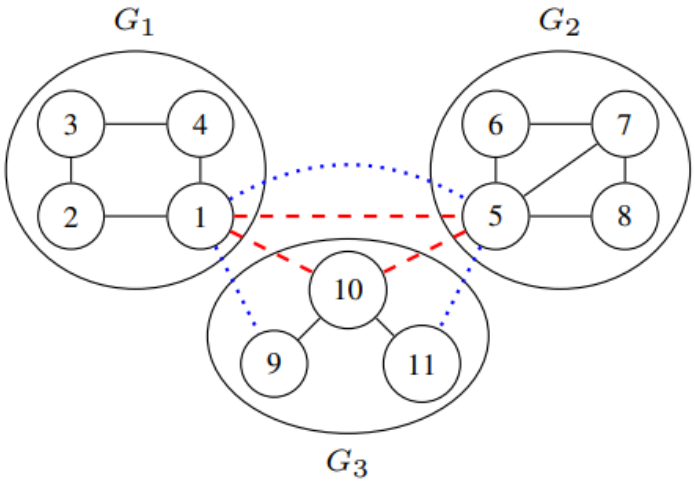


Рис. 2.3. Пример КоС, где несколько узлов моста в одном подграфе уменьшают согласованность. Опорные ребра в G отмечены пунктиром; ребра бэкбона в G’ отмечены точками

2.5.3. Аналитические примеры

В этом разделе представлены аналитические примеры, основанные на модели КоС в разделе 2.2 и результате в утверждении 2.2. Сначала представлен пример, в котором, когда мостовые узлы уже заданы, вместо этого согласованность оптимизируется по опорному графу. Рассмотрим КоС, где опорный граф представляет собой дерево. Согласованность КоС

может быть получена из утверждения 2.2 с помощью предположения 2.3 для замены $r(l_i,l_j)$ взвешенным графовым расстоянием $d(l_i,l_j)$ между узлами моста, чтобы получить:

$$H_C(G)=\frac{1}{2N}\left(\sum_{i=1}^{p}2n_iH_C(G_i)+\right.$$
$$\left.+d(l_i,l_j)\left|V_i\right|\left|V_j\right|+\sum_{i=1}^{p}\left|V-V_i\right|C_i\left(l_j\right)\right) \qquad (2.21)$$

Следствие 2.3. Пусть G - КоС, образованный из подграфов $G_1,\ldots,G_p$, с мостовыми узлами $l_1,...,l_p$ соответственно, где опорный граф B представляет собой дерево, и все веса опорных ребер равны 1. Далее, пусть подграфы являются ОГС. Тогда оптимальная топология магистрали представляет собой звездный граф с узлом моста $l_c\in V_c$ в центре звезды, где $V_c\in\text{argmax}_i\,|V_i|$.

Доказательство. Сначала докажем, что оптимальная магистральная топология представляет собой звездный граф. Это играет роль только в сумме $\sum_{i=1}^{p}\sum_{j=1}^{p}d(l_i,l_j)\left|V_i\right|\left|V_j\right|$ в (21). Поскольку древовидный граф размером p имеет p−1 ребер, имеется p−1 пара узлов l_i, l_j, где $d(l_i,l_j)=1$. Остальные $\frac{(p-1)(p-2)}{2}$ пары узлов моста имеют $d(l_i,l_j)>1$. В звездном графе остальные пары узлов моста имеют $d(l_i,l_j)=2$, тогда как в любой другой древовидной топологии есть по крайней мере одна пара узлов моста с $d(l_i,l_j)>2$. Таким образом, сумма минимизируется в звездном графе.

Далее рассмотрим оптимальное расположение подграфов в топологии «звезда». Предположим, что подграфы нумеруются в порядке уменьшения размера набора вершин, так что $|V_1|\geq|V_2|\geq\ldots\geq|V_p|$. Задача минимизации:

$$\sum_{i=1}^{p}\sum_{j=1}^{p}d(l_i,l_j)\left|V_i\right|\left|V_j\right|=\sum_{i=1}^{p}\left|V_C\right|\left|V_i\right|+2\sum_{i=1}^{p}\sum_{j=1}^{p}\left|V_i\right|\left|V_j\right|$$

Минимум достигается при $V_c = \text{argmax}_i |V_i|$.

Утверждение доказано.

Перейдем от деревьев к циклам. Будем исследовать КоС, в котором преобладают циклы – как B, так и G_i. При этом размер таких циклов строгофиксирован величиной n. Кроме того, каждый цикл – это ОГС совпадающими весами ребер, равными единице. Иных вариантов для положительно взвешенных ребер нет. Далее получена оценка снизу для величины согласованности мультицикла.

Следствие 2.4. Рассмотрим граф G, содержащий n подсистем $G_1,...,G_n$ как КоС. При этом каждая подсистема есть цикл с узловой мощностью n. Далее, пусть и в КоС, и в каждой подсистеме единственное ребро имеет вес $\frac{-1}{\rho}$, где $\rho > n-1$. Вес остальных ребер – единичный. Обозначим через G′ КоС, в котором n подсистем маршрутов $G'_1,...,G'_n$. Пусть все веса ребер в G' равны 1. Далее предположим, что узлы-мосты в G и G' выбраны как решения задачи (2.5). Тогда при нечетном n $H_C(G) \geq H_C(G')$ для всех $\rho > n-1$. Далее,

$$\lim_{\rho \to \infty} H_C(G) = H_C(G')$$

Доказательство. Сначала заметим, что по предположению как для G, так и для G′, $\Omega_{Gi} = \Omega_B$. Для G,

$$\Omega_{G_i} = \sum_{i=1}^{n-1} i(n-1)\frac{\rho - n + i + 1}{\rho - n + 1} = \frac{(n^3-1)(2\rho - n + 1)}{6(2\rho - 2n + 2)} \tag{2.23}$$

$$\Omega_{G'_i} = \frac{1}{6}(n^3 - n) \tag{2.24}$$

Заметим, что (2.23) больше, чем (2.24) при $\rho > n-1$. Для цикла, являющегося ОГС со всеми единично-положительными весами ребер с рядомстоящими узлами $(j,k) \in E^+$ с минимальным РПС, выполнено $r(j,k) = (\rho - n + 2)/(\rho - n + 1)$.

Ясно, что для несоседей (u,v) выполнено $r(u,v)>r(j,k)$. Для узлов $(u,v)\in E^-$, являющихся отрицательной парой, РПС равно $r(u,v)=\frac{\rho(n-1)}{\rho-n-1}$.

Тогда можем сформулировать набор условий, формирующих достаточное условие минимальности любой подсистемы в G:

1. Узел-мост l_i наиболее удален от отрицательного ребра;
2. Максимальное расстояние от l_i до границы графа $n/2$;
3. $r(e^-)$ не входит в (12) для $C_i(l_i)$;
4. l_i - промежуточная точка отрицательного ребра.

Тогда минимальный индекс сопротивления каждого подграфа в G равен

$$C_i(l_i)=2\sum_{i=1}^{\frac{n-1}{2}} i\frac{\rho-n+i+1}{\rho-n+1}=\frac{(n^2-1)(3\rho-2n+3)}{4(3\rho-3n+3)} \qquad (2.25)$$

Оптимальным мостовым узлом l_i' для каждого подграфа G′ является узел в центре маршрута [2.35]. Тогда индекс минимального сопротивления каждого подграфа в G' будет равен:

$$C_i(l_i')=2\sum_{i=1}^{\frac{n-1}{2}} 2i=\frac{1}{4}(n^2-1) \qquad (2.26)$$

Снова (2.25) больше, чем (2.26) для $\rho>(n-1)$. Из этого следует, что $H_C(G)\geq H_C(G')$ для всех $\rho>n-1$. Ясно, что при $\rho\to\infty$, (2.23) сходится к (2.24) и (2.25) сходится к (2.26). Таким образом, $H_C(G)$ сходится к $H_C(G')$.

Заметим, что с ростом n растет и ρ, а значит, веса отрицательных ребер уменьшаются. Таким образом, по мере роста n и уменьшения весов отрицательных ребер согласованность КоС, состоящего из циклов, являющихся ОГС, приближается к согласованности КоС с положительным весом, состоящего из графов путей с нижней границей в (2.24).

2.6. Численные примеры

Теперь исследуем согласованность КоС на нескольких численных примерах.

2.6.1. Влияние выбора узла-моста

Сначала исследуем, как расположение узлов-мостов влияет на согласованность КоС, в каждой подсистеме которого есть единственный узел-мост на примере мегасети энергосистемы, состоящей из 5 сетей [2.36] рис. 2.4 а-д.

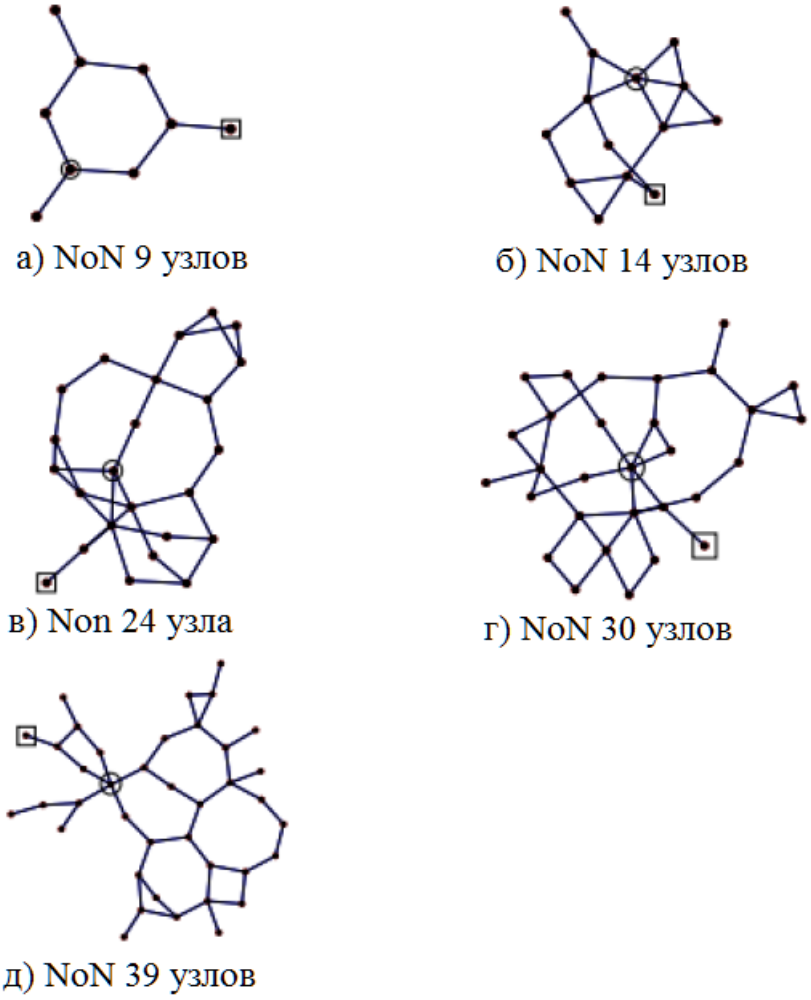


Рис. 2.4. Упрощенные графы тестовых сетей

В каждом подграфе узлы представляют шины, а ребра - линии электропередач. Каждое ребро имеет вес 1/r, где r - сопротивление,

указанное в наборе данных энергосистемы. С этого момента будем обозначать эти подграфы размером их множества вершин. Подграфы связаны с использованием топологии магистрали, указанной в каждом эксперименте. Устанавливаем все веса ребер магистрали равными 1.

В табл. 2.1 перечислены согласованности КоС. Магистральным графом является: цикл с подграфами, расположенными в порядке увеличения размера; путь с подграфами в порядке G9, G24, G39, G30, G14; и звезда с G39 в центре. Рассмотрим согласованность для КоС, построенных с использованием оптимальных узлов моста, $b_i=\text{argmin}_{v\in V_i}C_i(v)$, и наихудших узлов моста, $w_i=\text{argmax}_{v\in V_i}C_i(v)$. Эти узлы показаны в каждом подграфе на рис. 2.4 кружком и квадратом соответственноо.

Проведен расчет согласованности для всех топологий при узлах-мостах b_i (наилучших) и w_i (наихудших). Отсюда делаем вывод о большой дисперсии согласованности для КоС с наилучшими и наиходшими узлами-мостами.

Таблица 2.1

Структура бэкбона	$H_C(G)$ при $l_i=b_i$	$H_C(G)$ при $l_i=w_i$
Кольцо	6186600	36629000
Маршрут	6186600	36633000
Звезда	6186600	36629000

2.6.2. Согласованность «отрицательных» мегасетей

Используем данные предыдущего раздела, где G_i, i=1,...,p и B - все циклы, составленные для одного ребра с весом $-1/\rho$ и всех остальных ребер с весом 1. Также рассматриваем КоС, где G_i, i=1,...,p, и B - все графы путей с весами ребер, равными 1. Оба КоС состоят из пяти подграфов, каждый из которых имеет пять узлов. В обеих сетях выбираем

оптимальные узлы-мосты, которые являются, с одной стороны, узлами напротив отрицательного ребра и в центре маршрута – с другой. Сравнение согласованности между КоС с циклами с отрицательными весами и КоС с маршрутами представлено на рис. 2.5. По оси 0x дана развертка по ρ. В асимптотике по ρ КоС с циклами с отрицательными весами и КоС с маршрутами становятся близки.

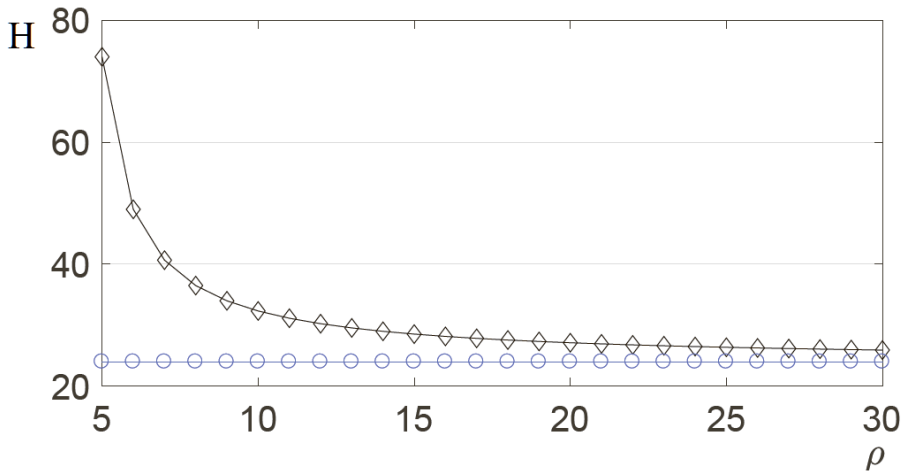


Рис. 2.5. Сравнение согласованности между КоС с циклами с отрицательными весами и КоС с маршрутами: —◇— - КоС с циклами с отрицательными весами; —○— - КоС с маршрутами

2.7. Выводы

Исследована проблема оптимизации согласованности КоС с произвольным знаком весов ребер. Решается задача выбора узлов-мостов между всеми подсистемами КоС, узлы-мосты связываются межсистемными ребрами. Проблема оптимизации решается с учетом связи согласованности и ЭФС. Минимизация осуществляется путем поиска узлов-мостов в графах ограниченного сопротивления. Эти узлы-мосты могут быть идентифицированы независимо от других подграфов и соединительной топологии. Доказано, что для КоС с древовидным опорным графом решение является оптимальным даже в рамках более общей модели, допускающей наличие нескольких узлов моста на подграф.

Представлены границы согласованности, а также аналитические примеры оптимальных КоС. Представлены численные результаты, иллюстрирующие производительность различных топологий КоС.

3. КЛАСТЕРНЫЕ АЛГОРИТМЫ СЕТЕВОЙ ОПТИМИЗАЦИИ ДАННЫХ В СПЕЦИАЛЬНЫХ МЕГАСЕТЯХ НА ОСНОВЕ НЕЧЕТКОЙ ЛОГИКИ

3.1. Оптимизация сети и кластеризация

Оптимизация мегасети играет ключевую роль в планировании и проектировании сети. Хорошо спроектированная сеть может улучшить эффективность сети, и таким образом лучше выполнять агрегацию. Таким образом, оптимизация сети стала главной целью. Кластеризация, показанная рис. 3.1, представляет собой методику, направленную на группирование узлов датчиков в несколько небольших групп - узлы кластера.

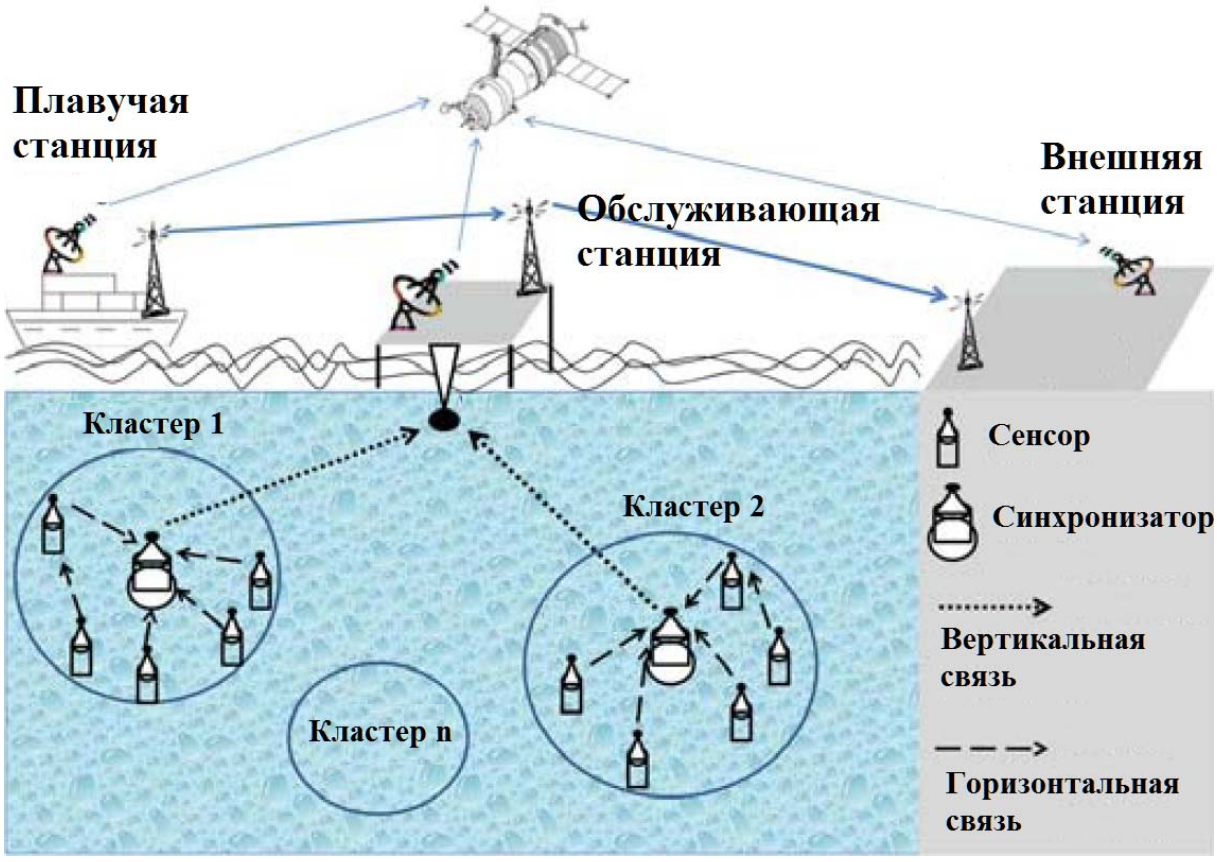


Рис. 3.1. Кластеризация и маршрутизация в сенсорных сетях

Эти узлы кластера отвечают за агломерацию данных от отдельных

узлов и их маршрутизацию в желаемое место. Ранее был изучен ряд проблем кластеризации, которые обычно возникают в системе, где необходимо агрегирование и разделение данных.

Большинство существующих алгоритмов основано на геометрических подходах. Точно так же алгоритм k-means, предложенный в [3.26], является наиболее популярным методом кластеризации среди геометрических подходов. В [3.34, 3.35] предложен алгоритм, основанный на модели, который плохо работает в некоторых областях науки. В частности, кластеризация высокоразмерных данных является сложной проблемой и, следовательно, показывает неутешительное поведение с высокоразмерными данными.

Для набора данных A цель кластерного анализа состоит в том, чтобы разделить A на различные кластеры и данные, воспринимаемые узлами датчиков в одном кластере, должны быть как можно более похожими, хотя они могут быть разными, насколько это возможно, в разных узлах. Для кластеризации на основе нечеткой логики, технология агрегации данных может использоваться для разделения данных для повышения производительности кластеризации. Этот процесс кластеризации разработан с разными начальными параметрами. Алгоритм SCOLPE, предложенный в [3.29], который является алгоритмом кластеризации, может использоваться для накопления данных категориальных атрибутов. Поэтому некоторые схемы кластеризации могут использоваться только для категориальных атрибутов кластера. Многие алгоритмы кластеризации были разработаны и использованы во многих приложениях. Кластеризация по-прежнему считается сложным процессом. Причиной разочарования исследователей является то, что каждый алгоритм хорошо работает в какой-то области, и ни один из них не может решить все проблемы кластеризации. Следовательно, сила одного кластера может компенсировать слабости другого. Следовательно, разумно объединять их

в группы кластеров для эффективной совместной работы. Эта комбинация различных схем кластеризации известна как кластерные группы. Это привело к наиболее точной и эффективной схеме кластеризации в исследованиях, предложенных в [3.7, 3.8].

Для многомерной сети существует огромное количество данных, которые необходимо кластеризовать. Чтобы разумно кластеризовать сенсорные узлы в многомерной сети, необходимо учитывать несколько параметров. Эти параметры включают географическое местоположение, мобильность узлов, поток, среду и т.д. Однако некоторые сетевые атрибуты не могут быть непосредственно измерены эффективно. С учетом многомерных данных классические схемы кластеризации могут работать неэффективно. Следовательно, желательно разработать алгоритм кластеризации, учитывающий как огромный объем сетевых данных, так и динамичность сетевого местоположения. Кроме того, необходимо понимать характеристики сети и анализировать схемы кластеризации до оптимизации сети в многомерной сети. При правильном методе кластеризации большую сенсорную сеть можно разбить на кластеры, где узлы датчиков максимально похожи, такие как ближайшее географическое местоположение, хранилище данных и т.д. Однако, когда количество узлов в сети увеличивается, оптимизация сети становится более сложной проблемой, и поэтому оптимизацию сети следует проводить путем кластеризации и агрегирования данных. Это не только повышает эффективность сети, но также снижает нагрузку на сеть.

Изложение организовано следующим образом: в разделе 3.2 кратко обсуждается введение в сенсорную сеть и кластеризацию. Методология с многочисленными определениями описана в разделе 3.3. Для оценки эффективности предложенной схемы в разделе 3.4 оценивается точность кластеризации. Выводы приводятся в разделе 3.5.

3.2. Алгоритм кластеризации

Три фазы процесса следуют в этой методологии:

- структура создается на основе иерархического анализа;
- определение лингвистических переменных для оценки кластеров сенсорных узлов, которые затем преобразуются в трапециевидные нечеткие числа, чтобы сделать их простыми и точными для дальнейшего рассмотрения;
- разработана методика кластеризации сенсорного узла для многомерной сети на основе нечеткой теории.

3.2.1. Структура, базирующая на иерархическом анализе

В этом разделе мы стремимся создать структуру, основанную на иерархическом анализе узлов датчиков и данных для кластеризации многомерной сети. Характеристики сенсорной сети определяются с помощью параметров, показанных на рис. 3.2. Они считаются наиболее важными параметрами на основе предыдущих исследований. Кроме того, эта структура может вводить и учитывать больше параметров. На основании этих параметров в этой структуре, оценка каждого кластера может быть обработана. При этом совокупная точность может быть оценена путем оценки точности для отдельного параметра. Все эти параметры в отношении эффективности алгоритма могут быть дополнительно проработаны следующим образом:

1. Количество кластеров: для оптимизации сети в различных исследованиях были приняты методики, которые привели к появлению ряда кластеров. Количество кластеров пропорционально количеству главных кластеров. Увеличение главных кластеров приводит к уменьшению внутрикластерного расстояния между главными кластерами и узлами присоединения, что приводит к энергоэффективной связи, как представлено в [3.28]. Следовательно, количество кластеров является

критическим параметром для повышения эффективности сети.

2. Внутрикластерное взаимодействие: после исполнения алгоритма, такого как алгоритм внутрикластерной маршрутизации, предложенного в [3.1], можно заметить, что для поддержания эффективности сенсорной сети для внутрикластерной связи используется маршрутизация один к одному или множественная маршрутизация, когда близкие региональные узлы могут выполнять маршрутизацию один к одному, а другие узлы могут выполнять несколько операций многоузловой маршрутизации. Это пропорционально количеству узлов и количеству главных кластеров.

3. Мобильность узлов: в подводных сенсорных сетях мобильность узлов является одним из наиболее важных параметров. Не целесообразно предполагать стационарный узел датчика и главного кластера под водой. Можно легко наблюдать негативное влияние на беспроводную связь. Это может привести к эксплуатации текущего кластера и попаданию в другой. Следовательно, беспроводная сеть, включающая в себя узел датчика и ассоциацию кластера, должна поддерживаться динамически.

4. Типы узлов: для повышения эффективности сети начальная энергия играет важную роль. Поэтому выбираются разные типы узлов с разной начальной энергией, уровнем энергопотребления и т.д. Узлы с высокой начальной энергией часто выбираются в качестве главного кластера, чем обычные узлы с такой же возможностью.

5. Выбор главного кластера. С точки зрения минимизации энергопотребления сети сложно выбрать главный кластер, так как энергия расходуется с большей скоростью для главных кластеров. Такой выбор делается на основе соответствующих критериев, таких как связность, расстояние до приемника, как показано на рис. 3.2, и стоимость связи, максимальное количество соседей, мобильность и многое другое. Такие узлы выбираются на основе детерминированных или вероятностных, или критериев, перечисленных выше.

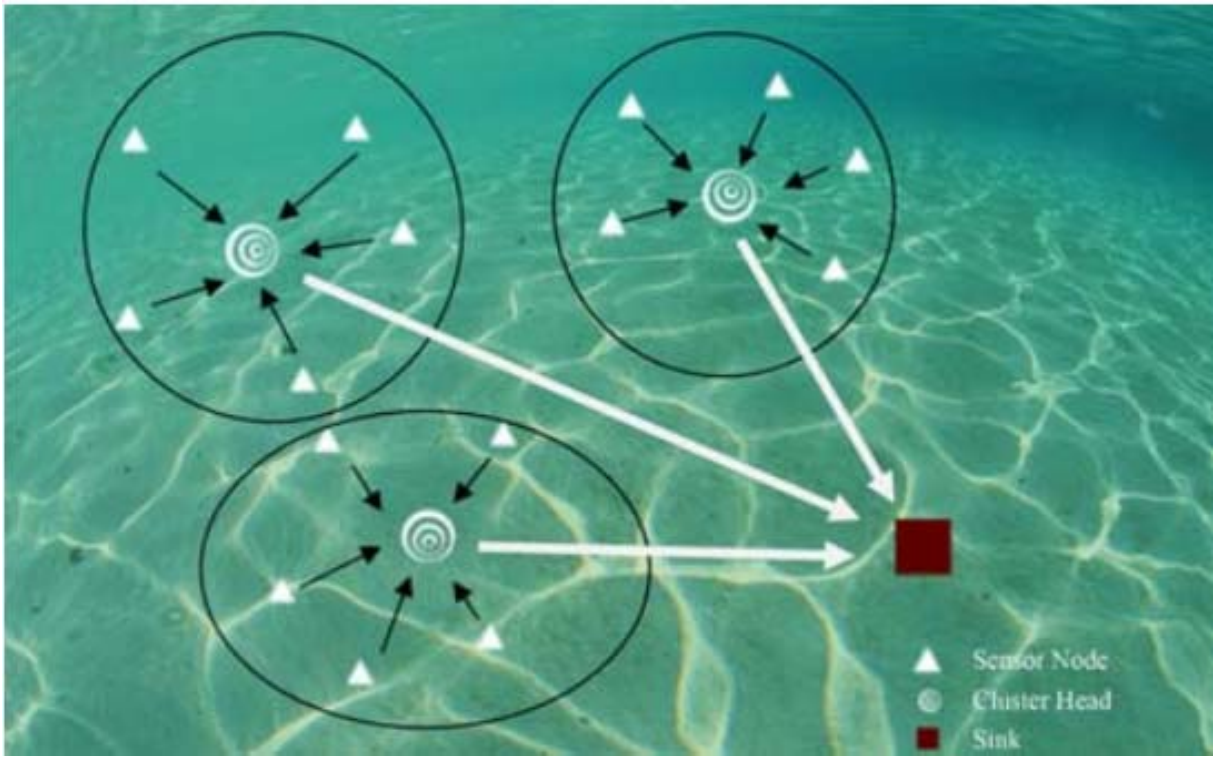


Рис. 3.2. Процесс кластеризации

6. Наложение: за этим следует разделение сенсорной сети на многочисленные перекрывающиеся кластеры с определенной средней точностью перекрытия. Чтобы выполнить перекрытие, каждый узел может иметь одно из состояний, среди главных кластеров, перекрывающихся узлов или нормального узла, где перекрывающиеся узлы связаны с несколькими перекрывающимися кластерами. Являясь важным параметром, он отвечает за надежность сенсорной сети.

Кроме того, вышеуказанные параметры помогают в создании структуры, которая может быть доработана следующим образом:

1. Окружение беспроводной сенсорной сети: когда речь идет о подводной сенсорной сети, среда является очень важным критерием для кластеризации. Распад узла, подвижность узла, разрядка аккумулятора и т.д. - это параметры, которые характерны специальной среде.

2. Совместимость с сетью: этот атрибут используется для измерения

сходства данных, воспринимаемых данным количеством датчиков. Чем выше совместимость, тем больше будет эффективность сети для агломерации данных.

3. Географическое расположение: если узлы находятся рядом друг с другом, то они могут обслуживаться одним и тем же главным кластером. Если приемник близок к определенному набору узлов, он может группироваться, образуя единый кластер, и обмениваться данными. Есть еще много исследований, касающихся кластеризации на основе географического положения.

4. Межкластерная маршрутизация: маршрутизация является наиболее заметным аспектом, лежащим в основе кластеризации. На основе разных параметров для кластеризации реализованы разные методы маршрутизации, так что узлы датчиков могут быть сгруппированы вместе, если они следуют по одному и тому же пути к приемнику. Маршрутизация данных от узлов датчиков к главному кластеру является межкластерной маршрутизацией.

5. Внутрикластерная маршрутизация: как обсуждалось выше, маршрутизация обнаруженных данных от центра кластера до приемника или между различными кластерами называется внутрикластерной маршрутизацией.

6. Методы управления: в процессе кластеризации могут быть разные способы управления, такие как централизованный, распределенный и гибридный в зависимости от критериев.

7. Характер выполнения: в процессе кластеризации итеративный, переменный и вероятностный характер, в котором процесс может быть выполнен.

8. Время конвергенции: время конвергенции - это показатель того, как быстро группа маршрутизации достигает состояния, имеющего одинаковую топологическую информацию о межсетевом взаимодействии.

Кроме того, его можно разделить на переменные, постоянные и детерминированные.

9. Гарантия подключения: гарантия подключения - это мера вероятности подключения, работающего в исправном состоянии.

10. Балансировка нагрузки: он масштабирует производительность, распределяя данные, передаваемые по нескольким головкам кластера, чтобы предотвратить разрядку батареи отдельного канала узла.

11. Качество обслуживания: качество обслуживания является количественной мерой нескольких аспектов сети, таких как пропускная способность, пропускная способность и доступность.

12. Отказоустойчивость: Отказоустойчивость - это свойство, которое позволяет системе продолжать работу в случае сбоя одного или нескольких элементов в сети.

3.2.2. Определение лингвистических переменных и их преобразование

Используем нечеткую теорию, чтобы преобразовать классификацию в числовые значения и затем преобразовать выходные данные в трапециевидные нечеткие числа. Трапециевидное нечеткое число предполагается для применений в нелинейных и нечетких линейных подходах типа (a, b, c, d) без каких-либо неотрицательных значений и представляется как Θ=(a, b, c, d) [3.36]. Функция ассоциации может быть рассчитана с использованием трапециевидного нечеткого числа $\tilde{A}$, где:

$$f(x)=\begin{cases}0, x \le a \\ \dfrac{x-a}{b-a}, a \le x < b \\ 1, b \le x < c \\ \dfrac{d-x}{d-c}, c \le x < d \\ 0, x \ge d\end{cases} \qquad (3.1)$$

где a, b, c, d - действительные числа. Из-за произвольной природы проблемы кластеризации в крупномасштабной многомерной сенсорной сети параметры используются в качестве лингвистических переменных. Здесь табл. 3.1 относится к оценке критериев кластеризации, указанных выше, на основе исследования [3.31].

Таблица 3.1

Лингвистические термины	Трапециевидное число
Идеально	1, 0.98, 0.95, 0.92
Очень хорошо	0.95, 0.92, 0.86, 0.82
Хорошо	0.85, 0.75, 0.67, 0.63
Удовлетворительно	0.63, 0.60, 0.58, 0.55
Плохо	0.58, 0.55, 0.52, 0.49
Очень плохо	0.52, 0.45, 0.42, 0.39
Крайне плохо	0.40, 0.37, 0.33, 0.30

Основные перечисления необходимы для управления алгоритмом кластеризации. Для выполнения алгоритма кластеризации необходимы следующие математические определения.

3.2.3. Разработка алгоритма кластеризации

Каждый критерий, указанный выше, должен быть оценен. Каждая оценка подуровня соответственно синхронизируется с критериями более высокого уровня. Затем проводится алгоритм кластеризации для кластеризации сенсорных узлов в несколько кластеров. Наконец, точность кластеризации разрабатывается и оценивается для определения количества обслуживаемых кластеров и количества сенсорных узлов, сопоставленных с каждым кластером.

$N=\{N_i|i=1,2,\ldots n\}$ - это сенсорный узел i, a n - общее количество сенсорных узлов, при этом

$\mu^1_{t,i}\ (t=1,2,...n; i=1,2,...,n)$ - степень ассоциации, дезагрегированная от подкритериев к основным критериям:

$$K^2_{u,t}\ (u=1,2,...n; t=1,2,...,n) \tag{3.2}$$

$K^2_{u,t}$ является оцененным значением для подкритерия t основного критерия m с точностью принятия решения u.

$A=\{A_u \mid u=1,2,...n\}$ представляет каждый начальный кластер, а n - общее количество кластеров:

$B=\{F_w \mid w=1,2,...n\}$ представляет каждый конечный кластер, а n - общее количество кластеров.

Пусть F - набор нечетких понятий, а x - выборка, принадлежащая нечеткому множеству F, где $\overline{F}(x)$ - набор степеней ассоциации, где

Пусть F - нечеткое понятие, а x - выборка, принадлежащая нечеткому множеству F, где $\overline{F}(x)$- множество степеней ассоциации, при этом

$$\overline{F}(x) < \rho_m(x)$$

где ρ_m - степень ассоциации нечеткого понятия m. Следовательно, математически уравнения можно выразить как:

$$\overline{F}(x)=\{w \mid w \in \rho_m(x)\} \tag{3.3}$$

$$L_m(x)=\frac{-\sum_{x\in x(m)} \rho_m(x)}{-\sum_{x\in X} \rho_m(x)} \tag{3.4}$$

где L - мера выборки, относящейся к простому нечеткому понятию:

$$\mu_f(x)=\inf_{m\in F}(L_m(x)) \in [0,1] \tag{3.5}$$

где $\mu_f(x)$ - степень ассоциации нечеткого понятия F.

Пусть S - выборочный набор узлов датчиков в сети, M - простой набор нечетких понятий S, где $F \subseteq M$. Используя эти термины, мы можем определить функцию энтропии ассоциации E (F) и функцию коэффициента ассоциации C (F) как:

$$E(F)=\Sigma_{x\in S}(\mu(x)\cdot\ln(\mu(x)))\ , \tag{3.6}$$

$$D(B)=\sum_{x\in S}\left(\frac{\mu(x)}{n}\ln\left(\frac{\mu(x)}{n}\right)\right) \tag{3.7}$$

Теперь индекс оценки может быть оценен как V=E(B)/D(B) и определен для оценки функции энтропии ассоциации и функции коэффициента ассоциации. Меньший индекс оценки, более разумный для нечеткой концепции F, чтобы описать выборку S. Мы разделили кластеризацию сети на три фазы. На первом этапе подкритерий оценки, определенный в уравнении (3.2), будет сопоставлен с основным критерием. Затем выполняется алгоритм кластеризации, чтобы сгруппировать узлы датчика в разные кластеры на втором этапе. Наконец, на третьем этапе точность кластеризации проектируется и оценивается для определения соответствующего количества кластеров и узлов датчиков, связанных с кластером.

Согласно уравнению (3.2), мы имеем $K_{u,t}^2$ в качестве значения оценки для подкритерия. Точно так же мы имеем $O_{u,t}^2$ как оценочное значение для основного критерия. Эти оценочные значения могут быть далее выражены в форме трапециевидного нечеткого числа как:

$$K_{u,t}^2=\left(a_{u,t}^2 b_{u,t}^2 c_{u,t}^2 d_{u,t}^2\right)$$

и

$$U_{u,t}^2=\left(\theta_{u,t}^2 h_{u,t}^2 j_{u,t}^2 k_{u,t}^2\right)$$

соответственно.

Пусть ‘+’ и ‘×’ - сложение и умножение вектора соответственно. Затем индекс оценки может быть далее выражен как:

$$Y_{t,i}^2=\frac{1}{m\times t}\times\sum_{u=1}^{m}\left(K_{u,t}^2\times O_{u,t}^2\right) \tag{3.8}$$

Используя уравнение

$$P(Y) = \frac{1}{6}(a + 2b + 2c + d),$$

как обсуждалось в [3.5, 3.23], мы можем определить степень ассоциации узла датчика как:

$$\mu_{t,i}^{1} = \frac{1}{6}\left(A_{u,t}^{2}, B_{u,t}^{2}, C_{u,t}^{2}, D_{u,t}^{2}\right) \tag{3.9}$$

Пусть f_k будет атрибутом образца x_i для степени ассоциации μ. Атрибут f_k может быть преобразован в атрибуты, такие как $m_{t,1}$, $m_{t,2}$, $m_{t,3}$, $m_{t,4}$, а степень их ассоциации может быть выражена как $\rho_{m_{t,1}}$, $\rho_{m_{t,2}}$, $\rho_{m_{t,3}}$, $\rho_{m_{t,4}}$ соответственно.

Мы выполняем следующие шаги для расчета нечетких атрибутов для каждого образца:

Во-первых, $L_m(x)$ вычисляется для всех атрибутов с использованием уравнения (3.4). Теперь нечеткая концепция нечетких значений ассоциации может быть выражена как

$$\Psi_{t,s}(x_i) = \max\{\mu_m(x_i)\} \tag{3.10}$$

Используя вышеприведенную нечеткую концепцию, мы можем оценить индекс оценки, вычислив соотношение энтропийной функции ассоциации и функции коэффициента ассоциации по отношению к результату в уравнении (3.10) как:

$$V_{\Psi_{t,s}(x_i)} = \frac{E(\Psi_{t,s}(x_i))}{D(\Psi_{t,s}(x_i))} \tag{3.11}$$

$\Psi_{t,s}(x_i)$ будет нечетким множеством, оно может быть записано как:

$$\Psi_{t,s}(x_i) = \left\{\Psi_{t,s}(x_i) \mid p = 1,2,...m; i = 1,2,...n\right\} \tag{3.12}$$

где m - общее количество основных критериев, а n - общее количество узлов датчиков.

Функция ассоциации энтропии и функция энтропии могут быть вычислены с использованием определения в уравнениях (3.6) и (3.7)

соответственно.

Вернувшись к уравнению (3.12), продолжаем рекурсию до $V_{\Psi_{t,s}(x_i)} \geq V_{\Psi_{t,s}(x_{i-1})}$. Обозначим:

$\mu_{\alpha(x_i)} = \inf_{1-\min}(L_m(x_i)) \in [0,1]$, соответствует наименьшему значению;

$\mu_{\beta(x_i)} = \inf_{2-\min}(L_m(x_i)) \in [0,1]$, соответствует второму наименьшему значению.

Используя подкритерии и степень ассоциации из уравнения (3.3), степени ассоциации можно записать как:

$$\rho_{m(X_i)} = \mu_{X_i},$$

где

$$X_{t,i} = \max\left\{\mu_{x_i}, \mu_{x_{i+1}}, \mu_{x_{i+2},\ldots}\right\}.$$

Выберем два атрибута δ и γ. Вычислим наименьшее значение степени ассоциации $\mu_{\delta(x_i)}$ по отношению к атрибуту δ. Теперь вычисляем другое наименьшее значение степени ассоциации $\mu_{\gamma(x_i)}$ по отношению к атрибуту γ. Используя уравнение (3.5), мы можем определить степень ассоциации для обоих атрибутов.

Аналогично, индекс оценки может быть рассчитан как отношение энтропийной функции ассоциации и функции коэффициента ассоциации для обоих атрибутов δ и γ по отдельности. Если индекс оценки по отношению к δ больше или равен γ, то исключаем δ и продолжаем вычислять индекс оценки для оставшихся атрибутов. Продолжаем цикл до тех пор, пока индекс оценки по δ не станет меньше, чем по γ. При этом у нас есть нечеткие атрибуты каждого образца.

Решение основано на нечеткой матрице отношений выборки, представленной в [3.24, 3.25]. Мы можем определить различную степень ассоциации на основе диагональных значений и других значений в матрице. Выразим диагональные значения как a=1, 2, 3, …, n, а затем

диагональные значения соответственно увеличим. Соответствующие выборки могут быть сгруппированы в один или несколько кластеров, а остальные выборки могут быть сгруппированы в другие кластеры. Этот цикл продолжается до тех пор, пока диагональные значения a=n. Каждая итерация будет генерировать начальный кластер $C_1^{'}$, $C_2^{'}$, $C_3^{'}$, ..., $C_n^{'}$.

Для того чтобы получить конечные кластеры с диагональными значениями и необходимой точностью кластеризации, мы соответственно разделяем выборки на разные кластеры. Для каждого начального кластера $C_1^{'}$, $C_2^{'}$, $C_3^{'}$, ..., $C_n^{'}$ мы можем вычислить взвешенную степень ассоциации, используя уравнение (3.5), и мы можем получить окончательные кластеры, в результате перечисленных выше $C_1^{'}$, $C_2^{'}$, $C_3^{'}$, ..., $C_n^{'}$. Наконец, на третьем этапе мы рассчитаем точность кластеризации, которая будет использоваться для оценки результата кластеризации и его эффективности. Точность кластеризации может быть выражена как:

$$CP_{\delta} = \frac{cX(c-1)\sum_{p}^{c}\sum_{x\in P_p}\sum_{m\subseteq P_p}\left(\rho_m(x) - o_p^m\right)^2}{2xp_e x\sum_{p}^{c}\sum_{x\in P_p}\sum_{m\subseteq P_p}\left(o_p^m - o_k^m\right)^2} \tag{3.13}$$

где числитель - это степень дисперсии между кластерами путем оценки и сравнения атрибутов, а знаменатель - это близость атрибутов из разных узлов в каждом кластере. Следовательно, точность кластеризации высока, когда близость внутри каждого кластера становится больше, а дисперсия между кластерами становится меньше.

3.3. Оценка эффективности

Рассмотрим сенсорную сеть из 20 узлов. Используя диагональные значения в матрице, мы можем генерировать начальный кластер с каждой итерацией. Для каждого различного значения диагонального элемента мы

можем оценить CP_δ, используя уравнение (3.12). При этом мы можем получить начальные результаты кластеризации, как показано на рис. 3.2, и точность кластеризации:

1. Когда a=0,6881, CP_δ=5,41, существует два кластера:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N1, N2, N3, N4, N5, N6, N7, N8, N9, N10, N11, N12, N16, N17, N18, N19, N20}.

2. Когда a=0,7380, CP_δ=4,61, существует три кластера:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N16, N17, N18, N19},

$C_3^{'}$ = {N1, N2, N3, N4, N5, N6, N7, N8, N9, N10, N11, N12, N20}.

3. Когда a=0,7848, CP_δ=2,88, существует четыре кластера:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N16, N17, N18, N19},

$C_3^{'}$ = { N4, N5, N6, N7},

$C_4^{'}$ = {N1, N2, N3, N8, N9, N10, N11, N12, N20}.

4. Когда a=0,8334, CP_δ=3,33, существует пять кластеров:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N16, N17, N18, N19},

$C_3^{'}$ = { N4, N5, N6, N7},

$C_4^{'}$ = { N8, N9, N10},

$C_5^{'}$ = {N1, N2, N3, N12, N20}.

5. Когда a=0,8574, CP_δ=3,80, существует шесть кластеров:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N16, N17, N18, N19},

$C_3^{'}$ = { N4, N5, N6, N7},

$C_4^{'}$ = { N8, N9, N10},

$C_5^{'}$ = {N1, N2, N3},

$C_6^{'}$ = { N12, N20}.

Для каждого полученного начального кластера мы можем вычислить взвешенную степень ассоциации, используя уравнение (3.5). В результате мы можем получить окончательные кластеры и точность кластеризации следующим образом.

1. Когда a=0,6881, CP_δ=5,02, существует два кластера:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N1, N2, N3, N4, N5, N6, N7, N8, N9, N10, N11, N12, N16, N17, N18, N19, N20}.

2. Когда a=0,7380, CP_δ=4,01, существует три кластера:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N16, N17, N18, N19, N20},

$C_3^{'}$ = {N1, N2, N3, N4, N5, N6, N7, N8, N9, N10, N11, N12 }.

3. Когда a=0,7848, CP_δ=2,41, существует четыре кластера:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N16, N17, N18, N19, N20},

$C_3^{'}$ = { N4, N5, N6, N7, N8},

$C_4^{'}$ = {N1, N2, N3, N9, N10, N11, N12}.

4. Когда a=0,8334, CP_δ=2,49, существует пять кластеров:

$C_1^{'}$ = {N13, N14, N15},

$C_2^{'}$ = {N16, N17, N18, N19, N20},

$C_3^{'}$ = { N4, N5, N6, N7, N8},

$C_4^{'} = \{ N9, N10\},$

$C_4^{'} = \{N1, N2, N3, N11, N12\}.$

Чем меньше значение точности кластеризации, тем выше будет эффективность кластеризации сети. Поэтому, анализируя окончательные результаты кластеризации, мы можем найти CP_{δ}=2,41 - наименьшее значение, указывающее, что эффективность кластеризации сети наиболее благоприятна, когда a=0,7848.

3.4. Сравнительное исследование

Современные алгоритмы кластеризации сравниваются с предложенным алгоритмом, чтобы доказать эффективность предложенного алгоритма кластеризации. В [3.3] представлена PANEL, как показано на рис. 3.3, для кластеризации беспроводной сенсорной сети. В [3.14] представлена еще одна схема кластеризации CCS, показанная на рис. 3.4, с учетом расположения приемника для повышения его производительности и увеличения срока службы сети.

Рис. 3.3. Вид панели кластеризации сети

В [3.20] точность кластеризации оценена как 3,25 и получено шесть кластеров с этим подходом. Позже в [3.21] точность кластеризации оценена как 3.79 и получено четыре кластера с предложенным подходом.

Все результаты кластеризации показаны на рис. 3.3 и 3.4. Точно так же в [3.19] точность кластеризации оценена как 2,83, получено семь кластеров. Точность кластеризации для начального и окончательного формирования кластера графически показана на рис. 3.5 и 3.6 соответственно.

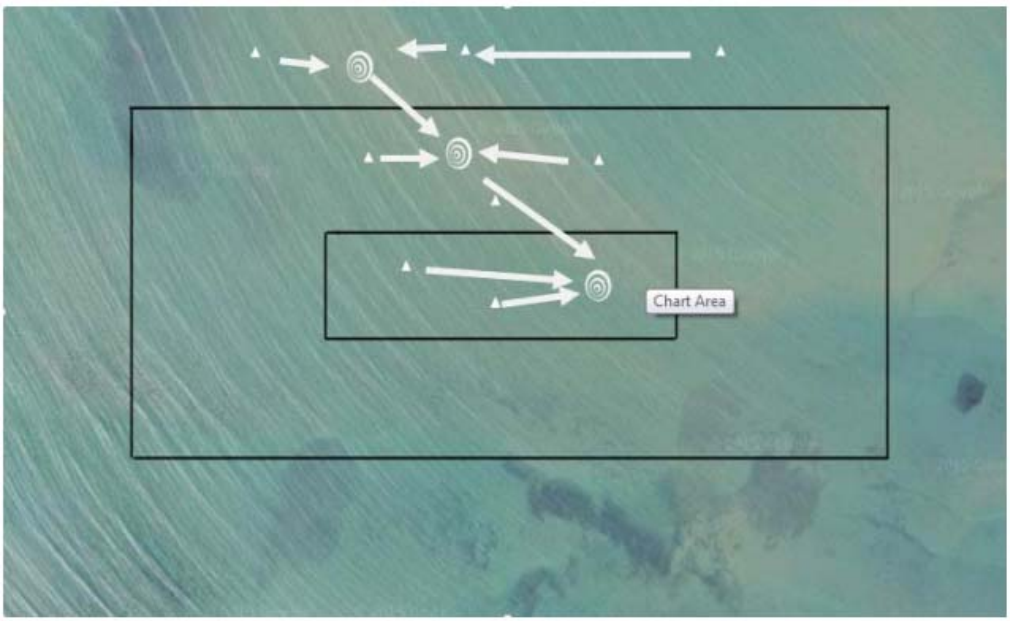


Рис. 3.4. Схема концентрической кластеризации по Юнгу

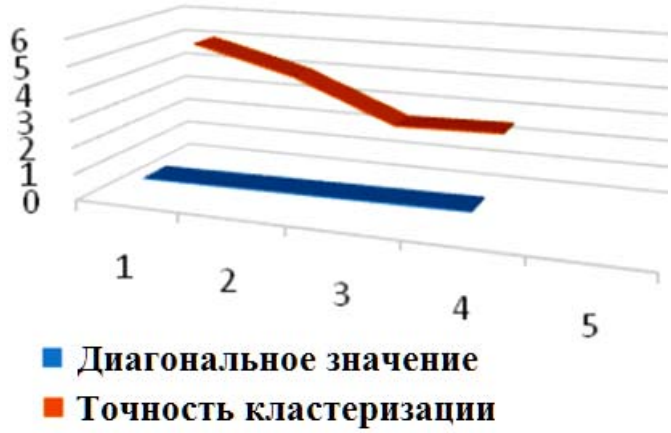


Рис. 3.5. Точность кластеризации для начального формирования кластера

Точность кластеризации учитывает как межкластерное расстояние, так и внутрикластерное расстояние и, таким образом, может использоваться для оценки эффективности результатов кластеризации. Чем меньше точность кластеризации, тем эффективнее алгоритм

кластеризации. С самой низкой точностью кластеризации представленная процедура кластеризации приводит к более эффективным результатам по сравнению с другими пятью алгоритмами.

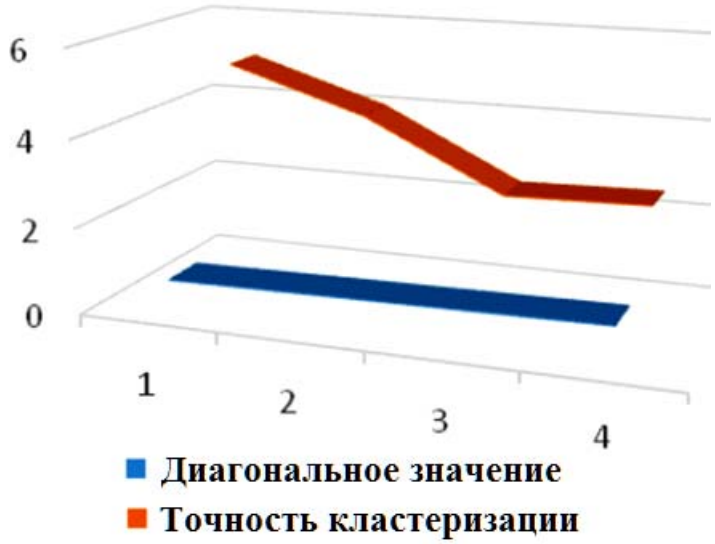


Рис. 3.6. Точность кластеризации для окончательного формирования кластера

Другая причина получения эффективных результатов заключается в том, что он может динамически работать с любым количеством атрибутов и способен разбивать многочисленные характеристики на обобщенные критерии. Следовательно, этот подход способен разделять разницу между узлами датчика. Кроме того, его свойство различать различные типы узлов помогает оценить ресурсы, необходимые для каждого типа сенсорного узла. Например, мы сравнили многочисленные алгоритмы кластеризации в табл. 3.2.

Подводя итог, представленный подход дает более благоприятные и эффективные результаты, принимая во внимание многочисленные атрибуты, связанные с каждым узлом датчика. Соответствующий подход к кластеризации может улучшить однородность межкластерного и внутрикластерного расстояния, снизить сложность сети, эффективное агрегирование данных и сократить время сходимости.

Таблица 3.2

Алгоритмы кластеризации	**Количество кластеров**	**Точность кластеризации**	**Время алгоритма**	**Возможность обработки больших данных**	**Процент точности**
UCLF [21]	6	3.25	$O(N^2)$	Да	Около 99%
Иерархическая агломеративная кластеризация, основанная на доверии [20]	4	3.73	O(N)	Да	96%
AVC [19]	7	2.83	$O(N^2)$	Да	Правильно классифицирует на 4 группы
CLARA [15]	Нет	Нет	$(O(K(40+K^2)+K(N-K))^+$	Нет	Точный, но в 15 раз медленнее, чем BIRCH
DBSCAN [6]	Нет	Нет	O(N log N)	Нет	Правильно классифицирует, нет понятия шума

Алгоритмы кластеризации	**Количество кластеров**	**Точность кластеризации**	**Время алгоритма**	**Возможность обработки больших данных**	**Процент точности**
BIRCH [40]	Нет	Нет	O(N)	Да	Правильно классифицирует экземпляры по группам
K-means подход к кластеризации	Зависит от подхода и количества точек данных	Нет	O(NKd)	Нет	69%
Нечеткий c-means	Зависит от подхода и количества точек данных	Нет	O(N)	Нет	98%
Иерархическая кластеризация	Зависит от подхода и количества точек данных	Нет	$O(N^2)$	Нет	96%
ROCK [9]	21	Нет	O(N log N)	Да	Почти 99%
Предложенный	4	2.41 (наи-	O(N)	Да	Правильно классифици-

Алгоритмы кластеризации	**Количество кластеров**	**Точность кластеризации**	**Время алгоритма**	**Возможность обработки больших данных**	**Процент точности**
подход		меньшая точность кластеризации)			рует на 4 группы

3.5. Выводы

Кластеризация беспроводных сенсорных сетей для сложной многомерной сети имеет решающее значение. Это важно для оптимизации многомерной сети, так как учитывает многочисленные атрибуты, дающие благоприятные результаты. Многие исследования дали несколько хорошо установленных результатов. Существует ряд факторов, влияющих на процесс кластеризации. В работе представлен подход к узлам кластерных датчиков с аналогичными характеристиками в рамках иерархической структуры. Структура может определять и классифицировать атрибуты каждого сенсорного узла по основным и второстепенным критериям. Лингвистические переменные используются для представления каждого критерия. Каждая лингвистическая переменная затем преобразуется в трапециевидное нечеткое число для повышения точности и стандартизации. При этом предложен кластерный подход для перечисления воздействия всех критериев. Точность кластеризации

определяется для измерения межкластерного и внутрикластерного расстояния. Другие исследования в этой области анализируются графически и проводится соответствующее сравнение. По сравнению с этими исследованиями, предлагаемый подход превосходит другие подходы. Он может обрабатывать несколько атрибутов для сенсорного узла в подводной беспроводной сенсорной сети. По сравнению с предыдущими подходами, это помогает значительно отличить разницу между сенсорными узлами. Кроме того, экспериментальный результат предлагаемого подхода может быть расширен для решения проблем кластеризации в других областях. Доказана важность кластеризации в подводной беспроводной сенсорной сети. В дополнение к этому, исследования могут быть проведены, чтобы объединить больше факторов в подходе.

4. РАЗРАБОТКА РАСПРЕДЕЛЕННЫХ ПОТОКОВЫХ ПРИЛОЖЕНИЙ В МЕГАСЕТЯХ НА ОСНОВЕ UML-МОДЕЛЕЙ

4.1. Инструмент разработки распределенных потоковых приложений в мегасетях, управляемых моделями

Представляем дизайн и разработку StreamGen, управляемого моделями инструмента, который опирается на концептуальные сходства между различными потоковыми платформами для упрощения и ускорения проектирования, разработки и эксплуатации распределенных потоковых приложений, одновременно преодолевая проблему привязки запуска к конкретному исполнению движка. StreamGen предлагает два основных преимущества:

• Новый предметно-ориентированный язык моделирования (DSML) в форме профиля UML [4.18, 4.28], StreamUML, который позволяет разработчикам моделировать потоковое приложение в виде графа потока данных, используя стандартное моделирование UML и, в частности, составные структурные диаграммы.

StreamUML предоставляет основные абстракции, необходимые для моделирования распределенного потокового приложения, тем самым освобождая разработчиков от необходимости знать подробности о конкретных целевых платформах, но при этом позволяя настраивать настройки для конкретной платформы.

• Модуль генерации кода, называемый StreamCGM, который преобразует профилированные модели UML в код приложения для выбранной целевой платформы. В настоящее время StreamCGM включает в себя 2 генератора кода для двух самых популярных платформ распределенной потоковой передачи с открытым исходным кодом, а именно Apache Flink и Apache Spark.

4.2. Общие сведения об особенностях разработки потоковых приложений

4.2.1. Контекст и основные понятия

В этом разделе приведена необходимая справочная информация, чтобы понять, как разрабатываются и выполняются современные распределенные потоковые приложения.

Что касается разработки приложений, то за последнее десятилетие многие платформы для разработки распределенных потоковых приложений, требующие больших объемов данных, претерпели значительные изменения. Одной из первых и наиболее популярных платформ является Apache Storm, известная своими надежными возможностями обработки распределенных потоков, которые, однако, предоставляли относительно ограниченные возможности разработки. Например, отсутствовала поддержка оконной семантики - то есть операционной семантики, позволяющей обрабатывать пакеты данных в Windows [4.15] без потери разделов данных или свойств своевременности, - которая должна была быть явно жестко запрограммирована разработчиками приложений. Более того, уровень абстракции был очень низким, поскольку приложения были представлены в терминах их топологии операторов передачи данных, которая в основном соответствовала тому, как приложение было фактически выполнено, без промежуточных абстракций между разработкой и операциями в поддержку количества и качества кода, используемого разработчиками. С тех пор были разработаны более продвинутые фреймворки, предлагающие больше готовых функций потоковой передачи (например, управление окнами) наряду с абстракциями более высокого уровня, направленными на упрощение разработки приложений. В частности, большинство современных фреймворков для разработки распределенных потоковых приложений предлагают функциональные модели программирования,

которые значительно упрощают процесс разработки. Примерами таких фреймворков являются Apache Flink, Apache Spark,7 и Apache Kafka.

В конечном счете, модель потока данных Google [4.2] представляет собой попытку объединить различные, похожие абстракции, предоставляемые многими доступными фреймворками, во всеобъемлющую модель и соответствующие API (например, Apache Beam).

4.2.2. Границы области применения и семантики моделирования

Что касается сферы применения исследовательского решения в области развертывания и эксплуатации распределенных потоковых приложений, то, хотя различные движки основаны на разных парадигмах, таких как массовая передача или микропакеты, основная идея заключается в преобразовании приложений в топологии или прямые ациклические графы (DAG), в которых операторы являются компонентами, которые можно развертывать по отдельности, обеспечивают механизмы отказоустойчивости. Кроме того, операторы обычно распараллеливаются, что означает наличие нескольких запущенных экземпляров оператора, каждый из которых можно рассматривать как отдельный процесс, который потенциально может выполняться в своей собственной среде.

Важно подчеркнуть, что, согласно этим стратегиям развертывания, одно потоковое приложение на самом деле представляет собой совокупность независимых и индивидуально развертываемых компонентов, хотя это может быть не сразу видно из дизайна приложения. Платформа распределенной потоковой передачи отвечает за принятие решений о том, как распределять параллельные вычисления, с конкретными указаниями, в соответствии с которыми должны быть разделены входные данные, в то время как планировщик затем приступает к планированию заданий и обеспечению их выполнения.

Целью и областью применения StreamGen - и аналогичных технологий - должна быть поддержка ролей и ответственности за определение потока данных, который должен быть реализован в целевой платформе потоковой передачи. Такая поддержка должна быть сосредоточена на быстром создании прототипов, т.е. на определении вышеупомянутого потока данных без углубления в операционную семантику и/или детали конфигурации для таких технологий.

4.3. Обзор инструмента разработки потоковых приложений

StreamGen ориентирован на выполнение следующих двух требований:

• Создание независимого от платформы языка графического моделирования для потоковых приложений.

Этот язык должен отражать абстракции, которые актуальны в контексте потоковой передачи, и помогать процессу разработки, обеспечивая надлежащую проверку согласованности проектов приложений. Кроме того, он должен обеспечивать графическую нотацию для моделирования, которую легко освоить и использовать.

• Для обеспечения быстрого прототипирования распределенных потоковых приложений на разных платформах.

Более конкретно, StreamGen должен предоставить командам разработчиков средства для сокращения времени, необходимого для создания прототипов и экспериментов с потоковым приложением, за счет автоматизации генерации кода для различных целевых платформ.

На рис. 4.1 представлен обзор рабочего процесса StreamGen с точки зрения разработчика приложения. Белые и серые круги обозначают начало и конец рабочего процесса соответственно. Квадратные прямоугольники обозначают действия, а сплошные стрелки соединяют действия для создания рабочего процесса. Символы документа, обозначенные

сплошными стрелками, представляют собой артефакты, которые создаются в конце определенного действия и передаются в качестве входных данных для следующего. Пунктирные стрелки связывают действия с инструментами, используемыми разработчиком приложения для их выполнения. Инструменты представлены в виде округлых рамок (те из них, которые являются оригинальными результатами данного исследования, отмечены серым цветом).

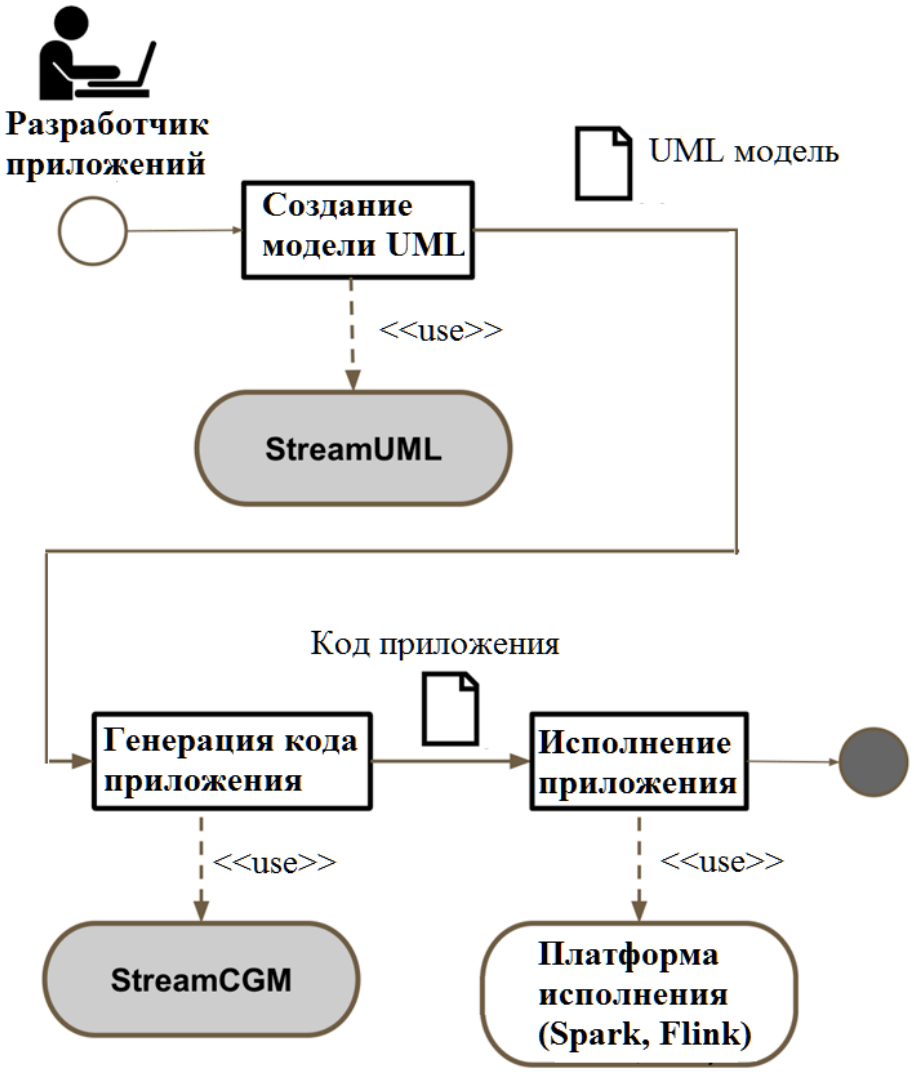


Рис. 4.1. Обзор рабочего процесса использования StreamGen

Разработчик использует UML-моделирование и, в частности, составные структурные диаграммы, обогащенные профилем StreamUML (реализованным в Eclipse Papyrus), для создания модели своего потокового

приложения независимо от платформы, как с точки зрения графика вычислений потока данных, так и с точки зрения более продвинутых аспектов, таких как распараллеливание и временная семантика. Во время этого процесса моделирования разработчик поддерживается механизмами проверки, определенными в StreamUML, с помощью ограничений OCL. Затем разработчик запускает генерацию кода с помощью StreamCGM (реализованного в Eclipse Acceleo), который принимает в качестве входных данных созданную UML-модель и создает код приложения.

Наконец, разработчик или любой другой пользователь потокового приложения может выполнить сгенерированный код, используя выбранную целевую платформу выполнения (например, Spark или Flink).

Далее будем использовать приложение StreamingWordCount в качестве рабочего примера. Это приложение непрерывно получает фрагменты текста по сокет-соединению, наблюдает за ними в течение 3 минут и вычисляет с помощью параллельной обработки количество встречений каждого отдельного слова, которое было замечено за последний промежуток времени. Такие результаты в конечном итоге сохраняются в локальном файле.

4.4. Язык моделирования

Определим StreamUML в соответствии с современным уровнем построения профилей UML [4.18, 4.28]. Сначала определим модель предметной области для потоковых приложений, а затем, основываясь на этом, разработаем профиль StreamUML.

В разделе 4.4.2 представлен обзор этого профиля, в разделе 4.4.3 представлены подробности его реализации в UML.

4.4.1. Модель предметной области для потоковых приложений

Модель предметной области основана на опыте создания прототипов приложений с использованием нескольких широко известных распределенных потоковых платформ, а именно Apache Storm, Apache Spark, Apache Flink, Apache Beam и Apache Kafka. Кроме используем проект DICE, где разработаны модели предметной области для приложений с интенсивным использованием данных (в том числе потоковых), разработанных с использованием различных платформ, среди которых Apache Storm и Apache Spark [4.21, 4.22]. Все платформы распределенной потоковой передачи предлагают общий набор функций, хотя и с использованием (иногда очень разных) API.

В общем, потоковое приложение можно представить как граф потока данных, в котором ребра представляют потоки данных, а узлы представляют операторов, которые создают, преобразуют и хранят потоки данных.

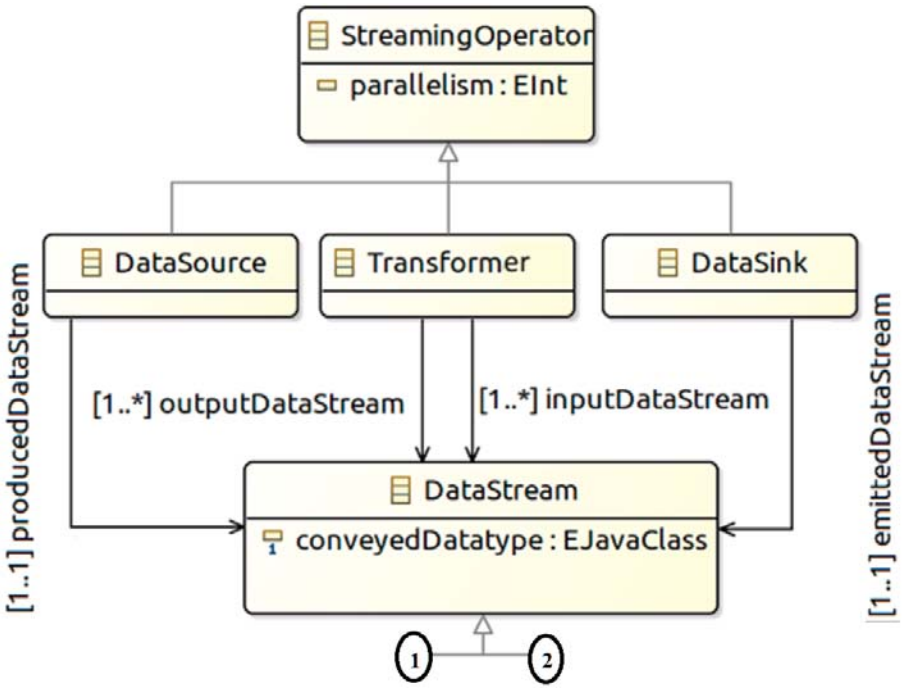
Каждый поток данных, по сути, представляет собой бесконечный набор кортежей. Кортеж потока данных обычно структурирован в виде конечного набора полей, определяющих схему потока данных. Учитывая, что операторы независимы друг от друга, они могут выполняться одновременно (возможно, на разных компьютерах), как только будут доступны их входные потоки, что обеспечивает параллелизм задач; более того, параллелизм данных также может быть использован посредством секционирования данных, то есть разделения данного потока данных на несколько секций, каждая из которых затем назначается заданной реплике оператора, подлежащей обработке. Чтобы справиться с бесконечностью потоков данных, обычно используется понятие оконного управления, позволяющее разделить поток данных на конечные, последовательно обрабатываемые фрагменты (обычно, но не всегда, вдоль временных границ).

Исходя из этого понимания, определим модель предметной области, основная часть которой показана на рис. 4.2.

Основные концепции заключаются в следующем:

DataStream: Потоки данных являются первоклассными пользователями в потоковых приложениях. Они связывают различные операторы приложения, тем самым определяя соответствующий график потока данных. Каждый поток данных содержит кортежи, которые соответствуют определенной и фиксированной схеме. Это определяется атрибутом conveyedDatatype для DataStream.

StreamingOperator: Узлы графа потока данных, представляющие потоковое приложение, являются операторами, создающими, преобразующими или сохраняющими потоки данных. Оператор может быть распараллелен, т.е. может быть создан в нескольких экземплярах, путем установки значения свойства parallelism.



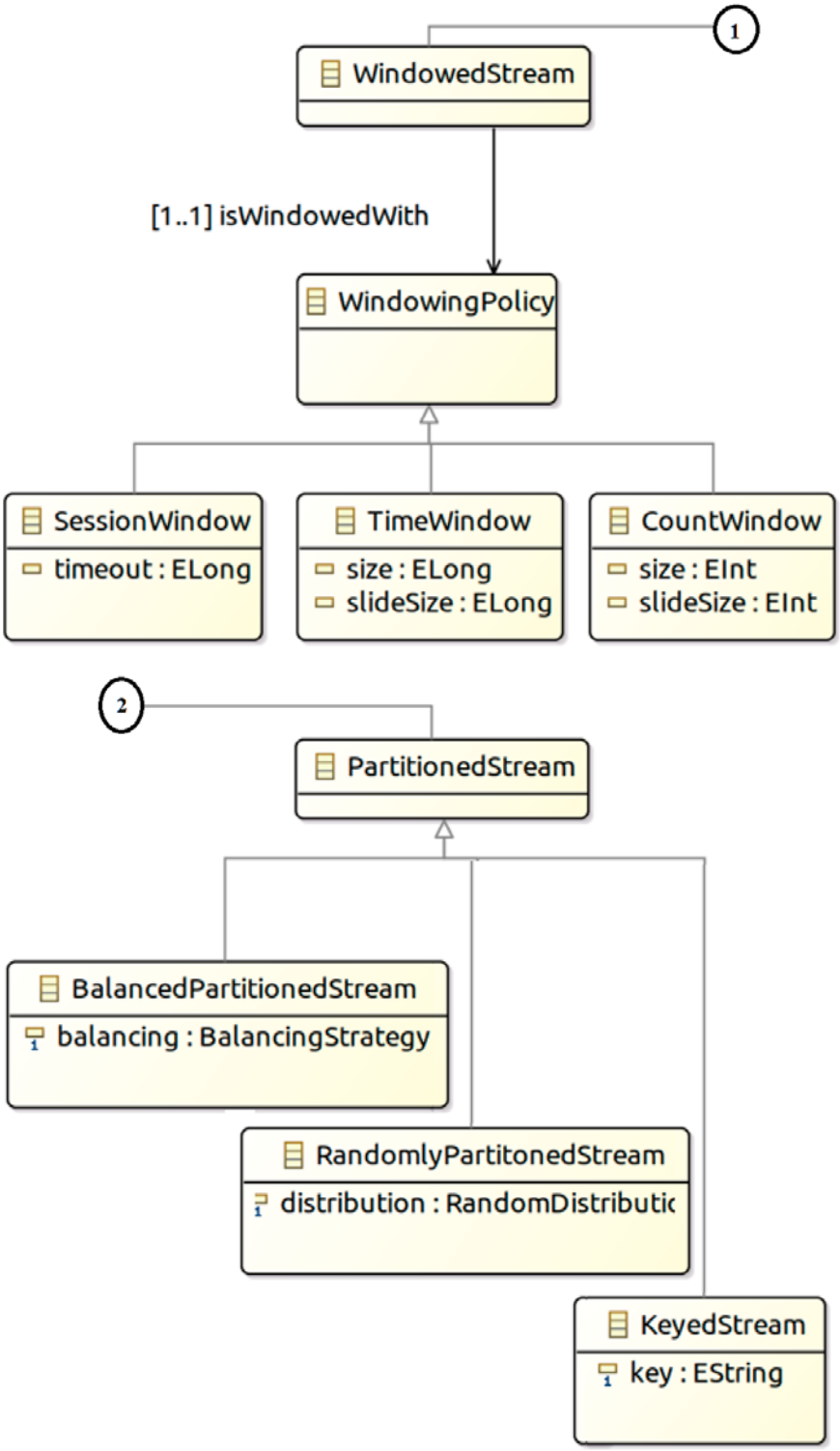


Рис. 4.2. Модель предметной области для потоковых приложений

DataSource: это особый тип оператора, который передает приложению поток входных данных. Источники данных - это единственные узлы в графе потоков данных, не имеющие входных

потоков, поскольку с топологической точки зрения они являются началом графа. Конкретные типы источников данных, которые расширяют абстрактное определение, затем используются для моделирования реальных источников данных, из которых данные фактически извлекаются/принимаются (например, датчики, веб-API и т.д.).

Transformer: это особый тип оператора, отвечающий за преобразование некоторых входных потоков в некоторые выходные. Это очень общее определение не накладывает ограничений ни на саму логику преобразования (которая может быть любой), ни на характеристики входных и выходных потоков данных. Конкретные типы преобразователей, которые расширяют абстрактное определение, имеют свою собственную семантику и требования. Например, для данного преобразователя может потребоваться один входной поток и выполнение суммарного суммирования значений поля.

DataSink: это особый тип оператора, отвечающий за передачу или сохранение заданного потока данных в некоторую внешнюю систему. Приемники данных - это единственные узлы в графе потоков данных, не имеющие выходных потоков, поскольку с топологической точки зрения они являются концом графа. Конкретные типы приемников данных, которые расширяют абстрактное определение, затем используются для моделирования различных внешних систем (например, баз данных, информационных панелей и т.д.).

PartitionedStream: Как упоминалось ранее, одной из основных особенностей распределенных потоковых приложений является то, что их обработка может быть распараллелена путем разделения потока таким образом, чтобы он мог обрабатываться параллельно несколькими экземплярами оператора. Способ разделения потока зависит от логики приложения. Например, он может быть разбит случайным образом на заданное количество секций (т.е. кортежи назначаются доступным секциям

в соответствии со случайным распределением) или на основе ключевого атрибута (т.е. кортежи группируются вместе, образуя секции в соответствии с указанным ключом). Другие стратегии секционирования основаны на специальных алгоритмах планирования (например, Round Robin) для отправки кортежей в разные секции.

WindowedStream: Как упоминалось ранее, чтобы справиться с неограниченностью потоков данных, они могут быть разделены на окна с помощью специальных политик управления окнами. Стоит отметить, что управление окнами потока отличается от его секционирования. Фактически, в то время как секционирование имеет дело с определением нескольких подпотоков, которые затем могут обрабатываться параллельно, оконное управление относится к способу обработки одного потока (или раздела потока), т.е. к разбиению его на блоки (или окна), которые обрабатываются последовательно.

WindowingPolicy: Поток данных может быть преобразован в окно в соответствии с различными политиками управления окнами.

Например, в CountWindow размер основан на количестве кортежей (т.е. каждое окно содержит фиксированное количество кортежей), в то время как во временном окне он основан на временных границах (т.е. каждое окно содержит все кортежи, поступившие в течение временного интервала фиксированной продолжительности). Более того, последующие окна могут перекрываться, и в этом случае политика отображения окон называется скользящей (как для окон Count, так и для окон Time есть атрибут slideSize, который определяет размер слайда). Другие типы окон могут быть определены на основе некоторых характеристик самих данных. Например, SessionWindows обычно определяются на основе некоторого таймаута бездействия, т.е. окно обрабатывается, когда в течение определенного периода времени не было получено никаких данных.

4.4.2. Особенности предлагаемого UML-профиля

StreamUML определяется как UML-профиль. Такой выбор имеет ряд преимуществ. Во-первых, UML, как правило, уже используется на этапах проектирования и известен разработчикам. Более того, доступны инструменты, позволяющие его внедрять, и которые предлагают соответствующие механизмы расширения.

Сначала спецификация UML была проанализирована, чтобы увидеть, что уже предлагается для моделирования потока данных между различными компонентами приложения. UML включает в себя пакет InformationFlows [4.20], который “поддерживает обмен информацией между системными объектами на высоком уровне абстракции”. Информационные потоки обычно определяются вместе с диаграммами классов, используя:

1) Информационный поток, представленный в виде пунктирной стрелки (аналогично зависимости UML) для моделирования однонаправленной передачи некоторой информации между двумя объектами системы (которые могут быть смоделированы как экземпляры двух классов UML),

2) Информационный элемент, расширение классификатора UML, для представления информации, передаваемой с помощью элемента информационного потока.

Согласно спецификации UML [4.20], классы UML также могут быть переданы. С одной стороны, информационные потоки предоставляют довольно слабые выразительные возможности, поскольку они не позволяют моделировать какие-либо детали о природе передаваемой информации или механизмах, с помощью которых она передается между различными компонентами. С другой стороны, они нацелены на тот аспект программной системы, который представляет интерес для потоковых приложений, то есть на поток данных между различными компонентами.

Другими подходящими элементами являются диаграммы действий, которые дают представление о потоках объектов [4.20] для моделирования передачи данных между действиями. Диаграммы действий более известны пользователям UML по сравнению с InformationFlows и, помимо понятия потоков объектов, предлагают также другие функции, которые потенциально полезны в нашем случае. Однако обоснование их использования в StreamGen также выявило некоторые недостатки, основным из которых является тот факт, что пользователям StreamGen придется определять большие модели и знать много деталей об UML, чтобы иметь возможность правильно моделировать свои приложения. Поскольку одной из главных целей StreamGen является упрощение процесса создания прототипов распределенных потоковых приложений, ключевыми показателями являются простота и удобство использования.

В конечном итоге определено использование диаграмм классов и их пакета управления информационным потоком в сочетании с составными структурными диаграммами как оптимальное решение, поскольку это позволяет:

1) точно моделировать потоковые приложения в соответствии с семантикой UML;

2) сохранять подход к моделированию достаточно простым, удовлетворяя требованиям, которые предъявлены к StreamGen.

При таком подходе диаграммы классов используются для определения компонентов приложения и обмена информацией на более высоком уровне абстракции, в то время как диаграммы составных структур используются для представления модели выполнения приложения, при этом экземпляры ранее определенных компонентов и передачи данных моделируются более подробно.

4.4.3. Релевантные стереотипы

В таблице 4.1 представлены наиболее релевантные стереотипы, определенные в StreamUML, которые отражают классы моделей предметной области, определенные в разделе 4.1. Для каждого стереотипа в таблице описаны его тип (второй столбец), обобщенные стереотипы (третий столбец) и расширенные метаклассы UML (четвертый столбец).

Таблица 4.1

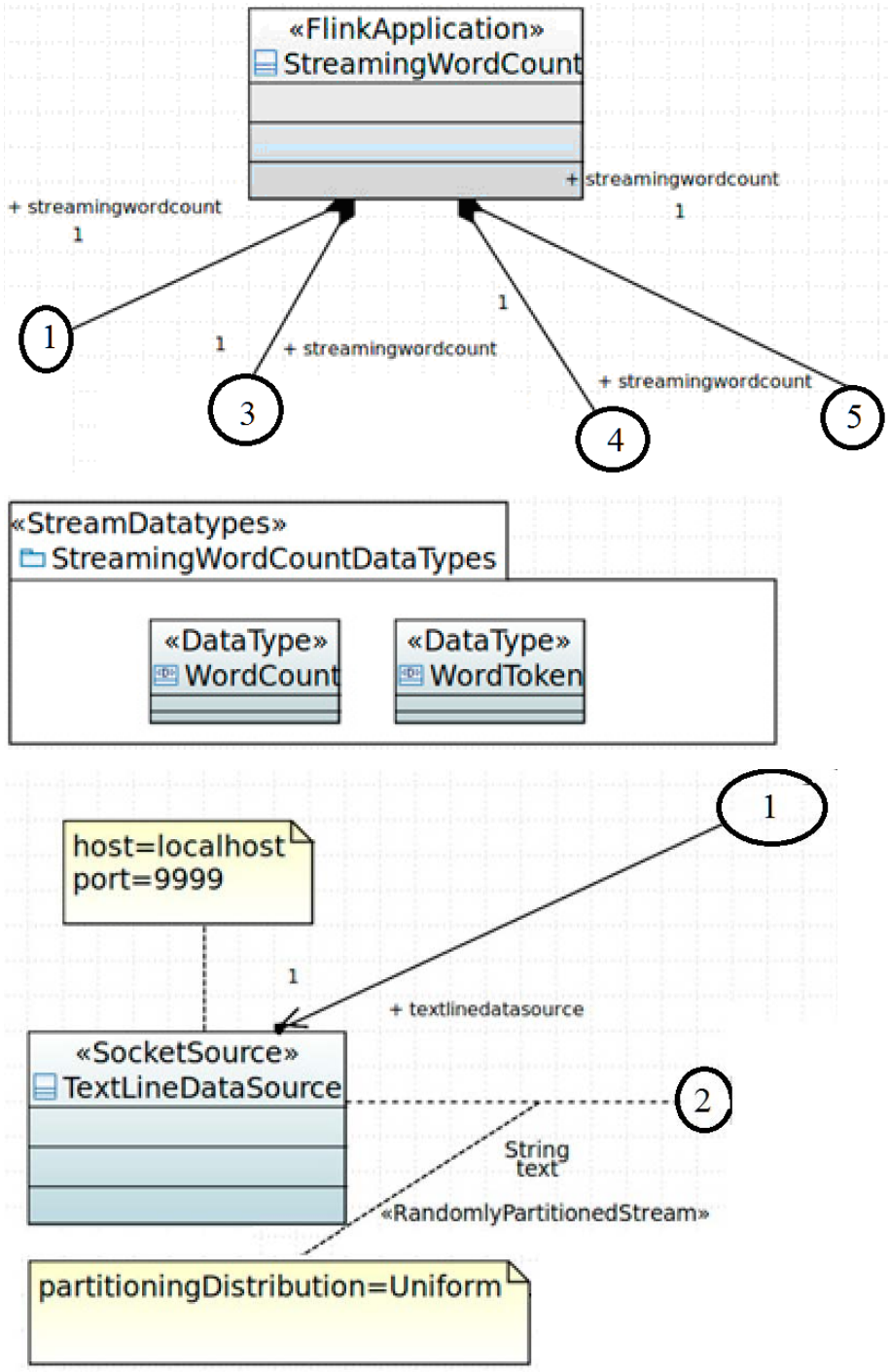
Некоторые стереотипы, представленные профилем StreamUML

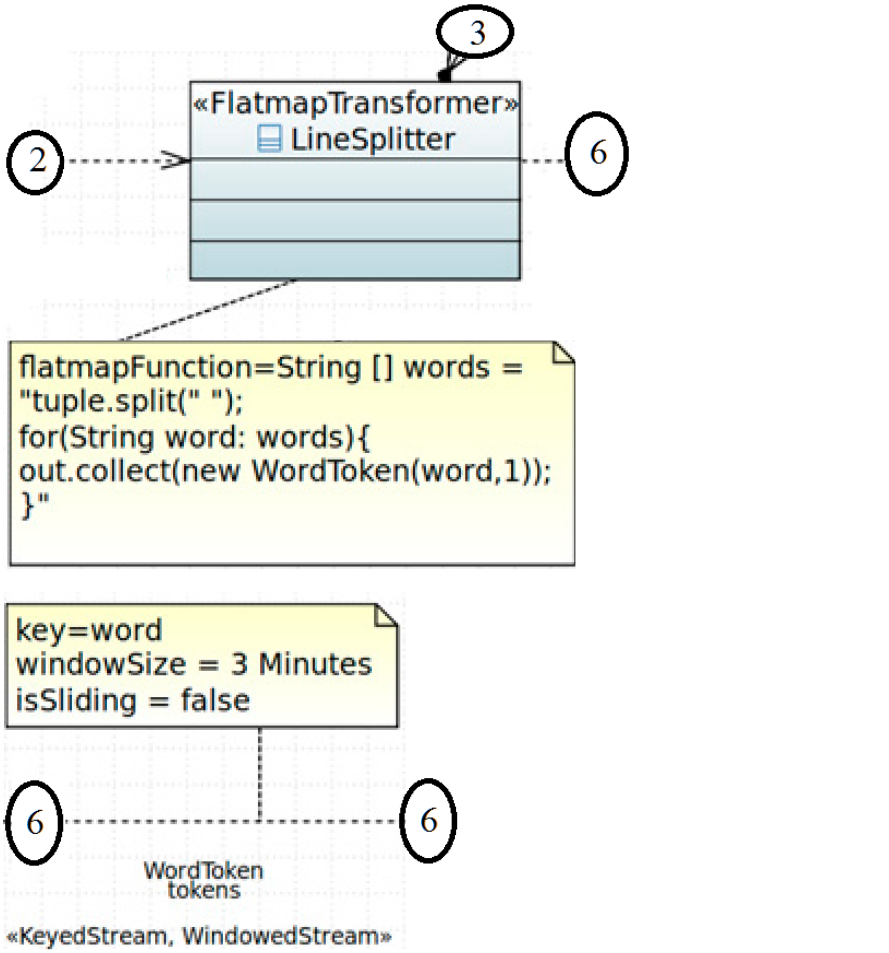
Стереотип	**Тип расширения**	**Стереотип расширения**	**Метакласс**
«DistributedStreamingApplication»	Абстрактный	None	UML::StructuredClassifier
«StreamingOperator»	Абстрактный	None	UML::StructuredClassifier
«DataStream»	Абстрактный	None	UML::Connector
«StreamDataTypes»	Конкретный	None	UML::Package
«FlinkApplication»	Конкретный	«DistributedStreamingApplication»	Нет
«SparkApplication»	Конкретный	«DistributedStreamingApplication»	Нет
«DataSource»	Абстрактный	«StreamingOperator»	Нет
«DataSink»	Абстрактный	«StreamingOperator»	Нет
«Transformer»	Абстрактный	«StreamingOperator»	Нет
«PartitionedStream»	Абстрактный	«DataStream»	Нет

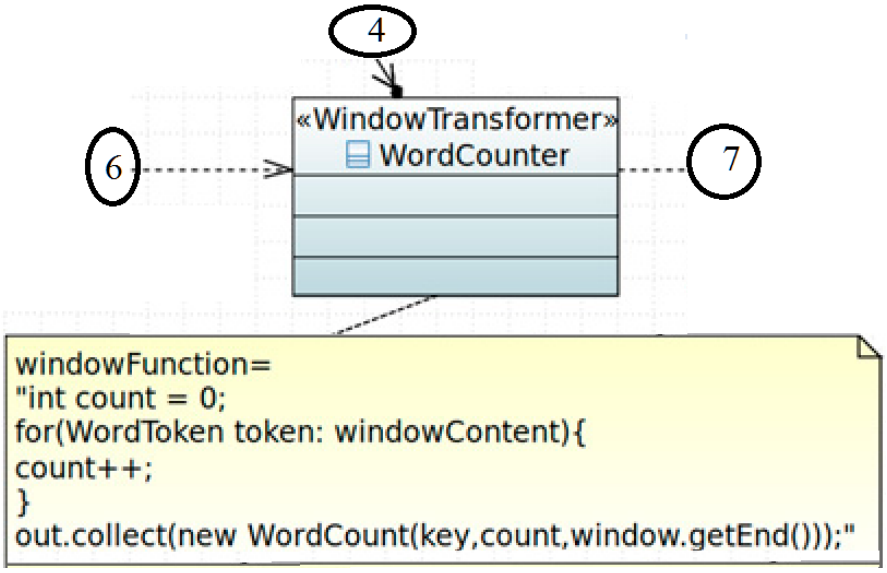
Стереотип	**Тип расширения**	**Стереотип расширения**	**Метакласс**
«WindowedStream»	Конкретный	«DataStream»	Нет
«KeyedStream»	Конкретный	«PartitionedStream»	Нет
«RandomlyPartitioned Stream»	Конкретный	«PartitionedStream»	Нет
«BroadcastedStream»	Конкретный	«PartitionedStream»	Нет
«NonParallelStream»	Конкретный	«PartitionedStream»	Нет
«KafkaSource»	Конкретный	«DataSource»	Нет
«SocketSource»	Конкретный	«DataSource»	Нет
«TextFileSink»	Конкретный	«DataSink»	Нет
«SocketSink»	Конкретный	«DataSink»	Нет
«MapTransformer»	Конкретный	«Transformer»	Нет
«FlatmapTransformer»	Конкретный	«Transformer»	Нет
«SumTransformer»	Конкретный	«Transformer»	Нет
«CountTransformer»	Конкретный	«Transformer»	Нет
«ReduceTransformer»	Конкретный	«Transformer»	Нет
«FilterTransformer»	Конкретный	«Transformer»	Нет
«JoinTransformer»	Конкретный	«Transformer»	Нет
«FlattenTransformer»	Конкретный	«Transformer»	Нет
«NMapTransformer»	Конкретный	«Transformer»	Нет
«NFlatmapTransformer»	Конкретный	«Transformer»	Нет
«WindowTransformer»	Конкретный	«Transformer»	Нет

На рис. 4.3 и 4.4 показана модель StreamUML для приложения Streaming WordCount.

В примечаниях к модели для наглядности выделены свойства, определенные для каждого используемого стереотипа. Далее мы подробно опишем эту модель, а также различные аспекты StreamUML.




3
«FlatmapTransformer»
LineSplitter
2
6
flatmapFunction=String [] words =
"tuple.split(" ");
for(String word: words){
out.collect(new WordToken(word,1));
}"
key=word
windowSize = 3 Minutes
isSliding = false
6
6
WordToken
tokens
«KeyedStream, WindowedStream»


4
«WindowTransformer»
WordCounter
6
7
windowFunction=
"int count = 0;
for(WordToken token: windowContent){
count++;
}
out.collect(new WordCount(key,count,window.getEnd()));"

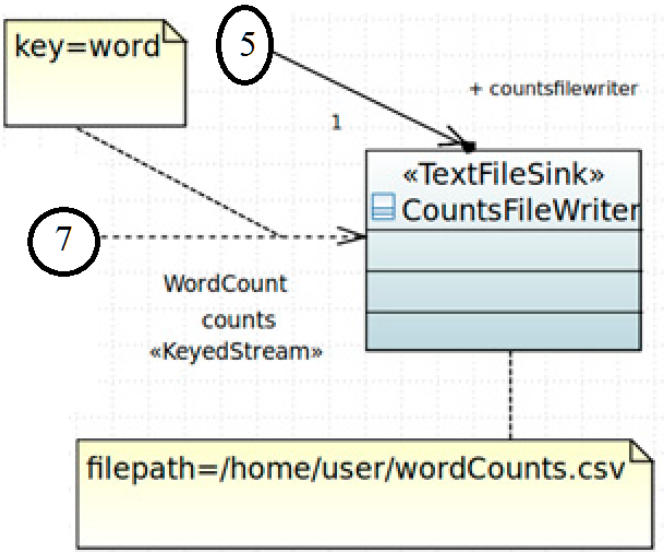
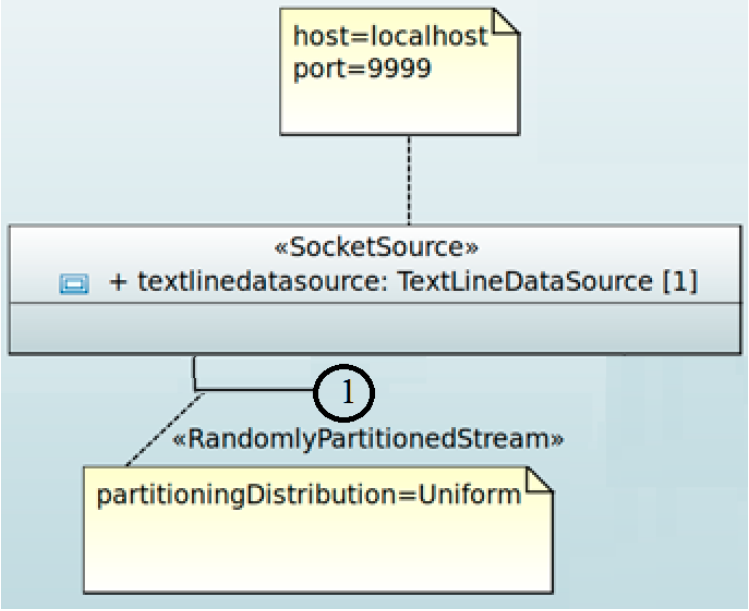
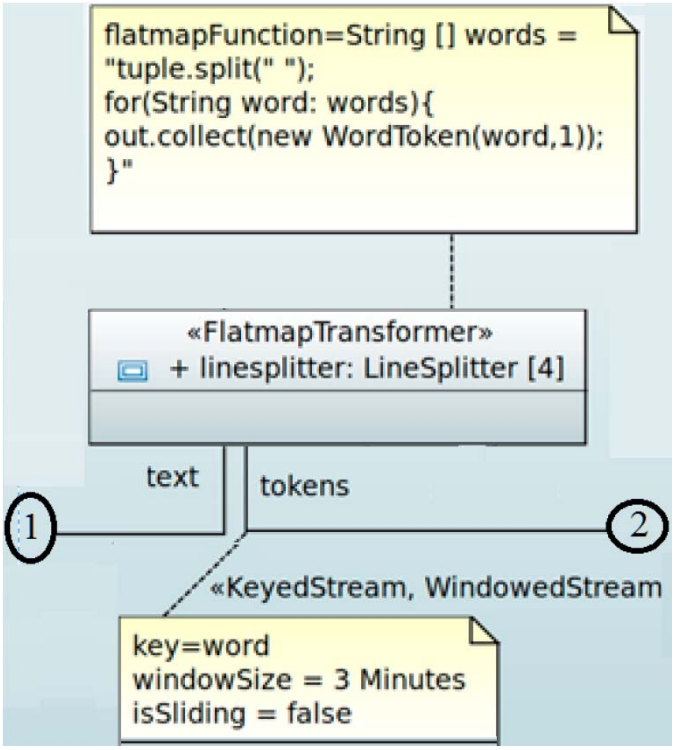
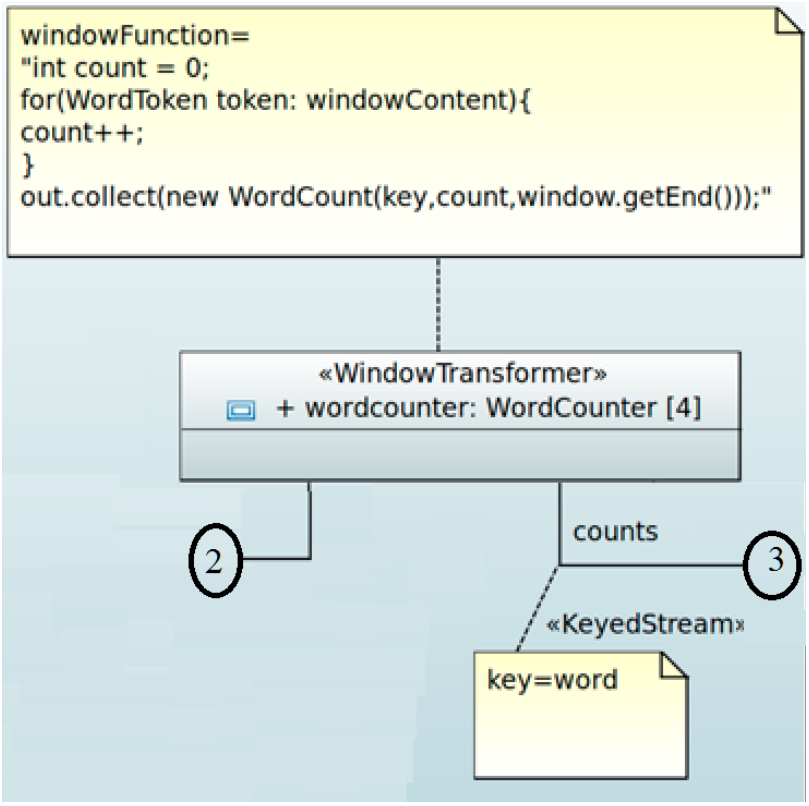


Рис. 4.3. Моделирование приложения StreamingWordCount с использованием StreamUML: диаграмма классов




flatmapFunction=String [] words =
"tuple.split(" ");
for(String word: words){
out.collect(new WordToken(word,1));
}"
«FlatmapTransformer»
+ linesplitter: LineSplitter [4]
text
tokens
1
2
«KeyedStream, WindowedStream
key=word
windowSize = 3 Minutes
isSliding = false


windowFunction=
"int count = 0;
for(WordToken token: windowContent){
count++;
}
out.collect(new WordCount(key,count,window.getEnd()));"
«WindowTransformer»
+ wordcounter: WordCounter [4]
2
counts
3
«KeyedStream»
key=word

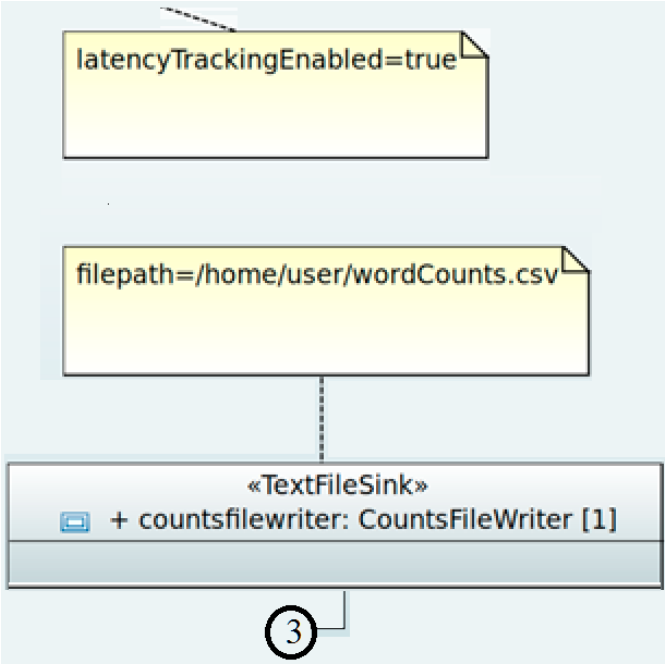


Рис. 4.4. Моделирование приложения StreamingWordCount с использованием предоставленного UMLprofile: Схема составной структуры

4.4.4. Определение топологии приложения.

Функциональная декомпозиция, типичная для потоковых приложений, определяется в терминах типов узлов, моделируемых как классы UML. Например, на диаграмме классов, показанной на рис. 4.3, есть четыре класса, которые затем создаются на диаграмме составной структуры, показанной на рис. 4.4. Более конкретно, в примере показаны источник данных, два узла-преобразователя и приемник. На рис. 4.4 также показано, что эти два преобразователя доступны в четырех экземплярах каждый. Потоки данных моделируются на уровне класса с помощью элементов информационного потока UML, каждый из которых должен передавать объекты (т.е. кортежи) определенного типа. На уровне экземпляра, т.е. на диаграмме составной структуры, потоки данных создаются как должным образом структурированные соединители. В приведенном примере есть три соединителя, которые обычно

обозначаются как «RandomlyPartitionedStream», «KeyedStream, WindowedStream» и «KeyedStream».

4.4.5. Типы данных приложения

Вместо определения специального стереотипа для представления типов данных, которые передаются различными потоками данных, мы разрешаем любому элементу DataType UML выполнять роль типа данных, при этом атрибуты DataType определяют различные поля. В StreamUML типы данных, передаваемые в потоковом приложении, группируются в пакет UML, который обычно называется «StreamDatatypes». На рис. 4.3 мы определяем типы данных WordToken и WordCount и связываем их с потоками tokens и counts соответственно, в то время как текстовый поток, т.е. входной поток приложения, передает данные типа String (уже доступные в UML).

4.5. Операторы потоковой передачи и трансформеры

4.5.1. Операторы потоковой передачи

StreamUML предоставляет четыре абстрактных стереотипа для моделирования узлов графа потока данных, представляющих потоковое приложение. Это стереотипы «StreamingOperator», «DataSource», «DataSink» и «Transformer». Все приведенные конкретные стереотипы являются продолжением этих абстрактных. Например, стереотипы предоставляются для представления различных типов источников данных, таких как стереотип «KafkaSource», который может быть использован для моделирования источника данных, извлекающего данные из Kafka, или «SocketSource», который получает входные данные из соединения через сокет.

В примере запуска StreamingWordCount «SocketSource» получает входные строки текста, прослушивая порт 9999. Аналогичным образом

предоставляются другие типы приемников данных (например, текстовые файлы, базы данных и т.д.). Опять же, в примере приложения StreamingWordCount используется простой «TextFileSink» для записи вычисленного количества слов в определенный текстовый файл.

4.5.2. Трансформеры

Остальные стереотипы предназначены для моделирования трансформеров.

Для некоторых трансформеров разработчику приложения не нужно проходить какой-либо этап кодирования, поскольку их семантика полностью определена. Например, стереотип «SumTransformer» моделирует оператор, который, учитывая числовое поле входного потока данных, постоянно обновляет и выводит, по мере поступления новых кортежей, совокупную сумму значений полей. Для некоторых других преобразователей разработчик гибко определяет соответствующую логику, поскольку они накладывают лишь некоторые ограничения на тип входных и выходных потоков. Например, «MapTransformer» преобразует каждый входящий кортеж входного потока данных в один и только один кортеж выходного потока данных. «WindowTransformer» использует в качестве входных данных поток оконных данных и преобразует входящие окна кортежей в любое количество кортежей выходного потока данных. Основная логика всех этих преобразований должна быть определена разработчиком. В частности, каждый из этих стереотипов предоставляет определенное свойство, в котором разработчик непосредственно записывает логику преобразования в терминах фрагментов Java-кода. При этом она может полагаться на следующие переменные:

• tuple: это позволяет нам получить доступ к текущему обрабатываемому кортежу и доступно в функциях, которые обрабатывают один кортеж за раз (например, «MapTransformer» и «FilterTransformer»).

• key: позволяет нам получить доступ к ключу, связанному с текущим обрабатываемым кортежем, и доступен только в тех функциях, входной поток которых разделен на основе заданного ключа (см. далее).

• out: Это обеспечивает метод collect для создания кортежей в выходном потоке. Он доступен в преобразователях, которые позволяют создавать несколько кортежей в своем потоке выходных данных (например, «FlatmapTransformer» и «WindowTransformer»).

• window: Это специальный объект, доступный в «WindowTransformer», который позволяет нам получать доступ к метаданным окна, таким как время его начала и окончания.

• windowContent: это итератор Java для доступа к кортежам, содержащимся в окне, обрабатываемом «WindowTransformer».

В случае приложения StreamingWordCount «FlatmapTransformer» (LineSplitter) принимает в качестве входных данных поток текстовых данных (который передает полученные строки текста), разбивает каждую строку текста на отдельные слова и для каждого слова создает новый кортеж в выходном потоке данных (именованные токены). Последний передает кортежи типа WordToken и управляется клавишами и окнами, как описано ранее. Затем он обрабатывается «WindowTransformer», тело которого просто подсчитывает количество кортежей в текущем обрабатываемом окне и выводит новый кортеж (типа WordCount) в поток wordCounts. Во время генерации кода каждый фрагмент кода статически проверяется, чтобы избежать генерации кода, который не будет компилироваться. В частности, StreamCGM проверяет синтаксическую корректность каждого фрагмента кода и то, что он использует только локально определенные переменные, а также переменные, предоставляемые конкретным преобразователем (например, кортеж, ключ и т.д.). На момент написания разработчики не могли полагаться на такие

функции, как автозавершение, подсветка синтаксиса или другие типичные возможности IDE при написании пользовательской логики transformer.

4.5.3. Параллельное выполнение

Каждый оператор в графе потока данных может быть распараллелен базовой платформой выполнения с учетом ограничений, определенных во время разработки. Степень параллелизма определяется на диаграмме составной структуры путем задания количества экземпляров данного класса с помощью свойства множественности. Чтобы позволить параллельным репликам работать с правильно разделенными данными, StreamUML поддерживает различные стратегии секционирования, такие как случайное секционирование (с использованием различных распределений), секционирование на основе ключей и сбалансированное секционирование. В случае приложения WordCount, TextLineDataSource и CountsFileWriter не распараллеливаются, в то время как операторы LineSplitter и WordCounter распараллеливаются с коэффициентом 4. В то время как первый принимает в качестве входных данных текст «RandomlyPartitionedStream», второй принимает поток данных токенов, который разбивается на разделы, используя стереотип «KeyedStream», в соответствии со значением поля word каждого токена (т.е. все токены с одинаковым словом принадлежат к одним и тем же разделам).

4.5.4. Моделирование StatefulTransformers

Некоторые трансформаторы могут быть статусными, что означает, что они сохраняют внутреннее состояние и управляют им, что необходимо, учитывая тип выполняемого трансформатора. Примером может служить трансформатор с подвижной суммой, смоделированный на основе стереотипа «SumTransformer». По сути, преобразователь скользящей суммы сохраняет в качестве внутреннего состояния последнее полученное

значение, чтобы получить обновленный результат при поступлении следующего кортежа. В общем, любой тип состояния может храниться внутри (объекты, карты, списки). StreamUML позволяет моделировать внутреннее состояние, сохраняемое и используемое универсальными преобразователями, например, «MapTransformer», «FlatmapTransformer», «WindowTransformer» и так далее. Внутреннее состояние может быть смоделировано путем определения одного или нескольких элементов свойств UML для класса, представляющего transformer. Затем к таким элементам свойств можно получить доступ и изменить их в теле transformer.

4.5.5. Политика управления окнами

Используя стереотип «WindowedStream», можно настроить окно для определенного потока данных. Стереотип позволяет задать различные политики управления окнами, описанные в разделе 4.3.1. В частности, поддерживаются окна, основанные на времени, и окна сеанса. В случае запущенного примера StreamingWordCount в потоке данных токенов было указано фиксированное временное окно продолжительностью 3 минуты.

4.5.6. Настройки, зависящие от платформы моделирования

Некоторые платформы требуют определения параметров, специфичных для конкретной платформы, но зависящих от приложения. Например, поскольку Spark основан на подходе к микропакетной обработке [4.29], требуется определить размер микропакета.

Этот параметр зависит от способа обработки данных в Spark, но, как правило, не является необходимым для определения прикладной логики потокового приложения. Тем не менее, при внедрении Spark разработчик может захотеть настроить этот параметр, поскольку он влияет на производительность приложения. StreamUML предоставляет специальный

шаблон для каждой поддерживаемой платформы, позволяющий настраивать различные параметры, зависящие от платформы. Примерами могут служить стереотип «SparkApplication», предоставляющий, например, свойство microBatchSize, или стереотип «FlinkApplication», который предоставляет свойство latencyTrackingEnabled (см. рис. 4.4). Эти дополнительные стереотипы расширяют метакласс UML Model и должны применяться к UML-моделям, содержащим потоковые приложения, для определения и настройки целевой платформы. Модель должна указывать не более одной целевой платформы, обладая не более чем одним из таких стереотипов, зависящих от платформы. Это ограничение связано с тем, что StreamCGM использует этот стереотип, чтобы определить целевую платформу, для которой должен быть сгенерирован код.

4.5.7. Проверка согласованности моделей

Мы добавили в StreamUML ряд ограничений, которые можно задать на диаграмме классов, чтобы помочь разработчикам создавать корректные модели. Такие ограничения были заданы непосредственно в Stream UML с использованием OCL и, таким образом, могут быть проверены в интерактивном режиме при создании модели. Ограничения также гарантируют, что модель может быть переведена в код без ошибок.

Другие ограничения накладывают другие правила моделирования, например, источники данных не должны иметь входного потока и должны иметь один выходной поток (и наоборот, для приемников данных), входной поток «WindowTransformer» должен быть «WindowStream». Кроме того, мы определили ряд зависящих от платформы ограничений OCL для проверки модели на соответствие конкретным ограничениям, налагаемым целевой платформой выполнения.

Стоит отметить, что в наших ограничениях OCL мы используем примитив getAppliedStereotypes(), который не является стандартным, а

специфичен для реализации UML2 в Eclipse. Если, с одной стороны, это ограничивает удобство использования StreamUML в контексте конкретной платформы (той, на основе которой был разработан инструмент StreamGen), с другой стороны, это предоставляет нам практическое решение для использования стереотипных элементов UML в рамках ограничений OCL. Более того, другие платформы могут предоставлять другие средства для доступа к стереотипам, связанным с элементами моделирования, что позволит определить эквивалентные ограничения. В худшем случае подмножество наших ограничений OCL будет недоступно в реализациях StreamUML на платформах, отличных от Eclipse.

4.6. Генерация кода

В этом разделе мы сосредоточимся на StreamCGM, то есть на модуле генерации кода StreamGen. StreamCGM основан на технологии шаблонов и реализован в Acceleo. Генераторы кода Spark и Flink, входящие в StreamCGM, генерируют Java-код. Это решение было принято из-за того, что большинство проанализированных потоковых платформ предоставляют Java API. Некоторые из них на самом деле предоставляют реализации API также для других языков, таких как Scala или Python, но это менее распространенный случай, и часто такие реализации все еще незрелые и нестабильные.

StreamCGM принимает в качестве входных данных модели StreamUML и начинает поиск стереотипа, который идентифицирует выбранную целевую платформу (т.е. стереотипы «SparkApplication» и «FlinkApplication»).

Исходя из этого, он активирует соответствующий генератор кода, который запускается с создания проекта Maven, уже настроенного и связанного со всеми необходимыми зависимостями целевой платформы.

Если используются определенные типы источников/приемников данных (например, «KafkaSource» или «CassandraSink»), добавляется зависимость Maven для импорта соответствующего драйвера. Затем сгенерированный код структурируется в три пакета: один для типов данных, используемых приложением, другой для пользовательских функций и третий для конструктора приложений и средства запуска. Как следствие, генерация кода состоит из трех этапов:

1) **Генерация типов данных**: на этом этапе генератор кода создает новый пакет, который заполняется необходимыми классами для всех типов данных, используемых в модели. В частности, при генерации выполняется поиск пакетов UML, помеченных стереотипом «StreamDatatypes», и для каждого вложенного типа данных создается соответствующий простой старый объект Java (POJO). Этот этап не зависит от выбранной целевой платформы.

2) **Генерация потоковых функций**: на этом этапе генератор кода создает новый пакет, который заполняется необходимыми классами, реализующими различные пользовательские операторы, присутствующие в модели. В частности, генератор кода ищет мета-типизированные объекты «StructuredClassifier», помеченные стереотипом, который требует от пользователя указать логику преобразования (например, «MapTransformer», «WindowTransformer» и т.д.) и для каждого из них создает новый класс Java, который реализует определенные интерфейсы, предоставляемые выбранная целевая платформа.

3) **Генерация потокового задания**: на этом этапе генератор кода создает новый пакет, содержащий фактическое приложение, которое должно быть выполнено, т.е. класс Java, который создает и выполняет график потока данных, определенный в модели. Например, в листинге 6 приведена выдержка из кода Flink, сгенерированного из модели приложения StreamingWordCount, которая отвечает за сборку и

выполнение графика потока данных. В этом классе различные функции, созданные на втором этапе (например, LineSplitter, WordCounter), создаются и подключаются с использованием потоков данных (например, текста, токенов, подсчетов), которые передают кортежи различных типов данных, определенных на первом этапе (например, WordToken, WordCount).

Хотя рабочий процесс генерации одинаков, генераторы, ориентированные на разные платформы, работают по-разному (за исключением первого этапа). Например, ссылаясь на «SumTransformer», Flink предоставляет явные API для вычисления совокупной суммы по числовому полю потока данных, в то время как Spark этого не делает. Как следствие, в то время как генератор Flink должен выводить только одну строку кода, генератор Spark должен добавлять код более низкого уровня, что в нашей текущей реализации приводит к более чем 20 строкам кода. Другим примером является случай, когда потоки данных отображаются в окне с использованием фиксированного временного окна. Flink явно позволяет устанавливать фиксированное временное окно для потока данных, в то время как в Spark это можно получить как дополнительный вариант скользящего временного окна, в котором длина слайда равна размеру самого окна.

Разработана структура программного прототипа системы создания потоковых приложений для обработки данных на основе моделей потоков данных.

Рис. 4.5. Структура программного прототипа системы создания потоковых приложений для обработки данных на основе моделей потоков данных

4.7. Оценка

При оценке StreamGen проведен эксперимент по смешанному методу с целью оценки различных аспектов StreamGen, связанных с выполнением требований. В частности, мы провели следующий набор экспериментов:

• Чтобы поэкспериментировать с созданием приложения, достаточно сложного для использования большинства элементов моделирования, предоставляемых StreamUML, создан пример под названием TempTrack для отслеживания температуры, который объединяет 11 различных

операторов и 10 потоков и интегрирует как Kafka, так и Cassandra, что демонстрирует хороший уровень сложности.

• Чтобы продемонстрировать способность StreamGen поддерживать моделирование реальных потоковых приложений и генерировать их код для различных потоковых платформ с производительностью, сравнимой с кодом, разработанным вручную, мы смоделировали и сгенерировали код для Yahoo streaming benchmark, который включает в себя уже существующие реализации одного и того же приложения как для Flink, так и для Spark. Этот эксперимент показывает, с одной стороны, способность StreamGen справляться с приложениями реального мира, с другой стороны, что производительность сгенерированного кода очень близка к той, которую предлагают ручные реализации .

• Чтобы оценить полноту использования StreamUML, мы сравнили его возможности моделирования, то есть предоставляемые концепции и функции потоковой передачи, с моделью Google Dataflow. Последняя считается самой современной моделью для систем потоковой обработки. Мы показываем, что StreamGen способен поддерживать большинство предложенных элементов моделирования.

• Чтобы оценить расширяемость StreamGen, добавлена поддержка Apache Apex.

4.7.1. Практический пример: Обработка данных с датчиков

Чтобы продемонстрировать практическую применимость, мы использовали StreamGen для разработки тематического приложения, которое мы назвали TempTrack, которое принимает в качестве входных данных и обрабатывает значения температуры, измеряемые различными датчиками, расположенными в помещениях нескольких зданий. Структурная схема StreamGen Composite для этого примера показана на рис. 4.5. Детали опущены из соображений экономии места, но полная

модель и сгенерированный код доступны в Интернете. TempTrack выполняет следующие действия:

• Данные принимаются от датчиков температуры (через компонент TemperatureSensor в модели) и передаются в операцию TemperatureParser, которая преобразует их в объекты типа RoomTemperature (смотрите переданный тип потока parsedTemperatures).

• Данные о комнатной температуре передаются в ClearRawData для очистки, а затем сохраняются в базе данных Cassandra для последующего анализа.

• Данные о температуре в помещении также передаются в систему ComputeRoomStatistics, которая вычисляет среднюю и максимальную температуру в каждой комнате за 2 минуты. Результаты передаются в приемник, который создает внешний csv-файл, доступный для дальнейшего анализа.

• Статистика также передается в MonitorCriticalPrediction, который непрерывно прогнозирует среднюю температуру в каждой комнате в течение следующих 10 минут и передает предупреждение в очередь Kafka, если прогнозируемое значение превышает заданный порог.

• Наконец, та же статистика передается в CheckCriticalTemperature, которая, в свою очередь, передает предупреждение в очередь Kafka, если средняя температура в помещении за последние 3 секунды превышает заданный порог.

TempTrack интегрируется с двумя внешними компонентами (Kafka и Cassandra) и предлагает различные концепции потоковой передачи данных и в общей сложности 11 операторов. Кроме того, он демонстрирует довольно жесткие ограничения в режиме реального времени для обнаружения и прогнозирования критических событий и высокую скорость срабатывания датчиков. Таким образом, его можно считать нетривиальным примером, подходящим для проверки выразительности

StreamUML и эффективности генерации StreamCGM. На основе приведенной выше модели, которая подходит для описания функций приложения, мы смогли сгенерировать и выполнить код как для Spark, так и для Flink (код доступен по той же ссылке, что и модель). Более того, измерив задержку, связанную с обеими реализациями TempTrack, мы смогли подтвердить, что код, сгенерированный для Flink, лучше соответствует нашим требованиям, как и ожидалось, учитывая микропакетный характер Spark, который не является лучшим выбором для минимизации задержки.

4.7.2. Моделирование оценки потоковой передачи Yahoo

Тест Yahoo streaming benchmark был разработан для сравнения производительности различных платформ потоковой передачи. Он состоит из приложения для анализа рекламных событий (ad events). Приложение получает рекламные события (например, просмотры, клики и т.д.) через экземпляр Kafka. Каждое рекламное событие анализируется, фильтруется, очищается и дополняется идентификатором рекламной кампании, к которой оно относится. Такое соответствие извлекается из базы данных Redis. Затем рекламные события группируются по кампаниям и временным интервалам в 10 секунд. Наконец, подсчитывается количество рекламных событий для каждой кампании и для каждого окна и сохраняется в базе данных Redis. Для Storm, Spark, Flink и Apex доступны тестовые версии, которые позволяют собирать и сравнивать показатели производительности, в частности, задержку вычислений.

Мы смоделировали тест с помощью StreamGen, чтобы показать, что он предлагает по крайней мере те элементы дизайна, которые требуются для этого стороннего приложения. Более того, мы сгенерировали соответствующий код для двух платформ, которые мы поддерживаем, и сравнили полученный результат с тем, который изначально был

разработан Yahoo, как с точки зрения качества кода, так и производительности.

На рис. 4.6 показана разработанная модель (некоторые комментарии, касающиеся настройки стереотипов, были опущены для экономии места). Действительно, тест основан на функциях потоковой передачи, которые в настоящее время поддерживаются StreamGen, таких как управление окнами, разделение на основе ключей (т.е. рекламные события группируются по рекламной кампании), операции с плоской картой и фильтрацией, а также вычисления с учетом состояния. StreamGen также поддерживает Kafka в качестве источника данных. Единственное ограничение, которое мы обнаружили, было чисто технологическим, поскольку StreamGen в настоящее время не поддерживает Redis в качестве источника/приемника данных.

Чтобы преодолеть это ограничение, в нашем эксперименте результаты обработки заявок сохраняются обратно во вторую очередь Kafka (смотрите ссылку на кампании «KafkaSink»). Более того, сопоставление между объявлениями и кампаниями извлекается в начале выполнения из текстового файла и затем сохраняется в памяти с помощью «NFlatmapTranformer» CampaignJoin, который отвечает за сопоставление событий с кампаниями. Это позволило нам протестировать поддержку StreamGen для преобразователей с отслеживанием состояния, при этом соответствие между объявлениями и кампаниями является внутренним состоянием. Модель вместе со сгенерированным кодом доступна онлайн.

Мы сгенерировали код как для Flink, так и для Spark и сравнили его с оригинальными реализациями.

Что касается Flink, то структура кода нашей сгенерированной реализации похожа на исходную и содержит всего на 9% больше строк кода (см. первую строку, первые два столбца таблицы 4.2). На самом деле, как и следовало ожидать от инструмента разработки, основанного на

моделях, StreamGen генерирует некоторый стандартный код. Например, он не объединяет несколько преобразователей для создания единого результирующего потока данных, а генерирует определенный код для каждого элемента InformationFlow, даже для тех, которые являются всего лишь промежуточными (и фактически никогда не использовались) потоками в цепочке преобразователей. Другим примером являются методы toString(), hashCode() и equals(), которые всегда переопределяются при генерации типов данных. Что касается Spark, мы не смогли сравнить код, поскольку исходная реализация написана на Scala, в то время как StreamGen генерирует Java-код.

Показатель производительности, предоставляемый Yahoo benchmark, - это задержка вычислений для каждого создаваемого окна кампании, то есть время, “прошедшее с момента, когда в Kafka было отправлено последнее событие для этого конкретного окна кампании, и когда оно было записано в Redis”. Для наших экспериментов мы использовали приложения, созданные StreamGen, для измерения той же метрики, заменив Redis на Kafka. Мы также позаботились о том, чтобы использовать те же входные данные, которые использовались в тестировании. В частности, тест Yahoo benchmark случайным образом генерирует рабочую нагрузку: сначала он генерирует сопоставление между объявлениями и кампаниями и сохраняет это сопоставление в Redis; затем он начинает загружать рекламные события в Kafka. В нашем эксперименте мы использовали тот же генератор загрузки, который был скорректирован таким образом, чтобы не записывать сопоставление между объявлениями и кампанией в Redis, а сохранять его в CSV-файл, который затем считывается сгенерированным приложением (см. AdToCampaignSource «Текстовый файл» на рис. 4.6).

Рис. 4.6. Модель StreamUML для примера применения Temp Track

Мы запустили как оригинальный тест Yahoo benchmark, так и сгенерированные StreamGen версии (как для Flink, так и для Spark) на экземпляре Amazon EC2 m5d.xlarge, который поставляется с 16 ГБ оперативной памяти и четырьмя процессорными ядрами. В каждом эксперименте мы устанавливали параллелизм приложений равным 4, т.е. у каждого оператора было 4 реплики, и мы вычисляли среднюю задержку вычислений во всех созданных окнах кампании. Это среднее значение затем усреднялось по десяти запускам каждого приложения.

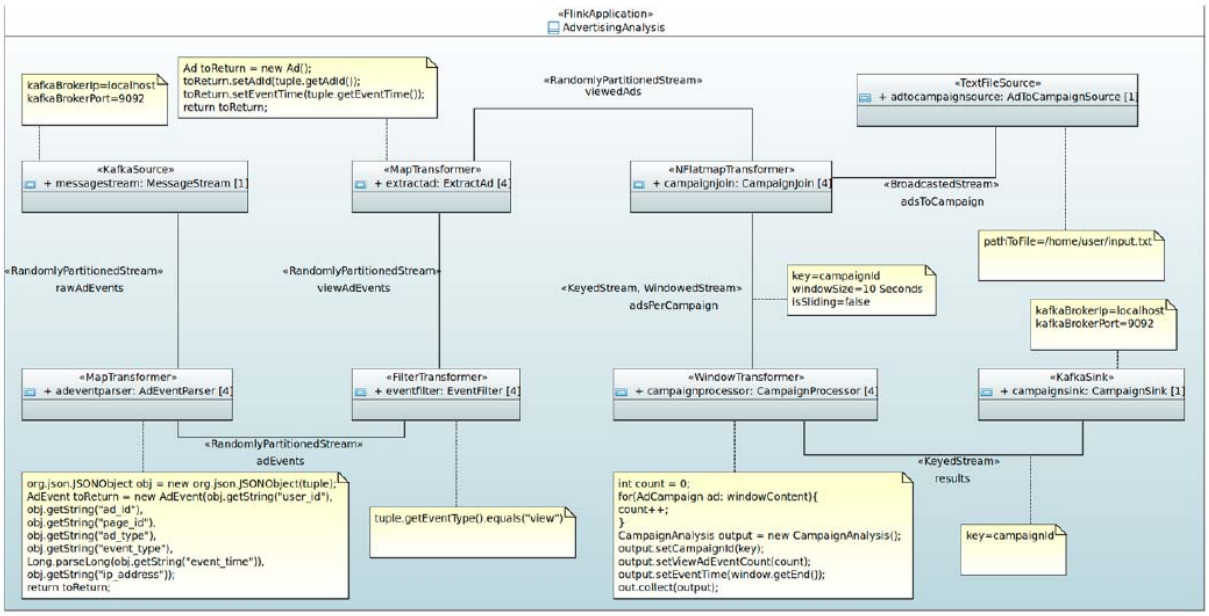


Рис. 4.7. Модель StreamUML для оценки Yahoo streaming

Таблица 4.2

Анализ реализации StreamGen, созданной в тесте Yahoo Benchmark

	FlinkOrig	FlinkStreamGen	SparkOrig	SparkStreamGen
Объем кода	608	666	-	-
Задержка, мс	9957.84	10433.82	9984.83	10704.73

Во второй строке таблицы 4.2 представлены результаты этих экспериментов. Приложение, сгенерированное с помощью Streamgen, работает в целом хуже всего, как в случае Spark, так и в случае Flink, при

этом средняя задержка увеличивается на 4,78% в случае Flink и на 7,21% в случае Spark, что является приемлемым, хотя и не пренебрежимо малым показателем. Возможное объяснение этому можно найти в структурах данных, которые внутренне используются различными операторами. В частности, в приложениях, созданных StreamGen, оператор CampaignJoin сохраняет сопоставление между объявлениями и кампаниями в виде списка объектов типа AdCampaign. Когда появляется новое рекламное событие, оператор выполняет линейный поиск по этому списку, чтобы найти соответствующую кампанию. Вместо этого в исходном тесте Yahoo benchmark сопоставление хранится в виде хэш-карты с постоянным временем доступа. Причина, по которой сгенерированные приложения используют список, заключается в том, что текущая реализация StreamGen не предоставляет способа моделирования хэш-карты как внутреннего состояния оператора с отслеживанием состояния. Расширение StreamGen в этом направлении может приблизить производительность к исходному тесту.

4.7.3. Оценка реализации функции потоковой передачи данных в сравнении с Google Dataflow

Модель Google Dataflow [4.2] была в качестве эталонной модели для описания возможностей, которые должен предлагать современный потоковый процессор. Основная идея, лежащая в основе модели Google Dataflow, заключается в преодолении разрыва между масштабными требованиями к обработке, предъявляемыми современными потоковыми источниками данных, и сложными функциональными требованиями, которые определили поставщики потоковых приложений, такими как возможность работы с различными окнами и временной семантикой, а также корректность результатов и задержка вычислений и затраты при обработке крупномасштабных потоков данных. Модель учитывает

различные аспекты, которые обобщены в таблице 4.3 и сопоставлены с тем, что предлагает Streamsend.

Таблица 4.3

Сравнение StreamGen с Google Dataflow

Категория	**Особенность модели потока данных**	**В потоковом режиме**	**Как это можно смоделировать**
Типы операций обработки данных	ParDo	Да	FlatmapTransformer
	groupByKey	Да	KeyedStream
	Выравнивание	Да	FlattenTransformer
	Объединение	Да	JoinTransformer
	CompositeTransform		-
	Побочные входы	Да	Преобразователь с несколькими входными потоками
	Исходный API	Да	Вспомогательные стереотипы источника данных
	Метрика		-
	Обработка с учетом состояния	Да	Состояние, моделируемое как атрибуты оператора
Разделение потоков данных	Глобальные окна	Да	WindowedStream + NonParallelStream
	Фиксированные окна	Да	Моделируется с помощью WindowedStream
	Скользящие окна	Да	Моделируется с

Категория	**Особенность модели потока данных**	**В потоковом режиме**	**Как это можно смоделировать**
			помощью WindowedStream
	Окна сеанса	Да	Моделируется с помощью WindowedStream
	Пользовательские окна	Нет	-
Обработка начального окна	Триггеры	Нет	триггеры времени обработки по умолчанию
Обработка улучшений предыдущих результатов	Накопление	Нет	-
	Отбрасывание	Нет	-
	Накопление и удаление	Нет	- (в настоящее время поддерживается непотоковым процессором)

Что касается типов операций обработки данных, StreamGen охватывает почти все выявленные аспекты, за исключением:

1) возможности определения составных преобразований, т.е. пользовательских преобразований, полученных путем составления примитивных преобразований, доступных "из коробки",

2) возможности указывать показатели, которые будут собираться во время выполнения.

Что касается методов разделения потоков данных, StreamGen поддерживает моделирование всей возможной семантики управления окнами, определенной Google Dataflow Model, но на данный момент она не позволяет задавать пользовательские (т.е. определяемые пользователем) политики управления окнами. Что касается механизмов запуска обработки окон, StreamGen не позволяет разработчикам указывать пользовательские триггеры, которые определяют, когда окна следует считать готовыми к обработке. На данный момент он поддерживает только одну семантику запуска, которая закрывает окно по истечении указанного периода времени. Наконец, в StreamGen отсутствует возможность определять, как обрабатываются уточнения предыдущих результатов. Например, при использовании политики накопления содержимое окна остается неизменным в памяти, а более поздние кортежи приводят к повторной обработке окна и повторному вычислению результатов. Семантика по умолчанию, поддерживаемая StreamGen на данный момент, является отбрасывающей, т.е. после выдачи результатов содержимое окна отбрасывается, и более поздние результаты не имеют никакого отношения к предыдущим.

StreamGen в настоящее время поддерживает различные функции, определенные моделью потока данных, но также имеет ряд ограничений. Большинство из отсутствующих в настоящее время функций, таких как возможность указать, как обрабатываются уточнения предыдущих результатов или какие показатели необходимо собирать во время выполнения, можно было бы указать в качестве конфигураций для различных операторов потоковой передачи. Другие функции, такие как возможность определения сложных преобразований, могут потребовать определения некоторых новых стереотипов.

Таблица 4.4

Распределение предполагаемого уровня сложности (шкала Лайкерта) предлагаемого приложения, подлежащего разработке

очень простой	простой	средний	сложный	очень сложный
0%	47.8%	34.8%	4.3%	13%

4.7.4. Предварительная оценка расширяемости StreamGen

StreamGen - это решение, позволяющее справиться с технологической неоднородностью в области распределенных потоковых платформ. Основная идея заключается в том, чтобы позволить пользователям StreamGen быстро ознакомиться с потоковыми приложениями с помощью StreamUML и легко и непринужденно экспериментировать с несколькими потоковыми платформами. Из-за этого ожидается, что StreamGen будет расширяемым, таким образом, новые платформы потоковой передачи данных можно будет легко и независимо подключать. В этом разделе мы расскажем о нашем собственном опыте расширения StreamGen для поддержки Apache Apex. Мы описываем шаги, необходимые для добавления в StreamGen поддержки дополнительных фреймворков потоковой передачи, показывая, что усилия по расширению StreamGen сокращаются благодаря предоставленным абстракциям и способу его разработки.

Очевидно, что первым обязательным шагом для того, кто хочет расширить StreamGen, является изучение и приобретение существенного опыта работы с добавляемым фреймворком. Это совершенно очевидно, учитывая, что, если мы хотим освободить пользователей StreamGen от необходимости знакомиться с несколькими фреймворками, кто-то должен проделать тяжелую работу по правильному сопоставлению специфических концепций фреймворка с абстракциями StreamUML.

После этого предварительного шага можно приступать к фактическому расширению StreamGen. С точки зрения расширяемости, основными преимуществами подхода StreamGen являются:

1) отсутствие связи между языком, используемым для определения приложения, и его конкретной реализацией,

2) модульная архитектура StreamCGM.

С одной стороны, единственное изменение, которое необходимо внести в StreamUML, - это добавление еще одного стереотипа, расширяющего «DistributedStreamingApplication», чтобы позволить добавляемой платформе потоковой передачи быть установленной в качестве целевой платформы для данной модели приложения. Например, в случае с Apex мы добавили стереотип «ApexApplication». С другой стороны, StreamCGM нуждается в дополнительном модуле, который инкапсулирует логику генерации кода для добавляемой платформы потоковой передачи. Тем не менее, ожидается, что генерация типов данных будет одинаковой для каждой потоковой платформы, поскольку типы данных на самом деле являются просто POJO.

Как следствие, добавление такого модуля генерации кода требует разработки:

• генерирующей логики для различных типов источников, приемников и преобразователей;

• генерирующей логики для сборки потока данных приложения;

• генерирующей логики для файла pom.xml, необходимого для управления созданным проектом.

Согласно нашему опыту добавления поддержки Apex (и фактически то же самое произошло при добавлении поддержки Spark), наиболее сложным этапом является определение генерирующей логики для различных типов преобразователей. Основными препятствиями являются различные способы, с помощью которых различные фреймворки

изначально поддерживают потоковые операции (например, управление окнами) и операторы более высокого уровня (например, суммирование, подсчет, фильтрация и т.д.). Действительно, при наличии встроенной поддержки генерирующая логика в большинстве случаев довольно проста, поскольку она состоит из, при правильном вызове функций. И наоборот, когда встроенная поддержка недоступна, как в случае «WindowTransformer» или «SumTransformer» в Spark, генерирующая логика становится более сложной, поскольку она должна генерировать реализацию таких операторов, которая использует API более низкого уровня. Например, чтобы сгенерировать «WindowTransformer» для Spark, к входному потоку применяется оконная функция reduce, которая объединяет входящие кортежи в окна. Затем они обрабатываются функцией flatmap, которая инкапсулирует фактическую логику обработки окон.

Расширение StreamGen в основном означает добавление модуля генерации кода в StreamCGM и что это может быть сделано довольно систематическим образом. Более того, необходимые усилия для разработки такого модуля строго зависят от собственных возможностей потоковой передачи добавляемой платформы. Также стоит отметить, что StreamCGM относится к конкретным версиям поддерживаемых целевых платформ. В частности, в настоящее время StreamCGM генерирует код для Flink 1.4.0 и Spark 2.3.0.

Обновление версии сгенерированного кода в будущем может повлечь за собой необходимость обновления соответствующего модуля генерации кода в случае изменения некоторых API.

4.7.5. Применение инструмнта разработки с использованием диаграмм активности

Как упоминалось ранее, диаграммы действий могут использоваться в UML для представления потока данных между действиями в Activities. Это достигается с помощью ObjectFlows, т.е. ребер, представляющих передачу объектов от исходного действия к целевому.

Мы поэкспериментировали с диаграммами действий и разработали альтернативную реализацию StreamUML под названием StreamUMLAD, которую затем использовали для моделирования приложения StreamingWordCount, как показано на рис. 4.8. Обратите внимание, что в данном случае мы имеем в виду трансформацию, а не Трансформатора, чтобы привести терминологию в соответствие с семантикой действий. На рис. 4.8 преобразования, требуемые в примере, представлены как действия, должным образом стереотипизированные с использованием той же терминологии, что и в разделе 4.4.3. Источник данных представлен как UML AcceptEventAction, стереотипизированный как «SocketSource». AcceptEventAction определяется непосредственно в UML и представляет собой определенный тип действия, которое ожидает наступления некоторых событий. Альтернативным способом моделирования источников данных было бы определение параметров потоковой передачи для Activity, а затем создание шаблонных ActivityParameterNodes (по одному для каждого источника данных). ActivityParameterNodes позволяют моделировать тот факт, что обновления значений для данного параметра (который в нашем контексте представляет собой входные данные) могут быть получены во время выполнения соответствующего действия. Приемник данных представлен в виде UML DataStoreNode, определенного типа ObjectNode, который постоянно содержит объекты. Потоки представлены потоками объектов, связанными с исходными и целевыми действиями через так называемые Pins (определенный тип ObjectNode).

Эти последние элементы предоставляют свойство type, которое определяет тип данных, которые могут передаваться по потоку. Согласно спецификации UML, свойство type для нижестоящего вывода должно быть таким же или быть супертипом для вышестоящего вывода (в нашем случае мы предполагаем, что эти два типа должны быть одинаковыми). Определение двух типов данных, специфичных для конкретных примеров, WordToken и WordCount, и, в общем, любого другого сложного типа данных, который потребуется определить пользователю, требует создания соответствующих классов или типов данных на диаграмме классов (не показано это здесь из соображений экономии места).

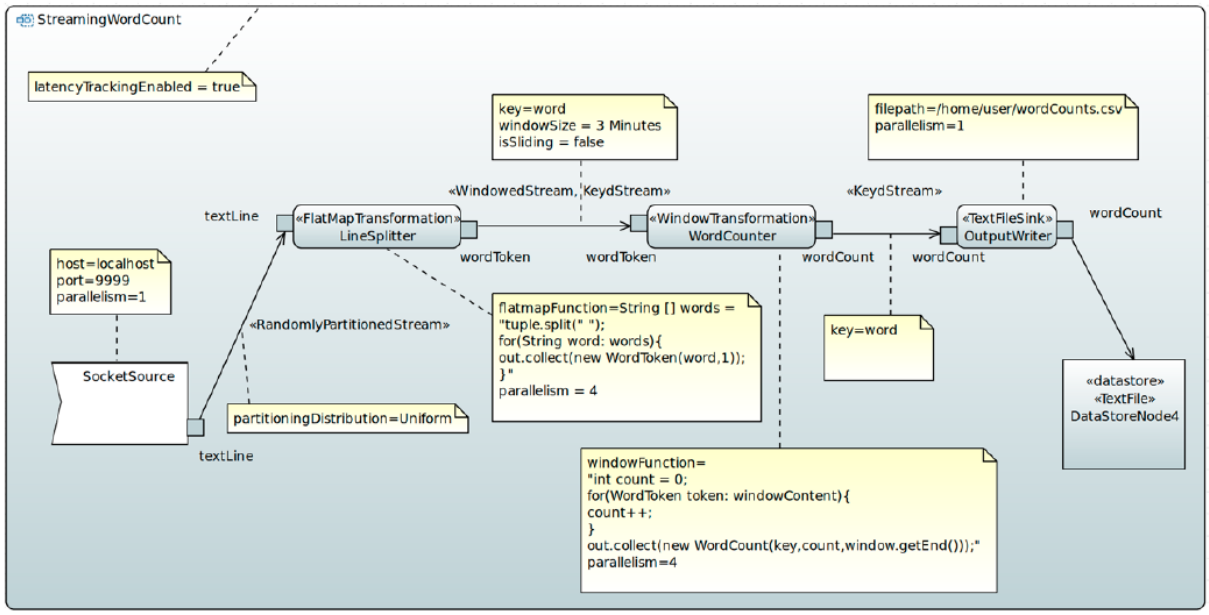


Рис. 4.8. Моделирование приложения StreamingWordCount с использованием $\text{StreamUML}_{\text{AD}}$

Кроме того, в случае диаграмм действий распараллеливание операторов, а также семантика управления окнами и секционирования, связанная с потоками, были определены с помощью новых подтипов и свойств, поскольку мы не могли использовать специфичные для UML возможности для этих целей.

Что касается параллелизма, UML действительно предоставляет понятие ExpansionRegion, которое можно рассматривать как часть Activity, включающую некоторые действия и ребра (включая ObjectFlows). Область расширения берет некоторые входные данные из коллекции и выполняет свои действия для каждого из этих входных данных. Выполнение действий может происходить в трех различных режимах: параллельном, что означает, что выполнение выполняется параллельно для каждого входного элемента, итеративном, что означает, что когда выполнение в области расширения завершено, может начаться новое, и потоковом, что препятствует параллелизму [4.20].

К сожалению, ни один из вышеперечисленных режимов не подходит для нашего случая, поскольку в потоковых системах есть возможность напрямую управлять количеством параллельных экземпляров определенного компонента способом, независимым от количества входных данных. Например, мы могли бы захотеть установить три или десять параллельных экземпляров определенного компонента, работающих со 100 входящими кортежами, которые затем распределяются между такими компонентами в соответствии с некоторым критерием разделения, в то время как параллельный режим, применяемый к ExpansionRegion, позволил бы нам иметь только 100 компонентов, работающих со 100 входящими кортежами. По этой причине в StreamUMLAD мы не могли полагаться на parallelmode, определенный в UML, и нам пришлось явно ввести новое свойство параллелизма в наши операторы преобразования. Аналогичный выбор был сделан в [4.21, 4.22, 4.24, 4.25].

Возможным кандидатом для моделирования оконных потоков является UML-элемент CentralBufferNode, который представляет собой особый тип ObjectNode, действующий как буфер между потоками объектов и передающий входящие токены объектов (кортежи в нашей терминологии) исходящим потокам в соответствии с некоторым общим

правилом упорядочения. Однако CentralBufferNode сам по себе не позволяет группировать буферизованные элементы (или токены объектов) и действует как простой механизм для разделения потока объектов между действиями.

Способ группирования элементов обеспечивается возможностью определения веса для ObjectFlow. Это определяет количество токенов, которые “должны пересекать границу одновременно” [4.20]. Так, например, мы могли бы сказать, что кортежи, буферизованные в CentralBufferNode, могут быть переданы преобразованию, как только они достигнут определенного числа. Это позволяет нам моделировать неперекрывающееся CountWindow (т.е., в терминах метамодели из раздела 4.4.2, окно с размером, равным размеру слайда), но этого недостаточно, чтобы охватить другие механизмы управления окнами. Более того, мы не смогли определить какой-либо способ моделирования семантики разделения данных, то есть возможности явно указать критерий, который будет использоваться для распределения входных токенов между набором параллельных экземпляров одного и того же оператора преобразования. По этим причинам нам в конечном итоге пришлось ввести два подстереотипа «потока данных», а именно «WindowedDataStream» и «PartitionedStream», также в $\text{StreamUML}_{\text{AD}}$.

В результате этого анализа мы пришли к выводу, что диаграммы действий в сочетании с диаграммами классов для определения типов данных, безусловно, являются подходящей альтернативой для создания StreamUML. Однако они не дают существенных преимуществ по сравнению с комбинацией диаграмм классов и составных структурных диаграмм.

Во-первых, с точки зрения элементов профиля $\text{StreamUML}_{\text{AD}}$ существенно не отличается от других.

Почти все элементы профиля, определенные в StreamUML, пришлось переопределить в $StreamUML_{AD}$, разумеется, расширив различные мета-классы. Единственным исключением является возможность повторного использования в $StreamUML_{AD}$ AcceptEventActions или параметров в качестве источников данных и DataStoreNode в качестве приемника. Однако в обоих случаях наиболее сложные аспекты, связанные с управлением окнами, секционированием и параллелизмом, должны быть определены с нуля. Кроме того, составные структурные диаграммы предлагают более эффективную функцию множественности для выражения внутренней характеристики параллелизма в потоковых приложениях.

Во-вторых, поскольку диаграммы активности богаче и выразительнее, чем информационные потоки, они предлагают несколько возможностей моделирования, которые не могут быть использованы для моделирования потоковых приложений с помощью $StreamUML_{AD}$. Это означает, что пользователи StreamGen должны быть более осведомлены о UML, чем в случае использования более простой альтернативы, основанной на информационных потоках. Например, для моделирования потока данных между двумя операторами пользователю потребуются два действия, два вывода и объектный поток между ними; свойство type выходного вывода исходного действия должно совпадать со свойством входного вывода целевого действия, т.е. оба вывода должны быть настроены одинаково. Кроме того, для моделирования сложных типов данных пользователю потребуется определить некоторые типы данных, создающие вторую диаграмму, например диаграмму классов. Таким образом, пользователь должен будет знать разницу между диаграммами действий и диаграммами классов и управлять двумя разными диаграммами.

4.7.6. Результаты экспериментов

Учитывая результаты наших экспериментов, мы приходим к выводу, что StreamGen в достаточной степени соответствует предъявляемым требованиям, даже если, как мы подчеркивали в предыдущих разделах, есть возможности для дальнейших исследований и доработки текущего прототипа. Во-первых, мы смогли использовать его для моделирования и экспериментов с относительно сложным приложением, которое использует множество функций потоковой передачи, состоит из 11 независимых операторов потоковой передачи и опирается на 2 внешние системы (а именно, Kafka и Cassandra). Во-вторых, мы показали, что StreamGen генерирует код Flink и Spark для приложения Yahoo Benchmark, что приводит к незначительному снижению производительности. В-третьих, мы сравнили лингвистические конструкции StreamGen с моделью данных Google и пришли к выводу, что она способна охватить большинство функций, определенных в этой модели. В-четвертых, наше предварительное эмпирическое исследование показывает, что около половины наших пользователей смогли создать прототип потокового приложения в среднем менее чем за час. Этот предварительный результат можно считать значительным как с точки зрения повышения производительности, так и с точки зрения сглаживания кривой обучения, даже если он нуждается в подтверждении другими исследованиями с участием экспертов по потоковой передаче и реальными промышленными сценариями. Наконец, мы поэкспериментировали с расширяемостью StreamGen и показали, что это относительно простая задача для опытного разработчика.

4.8. Выводы

Растущий спрос на распределенные потоковые приложения создает потребность в соответствующих абстракциях и связанных языках и

инструментах, позволяющих справиться с их сложностью и неоднородностью.

В рамках этого направления мы представляем StreamGen - инструмент разработки, основанный на моделях, состоящий из нового профиля UML, который позволяет определять модель приложения, используя знакомую нотацию UML, и двух генераторов кода для Apache Spark и Flink. Оценив StreamGen, мы показали, что его можно использовать для проектирования и разработки реальных приложений (например, Yahoo benchmark) и что, несмотря на некоторые текущие ограничения, он обеспечивает поддержку многих важных функций потоковой передачи.

В будущем планируется продолжить разработку StreamGen, добавив новые генераторы кода для других популярных платформ потокового вещания и обеспечив поддержку отсутствующих в настоящее время функций потокового вещания, таких как триггеры и методы обработки результатов.

Подводя итог, StreamGen демонстрирует следующие уникальные аспекты: будучи основан на высокоуровневом языке UML, он направлен на упрощение процесса обучения разработке потокового приложения; он позволяет разработчикам определять как структуру потокового приложения, так и логику взаимодействия между потоковыми приложениями, его компонентами и логикой приложения; наконец, он предлагает механизм генерации кода, который позволяет легко внедрять расширяемый набор распределенных потоковых фреймворков с небольшими накладными расходами, которые, как было показано в рассмотренных случаях.

ЗАКЛЮЧЕНИЕ

В процессе выполнения диссертационного исследования получены следующие основные результаты:

1. Предложена графовая модель синтеза мегасети из набора непересекающихся подграфов с согласованностью в виде нормы H_2, учитавающая непрерывность весов ребер на всей числовой оси и зашумленную динамику согласования узлов, реализующая соединение подграфов с обеспечением оптимальной согласованности финальной мегасети.

2. Предложена оптимизационная задача выбора мостов подграфов мегасети с построением соединительных ребер, использующая параллелизмом при решении и обеспечивающая получение оценок минимальной и максимальной согласованности с весами ребер на положительной полуоси.

3. Разработан алгоритм кластеризации данных групп сенсорных узлов мегасети на основе использования нечеткой логики для учета гетерогенных параметров сенсорной сети и гетерогенного управления сетью, обеспечивающий инвариантность к большому объему сетевых данных и динамичности сетевого местоположения узлов. Трудоемкость разработанного алгоритма составляет O(N), точность кластеризации по сравнению с известными прототипами лучше на 4%.

4. Предложена архитектура системы разработки потоковых приложений для обработки данных на основе моделей потоков данных с включением логики приложения в модель, и обеспечивающая интеграцию в среду IDE поверх UML.

5. Создана структура программного прототипа системы разработки потоковых приложений для обработки данных на основе моделей потоков данных. Элементы программного обеспечения зарегистрированы в ФИПС.

Рекомендации и перспективы дальнейшей разработки темы

1. Результаты исследования рекомендуются к применению в задачах проектирования вычислительных систем с разреженной архитектурой на основе потоковых приложений.

2. Дальнейшая разработка темы будет направлена на практическую реализацию теоретических и алгоритмических результатов, интеграцию в проекты наиболее распространенных распределенных систем. Развитие результатов будет направлено на автоматизацию разработки потоковых приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Абдихалык Ш.С. Методы и средства моделирования атак в больших компьютерных сетях // Студенческий вестник. 2023. № 16-11 (255). С. 51-57.

2. Абдулкадыров У.У., Джабраилов И.А., Амерханова З.Ш. Технологии информационной безопасности компьютерных сетей и тенденции их развития // Журнал прикладных исследований. 2023. № 6. С. 25-29.

3. Абрамов А.Г. Концептуальный взгляд на архитектуру сервисной платформы национальной исследовательской компьютерной сети России// Информационные технологии. 2022. Т. 28. № 2. С. 68-74.

4. Абрамов А.Г., Евсеев А.В., Гончар А.А., Шабанов Б.М. Вопросы увеличения пропускной способности и территориальной доступности национальной исследовательской компьютерной сети России // Системы и средства информатики. 2022. Т. 32. № 2. С. 4-12.

5. Абуков Ш.З., Евдокимова К.В., Евдокимов В.О., Куракова Н.Ю. Анализ и оценка эффективности интегрированного сетевого трафика телекоммуникационной и компьютерной сети // Экономика и предпринимательство. 2022. № 11 (148). С. 1371-1374.

6. Андриянов А.М. Компьютерные сети и сетевые технологии. Тюмень, 2023.

7. Анисимов В.Г., Анисимов Е.Г., Сауренко Т.Н., Лось В.П. Оценка эффективности систем защиты компьютерных сетей от вирусных атак // Проблемы информационной безопасности. Компьютерные системы. 2022. № 1. С. 11-17.

8. Атаева О. Развитие компьютерных сетей и их роль в экономике // Вестник науки. 2023. Т. 1. № 4 (61). С. 274-277.

9. Бабенко М.С. Исследование методов и средств анализа защищённости компьютерных сетей // Интернаука. 2022. № 47-1 (270). С. 16-19.

10.Башкатова А.Н., Логинова Л.Н. Свёрточные нейронные сети как метод для реализации компьютерного зрения // Соискатель - приложение к журналу "Мир транспорта". 2022. № 1 (11). С. 6-9.

11.Белов А.С., Добрышин М.М., Струев А.А., Горшков А.А. Модель компьютерной сети, функционирующая в условиях деструктивных программных воздействий и учитывающая требуемый уровень восстанавливаемости // Известия Тульского государственного университета. Технические науки. 2022. № 2. С. 83-89.
Бруй И.Ю. Кибербезопасность компьютерных сетей военного назначения// Новые информационные технологии в телекоммуникациях и почтовой связи. 2023. Т. 1. № 1. С. 252-253.

12.Булынин А.Г., Мельников Б.Ф., Мещанин В.Ю. и Терентьева Ю.Ю. «Алгоритмы проектирования коммуникационных сетей с использованием жадных эвристик разных типов» // Proceedings of the International Conference on Information Technology and Nanotechnology (ITNT-2020) - Самара: Самарский университет, 2020. - С. 856–860.

13.Валиахметова Е.Д., Макуха М.Ю. Построение функциональной и логико-вероятностной модели обнаружения инсайдера на основании уязвимостей компьютерной сети // Электронный сетевой политематический журнал "Научные труды КубГТУ". 2022. № 3. С. 47-54.

14.Винников А.М., Дедова М.А., Кочетова Н.П., Фролов А.Б. Компьютерная модель защищенной беспроводной сенсорной сети "умного дома" // Математические методы в технологиях и технике. 2023. № 5. С. 111-115.

15.Виноградов Г.П., Фомина Е.Е., Кошкина Г.В. Компьютерные сети. Работа в сети интернет. Тверь, 2022.

16.Вишневецкая А.А., Аверченко А.П. Проектирование компьютерной сети // Молодой ученый. 2022. № 20 (415). С. 123-125.

17.Воротницкий Ю.И., Румас Р.А. Архитектура аппаратно-программного средства однонаправленной передачи данных в компьютерных сетях // Доклады Белорусского государственного университета информатики и радиоэлектроники. 2023. Т. 21. № 3. С. 96-101.

18. Гомазкова Л.К., Серёженко И.Д., Трофимов А.И. Оценка точности методов краткосрочного прогнозирования нагрузки компьютерной сети // НБИ технологии. 2022. Т. 16. № 2. С. 11-16.

19.Гребенкина А.Ю. Суть и значение локальных и глобальных компьютерных сетей // Научные исследования XXI века. 2023. № 2 (22). С. 27-29.

20.Григорьев А.О. Проектирование компьютерной сети // Интернаука. 2022. № 46-1 (269). С. 28-29.

21.Дулесов А.С., Федоренко Н.С., Байшев А.В. Когнитивное моделирование в задаче повышения надёжности компьютерных сетей // Вестник Хакасского государственного университета им. Н.Ф. Катанова. 2022. № 2 (40). С. 104-108.

22.Дятлов П.А. Принципы построения и организация компьютерных сетей. Ростов-на-Дону ; Таганрог, 2022.

23.Епифанов Е.К. Различные методы и подходы к проектированию компьютерных сетей // Научно-исследовательский Центр "Science Discovery". 2023. № 12. С. 73-78.

24.Загидуллина Э.Г. Возможные решения проблем в контексте использования современных компьютерных сетей // Научно Исследовательский Центр "Science Discovery". 2023. № 13. С. 101-105.

25.Зиганурова Р.А. Инструментальное средство проектирования компьютерных сетей // Научно-исследовательский центр "Вектор

развития". 2022. № 7. С. 160-162.

26.Золкин А.Л., Мунистер В.Д. Проектирование цифровых экосистем окружающего интеллекта, сенсорных и компьютерных сетей. Москва, 2022.

27.Камиль В.А.К. Некомбинаторное решение задачи оптимизации согласованности для построения сообщества сетей// Информационные технологии моделирования и управления, №4(134), 2023. – С. 290-302

28.Камиль В.А.К., Мутин Д.И. Методы кластеризации сенсорных узлов на основе нечетких множеств// Информационные технологии моделирования и управления, №1(135), 2024. – С. 37-46.

29.Камиль В.А.К., Мутин Д.И., Атласов И.В. Теоретические основы управления согласованностью подсистем при обработке данных в объединении компьютерных сетей// Системы управления и информационные технологии, №4(94), 2023. С. 35-40

30.Камиль В.А.К., Мутин Д.И., Голиков А.А. Алгоритмизация определения количества обслуживаемых кластеров и количества сенсорных узлов, сопоставленных с каждым кластером// Экономика и менеджмент систем управления, №2(52), 2024. – С. 55-65.

31.Ким С.А., Бондарь К.М., Шевцов А.Н. Методы анализа и обеспечения защищённости компьютерных сетей государственной финансовой организации // Научно-техническое и экономическое сотрудничество стран АТР в XXI веке. 2023. Т. 1. С. 198-203.

32.Клычева Д.М., Юсупова С.Т. Архитектура компьютерных сетей // Вестник науки. 2022. Т. 2. № 12 (57). С. 308-311.

33.Лисьев Г.А., Романов П.Ю., Аскерко Ю.И. Программное обеспечение компьютерных сетей и WEB-серверов. Москва, 2023.

34.Львович Я.Е., Преображенский Ю.П., Ружицкий Е. Анализ особенностей приема и передачи сигналов в компьютерных сетях // Вестник Воронежского института высоких технологий. 2022. № 1 (40). С.

75-78.

35.Мельникова Е.М. Проблемы создания компьютерных сетей внутрипроизводственных предприятий // Научно-исследовательский центр "Technical Innovations". 2023. № 13. С. 152-156.

36.Мильнер Б.З. Теория организации. Учебник. М.: Инфа-М. 2006.

37.Мостовой Я.А., Бердников В.А. «Стратегия безопасности больших сетей» // Сборник трудов III международной конференции и молодёжной школы «Информационные технологии и нанотехнологии» (ИТНТ-2017) - Самара: Новая техника, 2017. - С. 896–903.

38.Никитенко И.Д. Локальные и глобальные компьютерные сети // Современные информационно-коммуникационные технологии. 2022. № 12. С. 37-40.

39.Олескин А.В. Междисциплинарные сетевые группы// Вестн. Росс. Акад. наук. 1998. № 11. С. 1016–1022.

40.Олескин А.В., Курдюмов В.С. Децентрализованные сетевые структуры в научном сообществе, системе образования, гражданском обществе и бизнесе: модель хирамы // Экономические стратегии. 2018. № 2. С. 2–18.

41.Олескин А.В. Роль децентрализованных кооперативных сетей (ДКС) в восстановительной медицине // Вестник восстановительной медицины. 2018. № 2. С. 21–28.

42.Олескин А.В. Сетевое общество: его необходимость и возможные стратегии построения. М.: УРСС. 2016. 194 с.

43.Олескин А.В. Сетевые структуры в биосистемах и человеческом обществе. М.: УРСС. 2012. 301 с.

44.Олескин А.В. Сети как неиерархические и нерыночные структуры: реализация в биологических и социальных системах // Экономические стратегии. 2013. № 5. С. 2-7.

45.Пашаев М.Т. Локальная компьютерная сеть предприятия //

Научно-технические ведомости Севмашвтуза. 2022. № 1. С. 12-17.

46.Платонов Д.Е. Оценка производительности технологической сети при обработке однопродуктового потока с использованием компьютерного моделирования // Вестник Санкт-Петербургского государственного университета технологии и дизайна. Серия 4: Промышленные технологии. 2023. № 1. С. 46-50.

47.Ретюнских С.Н. Способ оценки структурной надёжности вычислительных компьютерных сетей // Современные наукоемкие технологии. 2022. № 2. С. 86-91.

48.Самохвалов А.В., Соловьев Д.С., Соловьева И.А., Скворцов А.А. Обеспечение избыточности для повышения надежности функционирования корпоративной компьютерной сети передачи информации // Прикаспийский журнал: управление и высокие технологии. 2022. № 4 (60). С. 68-76.

49.Смородинская Н.В. Глобализированная экономика: от иерархий к сетевому укладу. М.: Институт экономики РАН. 2015.

50.Степанов В.Ю. Направления развития беспроводных компьютерных сетей // Научный альманах Центрального Черноземья. 2022. № 1-10. С. 116-119.

51.Тищенко В.И., Жукова Т.И., Попов Ю.С. Сетевые взаимодействия. Предмет исследования и объект моделирования. М.: УРСС, 2014. 352 с.

52.Харитонов А., Джаманкулов А. Компьютерные сети и протоколы передачи данных. Санкт-Петербург, 2023.

53.A new notion of effective resistance for directed graphs - part i: Definition and properties/ G.F. Young, L. Scardovi, N.E. Leonard// IEEE Trans. Autom. Contr., vol. 61, no. 7, pp. 1727-1736, 2015.

54.Adeel, A., Abid, A., Sohail, J. (2010). Energy aware intra cluster routing for wireless sensor networks// International Journal of Hybrid

Information Technology, Vol. 3, No. 1, pp.1–6.

55.Affetti L. et al. Defining the execution semantics of stream processing engines// J. Big Data, 20174, 1, 12. https://doi.org/10.1186/s40537-017-0072-9.

56.Akidau T. et al. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proc. VLDB Endow, 2015. 8, 12, pp. 1792–1803.

57.Algorithms for leader selection in stochastically forced consensus networks/ F. Lin, M. Fardad, M.R. Jovanovic//́ IEEE Trans. Automat. Contr., vol. 59, no. 7, pp. 1789-1802, 2014.

58.Almeida J.P.A. Model-driven Design of Distributed Applications. Ph.D. Dissertation. - University of Twente, Netherlands, 2006.

59.Altafini C. Consensus problems on networks with antagonistic interactions// IEEE Trans. Automat. Contr., vol. 58, no. 4, pp. 935-946, Apr 2013.

60.Ardagna C.A. et al.A model-driven methodology for big data analytics-as-a-service// Proc. of the 2017 IEEE Int. Congress on Big Data (BigData. 2017. Pp. 105–112. https://doi.org/10.1109/BigDataCongress.2017.23.

61.Ardagna D. et al. MODAClouds: A model-driven approach for the design and execution of applications on multiple clouds// Proc. of the 4th International Workshop on Modeling in Software Engineering. - IEEE Press, 2012, pp. 50–56. http://dl.acm.org/citation.cfm?id=2664431.2664439.

62.Balasubramanian K. et al. Developing applications using model-driven design environments// Computer 2006. 39, 2, pp. 33–40. DOI: 10.1109/MC.2006.54.

63.Bezdek, C., Gunderson, C.R., Watson, J. (1981). Detection and characterization of cluster substructure – linear structure, fuzzy c-varieties and convex combinations thereof// SIAM J. Appl. Math., Vol. 40, No. 2, pp.358–372.

64.Brambilla M., Cabot J., Wimmer M. Model-Driven Software

Engineering in Practice. - Morgan & Claypool, 2012. 198 p.

65.Buttyán, L., Schaffer, P. (2010). PANEL: position-based aggregator node election in wireless sensor networks// International Journal of Distributed Sensor Networks, Vol. 2010, Article ID 679205, pp.1–15.

66.Casale G. et al. DICE: Quality-driven development of data intensive cloud applications// Proc. of the 2015 IEEE/ACM 7th Int. Workshop on Modeling in Software Engineering. 2015. pp. 78–83. DOI:10.1109/MiSE.2015.21.

67.Catastrophic cascade of failures in interdependent networks/ S.V. Buldyrev, R. Parshani, G. Paul, H.E. Stanley, S. Havlin// Nature, vol. 464, no. 7291, pp. 1025-1028, 2010.

68.Chapman A., Mesbahi M. Multiple time-scales in network-of-networks// in Proc. American Control Conf., 2016, pp. 5563-5568.

69.Chau, M., Cheng, R., Kao, B., Ng, J. (2006). Uncertain data mining: an example in clustering location data// Proceedings of the 10th Pacific-Asia Conference on Knowledge Discover and Data Mining (PAKDD 2006), Lecture Notes in Computer Science, Vol. 3918, pp.199–204.

70.Chou, C.C. (2009). Integrated short-term and long-term MCDM model for solving location selection problems// Journal of Transportation Engineering, Vol. 135, No. 11, pp.880–893.

71.Coherence in large-scale networks: Dimension-dependent limitations of local feedback/ B. Bamieh, M.R. Jovanovic, P. Mitra, S. Patterson// IEEE Trans. Autom. Contr., vol. 57, no. 9, pp. 2235-2249, 2012.

72.Colas M. et alCracking the Data Conundrum: How Successful CompaniesMake Big Data Operational. Technical Report. - Capgemini consulting, 2015. https://www.capgemini-consulting.com/cracking-the-data-conundrum.

73.Consensus and cooperation in networked multi-agent systems/ R. Olfati-Saber, J.A. Fax, R.M. Murray// Proc. IEEE, vol. 95, no. 1, pp. 215-233,

2007.

74.Control and communication challenges in networked real-time systems/ J. Baillieul, P.J. Antsaklis// Proc. IEEE, vol. 95, no. 1, pp. 9-28, 2007.

75.Controllability and observability of network-of-networks via cartesian products/ A. Chapman, M. Nabi-Abdolyousefi, M. Mesbahi// IEEE Trans. Automat. Contr., vol. 59, no. 10, pp. 2668-2679, Oct 2014.

76.Convergence rate of consensus in a network of networks/ A. Das, Y. Yi, S. Patterson, B. Bamieh, Z. Zhang// Proc. IEEE Conf. Decision and Control, 2018.

77.Coordinated vehicle platoon control: Weighted and constrained consensus and communication network topologies/ L.Y. Wang, A. Syed, G. Yin, A. Pandya, H. Zhang// Proc. IEEE Conf. Decision and Control, 2012, pp. 4057-4062.

78.Da Silva Marcos A.A. et al. JUNIPER: Towards modeling approach enabling efficient platform for heterogeneous big data analysis// Proc. of the 10th Central and Eastern European Software Engineering Conf. in Russia. - ACM, 2014. Article 12, 7 p. DOI: 10.1145/2687233.2687252.

79.Damiani E. et al. Toward model-based big data-as-a-service: The TOREADOR approach// In Advances in Databases and Information Systems. - Springer International Publishing, Cham, 2017. pp. 3–9.

80.Effective graph resistance/ W. Ellens, F. Spieksma, P. Van Mieghem, A. Jamakovic, R. Kooij// Linear algebra and its applications, vol. 435, no. 10, pp. 2491-2506, 2011.

81.Ester, M., Kriegel, H., Sander, J., Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise// Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining (KDD’96), pp.226–231.

82.Fardad M. On the optimality of sparse long-range links in circulant consensus networks// Proc. Am. Control Conf., pp. 2075-2080, 2015.

83.Feldt R., Magazinius A. Validity threats in empirical software

engineering research - an initial survey// Proc. of the Software Engineering and Knowledge Engineering Conf. 2010. pp. 374–379.

84.Fitch K., Leonard N.E. Information centrality and optimal leader selection in noisy networks// Proc. IEEE Conf. Decision and Control, pp. 7510-7515, 2013.

85.Fitch K., Leonard N.E. Joint centrality distinguishes optimal leaders in noisy networks// IEEE Trans. Contr. Netw. Syst., vol. 3, no. 4, pp. 366-378, 2016.

86.Fred, L.N., Leitao, M.N. (2003). A new cluster isolation criteria based on dissimilarity increments// IEEE Trans. Pattern Anal. Machine Intelligence, Vol. 25, No. 8, pp.944–958.

87.Gomez A. et al. Towards a UML profile for data intensive applications// Proc. of the 2nd Int. Workshop on Quality-Aware DevOps. - ACM, 2016. pp. 18–23. DOI: 10.1145/2945408.2945412.

88.Greene, D., Cunningham, P. (2006) Efficient Ensemble Methods for Document Clustering, Technical Report, Department of Computer Science, Trinity College Dublin, pp.1–6.

89.Guerriero M., Nesta A., Di Nitto E. Streamgen: A UML-based tool for developing streaming applications// Proc. of the 10th Int. Workshop on Modelling in Software Engineering. 2018. pp/ 57–58. DOI: 10.1145/3193954.3193964.

90.Guhe, S., Rastogi, R., Shim, K. (2000). ROCK: a robust clustering algorithm for categorical attributes// Inf. Syst., Vol. 25, No. 5, pp.345–366.

91.Guo, L-Q., Xie, Y., Yang, C-H., Jing, Z-W. (2010). Improvement on LEACH by combining adaptive cluster head election and two-hop transmission// Proceedings of the Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science – 2010.

92.Hassan-Moghaddam S., Jovanovic M.R. Topology design for stochastically forced consensus networks// IEEE Trans. Contr. Netw. Syst., vol.

5, no. 3, pp. 1075-1086, 2018.

93.He, Z., Xu, X., Deng, S. (2008) Clustering Mixed Numeric and Categorical Data: A Cluster Ensemble Approach, pp.1–6, ARXiv Computer Science e-prints.

94.Heinzelman, W.B., Chandrakasanand, A.P., Balakrishnan, H. (2002). An application specific protocol architecture for wireless micro sensor networks// IEEE Trans. Wireless Communication, Vol. 1, No. 4, pp.660–670.

95.https://icseg.iti.illinois.edu/power-cases/.

96.https://www.omg.org/spec/UML/2.5.1.

97.Huruiala, P.C., Urzica, A., Gheorghe, L. (2010). Hierarchical routing protocol based on evolutionary algorithms for wireless sensor networks// 9th RoEduNet IEEE International Conference, pp.387–392.

98.Jung, S., Han, Y., Chung, T. (2007). The concentric clustering scheme for efficient energy consumption in the PEGASIS// Proceedings of the 9th International Conference on Advanced Communication Technology, pp.260–265.

99.Kamil W.A.K., Mutin D.I. Analytical examples of optimal network design in network integration based on the NoN-model// Modern informatization problems in the technological and telecommunication systems analysis and synthesis (MIP-2024'AS): Proceedings of the XXIX-th International Open Science Conference. - Yelm, WA, USA: Science Book Publishing House, 2024. Pp. 153-158.

100. Kaneko R., Miyaguchi K., Yamanishi Ke. Detecting changes in streaming data with information theoretic windowing// Proc. of the 2017 IEEE Int. Conf. on Big Data. - IEEE Computer Society, 2017. pp. 646–655. DOI: 10.1109/BigData.2017.8257980.

101. Kaufman, L., Rousseeuw, P. (1990) Finding Groups in Data: An Introduction to Cluster Analysis, pp.223–226, Wiley, Applied Probability and Statistics Series, New York, NY.

102. Kent S. Model driven engineering// Integrated Formal Methods. - Springer, 2002. pp. 286–298.

103. Khaji, N., Mehrjoo, M. (2014). Crack detection in a beam with an arbitrary number of transverse cracks using genetic algorithms// Journal of Mechanical Science and Technology, Vol. 8, No. 3, pp.823–836.

104. Khalil, I.M., Gadallah, Y., Hayajneh, M., Khreishah, A. (2012). An adaptive OFDMA-based MAC protocol for underwater acoustic wireless sensor networks// Sensors, Vol. 7, pp.8782–8805, MDPI, Basel, Switzerland.

105. Kim, J-M., Park, S-H., Han, Y-J., Chung, T-M. (2008). CHEF: cluster head election mechanism using fuzzy logic in wireless sensor networks// Proc. 10th Int. Conf. Advanced Communication Technology ICACT, Vol. 1, pp.654–659.

106. Kleinand D., Randic M. Resistance distance// J. Math. Chem., vol.12, no. 1, pp. 81-95, 1993.

107. Labunets K. Model comprehension for security risk assessment: An empirical comparison of tabular vs. graphical representations// Proc. of the 40th Int. Conf. on Software Engineering. - ACM, 2018. pp. 395–395. DOI: 10.1145/3180155.3182511.

108. Lagarde F. Improving UML profile design practices by leveraging conceptual domain models// Proc. of the 22nd IEEE/ACM Int. Conf. on Automated Software Engineering. - ACM, 2007. pp. 445–448.

109. Lao, Y., Wu, Y., Wang, Y., McAllister, K. (2012). Fuzzy logic-based mapping algorithm for improving animal-vehicle collision data// Journal of Transportation Engineering, Vol. 138, No. 5, pp.520–526.

110. Lau, H.C.W., Jiang, Z.Z., Ip, W.H., Wang, D.W. (2010). A credibility-based fuzzy location model with Hurwicz criteria for the design of distribution systems in B2C e-commerce// Computers and Industrial Engineering, Vol. 59, No. 4, pp.873–886.

111. Li, J., Liao, G., Wang, F., Li, J. (2013). Maximum lifetime routing

based on fuzzy set theory in wireless sensor networks// JSW, Vol. 8, No. 9, pp.2321–2328.

112. Lindsey, S., Raghavendra, C.S. (2002). PEGASIS: power-efficient gathering in sensor information systems// Proceedings of the IEEE Aerospace Conference, Vol. 3, pp.125–1130.

113. Liu X., Iftikhar N., Xie X. Survey of real-time processing systems for big data// Proc. of the 18th Int. Database Engineering and Applications Symp. - ACM, 2014. pp. 356–361. https://doi.org/10.1145/2628194.2628251.

114. Liu, P.D., Jin, F. (2012). A multi-attribute group decision-making method based on weighted geometric aggregation operators of interval-valued trapezoidal fuzzy numbers// Applied Mathematical Modelling, Vol. 38, No. 1, pp.2498–2509.

115. Liu, X.D. (1998a). The fuzzy sets and systems based on AFS structure, EI algebra and EII algebra// Fuzzy Sets and Systems, Vol. 95, No. 2, pp.179–188.

116. Liu, X.D. (1998b). The fuzzy theory based on AFS algebras and AFS structure// Journal of Mathematical Analysis and Applications, Vol. 217, No. 2, pp.459–478.

117. Mackin E., Patterson S. Optimizing the coherence of composite networks// Proc. Am. Control Conf., pp. 4334-4340, 2017.

118. MacQueen, J.B. (1967). Some methods for classification and analysis of multivariate observations// Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, pp.281–297.

119. Marconi F., Bersani M.M., Rossi M. A model-driven approach for the formal verification of storm-based streaming applications// SIGAPP Appl. Comput. Rev. 2017. 17, 3, pp. 6–15. DOI: 10.1145/3161534.3161535.

120. Masseglia, F., Cathala, F., Poncelet, P. (1998). The PSP approach for mining sequential patterns// Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery, pp.176–

184.

121. Minimizing effective resistance of a graph/ A. Ghosh, S. Boyd, A. Saberi// SIAM Rev., vol. 50, no. 1, pp. 37-66, 2008.

122. Modeling interdependent infrastructures using interacting dynamical models/ V. Rosato, L. Issacharoff, F. Tiriticco, S. Meloni, S. Porcellinis, R. Setola// Int. J. Crit. Infrastruct., vol. 4, no. 1-2, pp. 63-79, 2008.

123. Nam, C.S., Han, Y.S., Shin, D.R. (2011). Multi-hop routing-based optimization of the number of cluster-heads in wireless sensor networks [J]// Sensors, Vol. 11, No. 3, pp.2875–2884.

124. Network composition for optimal disturbance rejection/ R. Santini, A. Gasparri, F. Pasqualetti, S. Panzieri// Proc. Am. Control Conf., 2016.

125. Noise-induced limitations to the scalability of distributed integral control/ E.Teglingand, H.Sandberg// Syst. Control Lett., vol. 130, pp. 23-31, 2019.

126. On the definiteness of graph laplacians with negative weights: Geometrical and passivity-based approaches/ Y. Chen, S.Z. Khong, T.T. Georgiou// Proc. Am. Control Conf., 2016, pp. 2488-2493.

127. Ong, K-L., Li, W., Ng, W-K., Lim, E-P. (2004). SCLOPE: an algorithm for clustering data streams of categorical attributes// Lecture Notes in Computer Science, Vol. 3181, pp.209–218.

128. Peixoto T.P., Bornholdt S. Evolution of robust network topologies: Emergence of central backbones// Phys. Rev. Lett., vol. 109, no. 11, p. 118703, 2012.

129. Perez-Palacin D. A UML profile for the design, quality assessment and deployment of data-intensive applications// Softw. Syst. Model. 2019. 18, 6, pp. 3577–3614. DOI: 10.1007/s10270-019-00730-3.

130. Rajbhoj A., Kulkarni V., Bellarykar N. Early experience with model-driven development of MapReduce based big data application// Proc. of the 2014 21st Asia-Pacific Software Engineering Confe. 2014. Vol. 1. pp. 94–

97. DOI: 10.1109/APSEC.2014.23.

131. Requeno J., Merseguer J., Bernardi S. Performance analysis of Apache Storm applications using stochastic Petri nets// Proc. of the 2017 IEEE Int. Conf. on Information Reuse and Integration. 2017. pp. 411–418. DOI: 10.1109/IRI.2017.64.

132. Requeno J.I., Gascon I., Merseguer J. Towards the performance analysis of Apache Tez applications// Companion of the 2018 ACM/SPEC Int. Conf. on Performance Engineering. 2018. pp. 147–152. DOI: 10.1145/3185768.3186284.

133. Robustness of a network of networks/ J. Gao, S.V. Buldyrev, S. Havlin, H.E. Stanley// Phys. Rev. Lett., vol. 107, no. 19, p. 195701, 2011.

134. Robustness of first- and second-order consensus algorithms for a noisy scale-free small-world Koch network/ Y. Yi, Z. Zhang, L. Shan, G. Chen// IEEE Trans. Control Syst. Technol., vol. 25, no. 1, pp. 342-350, Jan 2017.

135. Robustness of noisy consensus dynamics with directed communication/ G.F. Young, L. Scardovi, N. E. Leonard// Proc. Am. Control Conf., pp. 6312-6317, 2010.

136. Runeson P., Host M. Guidelines for conducting and reporting case study research in software engineering// Emp. Softw. Eng. 2009. 14, 2, pp. 131–164.

137. Santurkar S., Arora A., Chandrasekaran K. Stormgen - a domain specific language to create ad-hoc storm topologies// Proc. of the Federated Conf. on Computer Science and Information Systems. 2014. pp. 1621–1628. DOI: 10.15439/2014F278.

138. Selic B. A systematic approach to domain-specific language design using UML// Proc. of the 10th IEEE Int. Symposium on Object-Oriented Real-Time Distributed Computing. -IEEE Computer Society, 2007. pp. 2–9.

139. Sherman J., Morrison W.J. Adjustment of an Inverse Matrix Corresponding to Changes in the Elements of a Given Column or a Given Row

of the Original Matrix (abstract)// Annals of Mathematical Statistics. 20: 621. 1949.

140. Siami M., Motee N. Fundamental limits and tradeoffs on disturbance propagation in linear dynamical networks// IEEE Trans. Autom. Contr., vol. 61, no. 12, pp. 4055-4062, 2016.

141. Siami M., Motee N. Growing linear dynamical networks endowed by spectral systemic performance measures// IEEE Trans. Autom. Contr., vol. 63, no. 7, pp. 2091-2106, 2018.

142. Smaragdakis, G., Matta, I., Bestavros, A. (2004). SEP: a stable election protocol for clustered heterogeneous wireless sensor networks// Proceedings of the 2nd International Workshop on Sensor and Actor Network Protocols and Applications (SANPA '04), pp.251–261.

143. Stephenson K., Zelen M. Rethinking centrality: Methods and examples// Soc. Networks, vol. 11, no. 1, pp. 1-37, 1989.

144. Sun, D., Deng, Y. (2006). Determine discounting coefficient in data fusion based on fuzzy ART neural network// Proc. of the Third International Conference on Advances in Neural Networks, Vol. 1, pp.1286–1292.

145. Tamura, S., Higuchi, S., Tanaka, K. (1973). Pattern classification based on fuzzy relations// IEEE Trans. Syst. Man Cybern., Vol. SMC-3, pp.98–102.

146. Tassa, T., Cohen, D.J. (2013). Anonymization of centralized and distributed social networks by sequential clustering// IEEE Transactions on Knowledge and Data Engineering, Vol. 25, No. 2, pp.2–8.

147. Topology design for optimal network coherence/ T. Summers, I. Shames, J. Lygeros, F. Dorfler// Proc. Euro. Control Conf., pp. 575-580, 2015.

148. Tran, T.N., Wehrens, R., Buydens L.M.C. (2006). SMIXTURE: a strategy of mixture models clustering of multivariate images// Journal of Chemometrics, Vol. 19, No. 11, pp.607–614.

149. Tritchler, D., Fallah, S., Beyene, J. (2005). A spectral clustering

method for microarray data// Computational Statistics and Data Analysis, Vol. 49, No. 1, pp.63–76.

150. Xia W., Cao M. Clustering in diffusively coupled networks// Automatica, vol. 47, no. 11, pp. 2395-2405, 2011.

151. Xiao L., Boyd S. Fast linear iterations for distributed averaging// Systems & Control Letters, vol. 53, no. 1, pp. 65-78, 2004.

152. Xu, Z.Y., Shang, S.C., Qian, W.B., Shu, W.H. (2011). A method for fuzzy risk analysis based on the new similarity of trapezoidal fuzzy numbers// Expert Systems with Applications, Vol. 37, No. 3, pp.1920–1927.

153. Yu, Z., Wong, H-S. (2006). Mining uncertain data in low-dimensional subspace// The 18th International Conference on Pattern Recognition, (ICPR'06), 0-7695-2521-0/.

154. Zadeh, L.A. (1965). Fuzzy sets// Information and Control, Vol. 8, No. 3, pp.338–353.

155. Zaharia M. et al. Das T. et al. Discretized streams: Fault-tolerant streaming computation at scale// Proc. of the 24th ACM Symposium on Operating Systems Principles. ACM, 2013. pp. 423–438. DOI: 10.1145/2517349.2522737.

156. Zelazo D., Bürger M. On the definiteness of the weighted laplacian and its connection to effective resistance// Proc. IEEE Conf. Decision and Control, pp. 2895-2900, 2014.

157. Zelazo D., Bürger M. On the robustness of uncertain consensus networks// IEEE Trans. Contr. Netw. Syst., vol. 4, no. 2, pp. 170-178, 2015.

158. Zhang, C., Gao, M., Zhou, A. (2009). Tracking high quality clusters over uncertain data streams// Proceedings of the IEEE International Conference on Data Engineering, Vol. 1, pp.1641–1648.

159. Zhang, T., Ramakrishnan, R., Livny, M. (1993). BIRCH: an efficient data clustering method for very large databases// Proc. ACM SIGMOD Conf. Management of Data, pp.103–114.