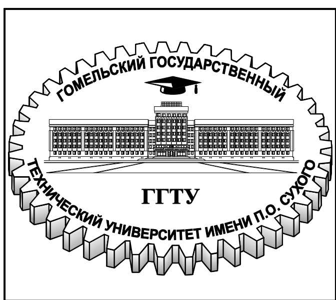


Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический университет имени П. О. Сухого»

Кафедра «Информационные технологии»

И. Л. Стефановский, К. С. Курочка

ОБЛАЧНЫЕ ТЕХНОЛОГИИ И СРЕДСТВА ОБРАБОТКИ БОЛЬШИХ ОБЪЕМОВ ИНФОРМАЦИИ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ
для студентов специальностей 1-40 05 01 «Информационные системы и технологии (по направлениям)» и 1-40 80 04«Информатика и технологии программирования» дневной и заочной форм обучения

Электронный аналог печатного издания

Гомель 2021

Библиотека ГГТУ им. П. О. Сухого

УДК 004.75(075.8)
ББК 32.973.4я73
С79

*Рекомендовано к изданию научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого*
(*протокол № 10 от 04.05.2020 г.*)

Рецензент: зав. каф. «Автоматизированные системы обработки информации» Гомельского государственного университета имени Ф. Скорины канд. техн. наук, доц. *В. Д. Левчук*

Стефановский, И. Л.

С79 Облачные технологии и средства обработки больших объемов информации : учеб.-метод. пособие для студентов специальностей 1-40 05 01 «Информационные системы и технологии (по направлениям)» и 1-40 80 04 «Информатика и технологии программирования» днев. и заоч. форм обучения / И. Л. Стефановский, К. С. Курочка. – Гомель : ГГТУ им. П. О. Сухого, 2021. – 198 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

ISBN 978-985-535-460-5.

Пособие знакомит студентов с архетиктурой, разработкой и использованием облачных информационных систем уровня предприятия, средствами обработки больших объемов информации.

Для студентов специальностей 1-40 05 01 «Информационные системы и технологии (по направлениям)» и 1-40 80 04 «Информатика и технологии программирования» дневной и заочной форм обучения.

УДК 004.75(075.8)
ББК 32.973.4я73

ISBN 978-985-535-460-5

© Стефановский И. Л., Курочка К. С., 2021
© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2021

Библиотека ГГТУ им. П. О. Сухого

Оглавление

Предисловие ... 5
Глава 1. Введение в облачные информационные системы ... 8
1.1. Размещение облачных приложений ... 9
1.2. Основные характеристики облачных вычислений ... 10
1.3. Облачные вычисления и предоставляемые ими сервисы ... 12
1.4. Облачные сервисы и границы управляемости ... 15
1.5. Существующие облачные платформы ... 15
Глава 2. Платформа Windows Azure ... 18
2.1. Обзор платформы Windows Azure ... 18
2.2. Компоненты облачной платформы ... 20
2.2.1. Web-сайты ... 20
2.2.2. Виртуальные машины ... 21
2.2.3. Облачные службы ... 22
2.2.4. Мобильные службы ... 22
2.2.5. Данные большого объема ... 24
2.2.6. Службы мультимедиа ... 27
2.3. Развитие платформы ... 28
2.4. Магазин Windows Azure ... 29
Глава 3. Разработка приложений с Windows Azure Cloud Services ... 30
3.1. Создание проекта в Visual Studio ... 31
3.2. Создание модели данных для элементов в Table Storage ... 33
3.3. Создание Web-роли для отображения гостевой книги ... 37
3.4. Организация очереди рабочих элементов ... 42
Глава 4. Авторизация и безопасность с Windows Azure Active Directory ... 44
4.1. Аутентификация на базе утверждений ... 44
4.2. Федеративная аутентификация в Windows Azure ... 45
4.3. Программная реализация проверки токенов безопасности на стороне клиента ... 48
4.4. Программная модель Windows Identity Foundation ... 49
4.5. Windows Azure Access Control Service ... 50
4.6. Active Directory Federation Services 2.0 ... 52
4.7. Многофакторная проверка подлинности Windows Azure ... 53
Глава 5. Хранение и обработка данных с Windows Azure Storage и Windows Azure SQL Databases ... 55
5.1. Блобы ... 56
5.2. Таблицы ... 59

Библиотека ГГТУ им. П.О. Сухого

5.3. Очереди ... 62
5.4. Партиции ... 65
5.5. Избыточность хранилища Windows Azure Storage ... 68
5.6. Диагностика хранилища ... 70
5.7. Обеспечение безопасности с использованием Shared Access Signatures ... 71
5.8. Windows Azure SQL Databases ... 74
Глава 6. Бизнес-аналитика и анализ данных с SQL Reporting и Hadoop ... 81
6.1. Обзор Business Intelligence ... 81
6.2. Создание отчетов в Report Builder ... 83
6.3. Анализ данных с Hadoop ... 88
Глава 7. Доступ к сервисам предприятия с Windows Azure Service Bus ... 96
7.1. Сценарии использования Windows Azure Service Bus ... 100
7.2. Windows Azure Notification Hubs ... 102
7.3. Определение дубликатов сообщений ... 103
7.4. Шаблон проектирования каналов и фильтров ... 104
Глава 8. Apache Kafka ... 113
8.1. Общие сведения об Apache Kafka в Azure HDInsight ... 113
8.2. Архитектура Apache Kafka в HDInsight ... 114
8.3. Создание кластера Apache Kafka в Azure HDInsight с помощью портала Azure ... 116
8.4. Получение сведений об узлах Apache Zookeeper и брокера ... 124
8.5. Создание кластера Apache Kafka в Azure HDInsight с помощью шаблонов Resource Manager ... 125
8.6. KafkaProducer API ... 133
Глава 9. Apache Storm ... 140
9.1. Общие сведения об Apache Kafka в Azure HDInsight ... 140
9.2. Создание топологии ... 141
9.3. Определение топологии ... 146
Глава 10. Параллельное программирование на основе MPI ... 150
10.1. MPI: основные понятия и определения ... 150
10.2. Введение в разработку параллельных программ с использованием MPI ... 155
10.3. Операции передачи данных между двумя процессами ... 170
10.4. Производные типы данных в MPI ... 180
10.5. Управление группами процессов и коммуникаторами ... 189
Контрольные вопросы ... 196
Литература ... 198

Предисловие

Современный специалист в области разработки программного обеспечения должен уметь пользоваться технологиями облачных сервисов и владеть основами создания облачных приложений различного уровня сложности.

Дисциплины «Введение в облачные вычисления» и «Облачные технологии и средства обработки больших объемов информации» знакомят студентов и магистрантов с основными концепциями, технологиями разработки облачных информационных систем (ИС) уровня предприятия.

Целью вышеуказанных дисциплин является подготовка специалиста, владеющего знаниями и практическими навыками по архитектуре, разработке и использованию облачных ИС уровня предприятия.

Основными задачами дисциплин являются следующие:

– изучение студентами и магистрантами теоретических основ и технологий разработки облачных ИС уровня предприятия;

– приобретение студентами и магистрантами практических навыков по проектированию и особенностям использования облачных ИС в сетях на основе технологии *Windows Azure*;

– освоение студентами и магистрантами технологий: применения средств проектирования облачных ИС на основе технологии *Windows Azure*, доступа к базам данных (БД) с использованием *Windows Azure*, разработки *web*-систем на основе технологии *Windows Azure*, технологий удаленного доступа к облачным ИС.

После изучения дисциплин студенты и магистранты должны иметь представление:

– о принципах программирования облачных информационных систем;

– принципах функционального и логического программирования;

– различных технологиях создания программных комплексов;

– перспективах развития технологий программирования облачных информационных систем.

В результате изучения учебных дисциплин студенты и магистранты должны:

• *знать*:

– теоретические основы и базовые технологии проектирования и использования современных облачных информационных систем;

Библиотека ГГТУ им. П.О.Сухого

– технологии организации взаимодействия серверной и клиентской частей облачных информационных систем на основе технологии *Windows Azure*;

– технологии организации работы облачных информационных систем;

• *уметь*:

– разработать клиент-серверную архитектуру облачных информационных систем согласно требованиям заданной предметной области;

– разработать клиентскую и серверную части облачных информационных систем на основе технологии *Windows Azure*, используя инструментальные средства создания внешних приложений и реляционную СУБД;

– организовать работу созданной облачной информационной системы в многопользовательском режиме.

• *владеть*:

– методами и технологиями разработки современных облачных информационных систем на основе технологии *Windows Azure*;

– технологией и методами конструирования программ на основе поставляемых библиотек и инструментальных средств разработки выбранной платформы;

– техникой, методами и средствами организации взаимодействия и обработки данных с использованием современных СУБД на основе технологии *Windows Azure*;

– методами разработки программных приложений в облачной среде;

– приемами и средствами отладки разрабатываемых программ и систем.

Главы 1–7 настоящего учебно-методического пособия предназначены для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» и соответствуют программе курса «Введение в облачные вычисления». Главы 8–10 предназначены для магистрантов специальности 1-40 80 04 «Информатика и технологии программирования» и соответствуют программе курса «Облачные технологии и средства обработки больших объемов информации».

Пособие разработано в ходе реализации ГГТУ им. П. О. Сухого проекта MaCICT (Modernisation of Master Curriculum in Information Computer Techologies) в рамках Программы Европейского Союза Erasmus+.

Библиотека ГГТУ им. П.О.Сухого

В данном проекте предствлены лучшие практики европейских университетов-партнеров с целью внедрения в учебный процесс изучения гибких навыков, позволяющих повысить конкурентоспособность выпускников ИТ-специальностей на рынке труда.

Книга предназначена для изучения на первой ступени высшего образования по специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» и на второй ступени по специальности 1-40 80 04 «Информатика и технологии программирования».

Библиотека ГГТУ им.П.О.Сухого

Глава 1. Введение в облачные информационные системы

Облачные вычисления и технологии являются сегодня одним из ведущих трендов мирового ИТ-рынка. Облачные вычисления представляют собой высокоэффективный инструмент повышения прибыли и расширения каналов продаж для независимых производителей программного обеспечения (Independent Software Vendors, ISV), операторов связи и VAR-посредников, расширяющих возможности существующих продуктов с целью их перепродажи конечным пользователям. Облачный подход позволяет организовать динамическое предоставление услуг, когда пользователи могут производить оплату по факту и регулировать объем своих ресурсов в зависимости от реальных потребностей без долгосрочных обязательств.

Существует большое количество вариантов определения для терминов «облачные вычисления», или «облачная платформа». Это связано с тем, что различные поставщики стараются подчеркнуть уникальность своих предложений и выбирают разные названия, которые часто не совсем верно отражают реальную суть предлагаемых сервисов. Когда говорят про облачную платформу, обычно используют такие термины, как «инфраструктура как сервис» (*IaaS*), «платформа как сервис» (*PaaS*) или «приложения как сервис» (*SaaS*).

Облачные вычисления обладают многими преимуществами по сравнению с традиционными решениями для построения инфраструктур предприятий, предложению сервисов и услуг и т. п. Среди таких преимуществ выделяются:

- гибкость;
- масштабируемость;
- оплата за фактически использованные ресурсы;
- высокая надежность и отказоустойчивость.

Предлагаемые облачные платформы и сервисы сегодня отличаются как по функционалу, так и стоимости. В зависимости от поставленных задач необходимо правильно выбрать поставщика и определить оптимальный план использования.

Корпорация *Microsoft* предлагает свою платформу *Windows Azure*, которая содержит множество сервисов, имеет гибкие планы подписок, поддерживает различные средства и языки разработки приложений. Платформа быстро развивается и на сегодня она включает в себя более пяти основных видов услуг – от облачного хостинга веб-

сайтов до полноценной архитектуры предприятия со множеством сервисов, виртуальными машинами, хранилищами данных и пр.

При рассмотрении темы облачных вычислений необходимо рассмотреть следующие концепции [1]:

– размещение облачных приложений;
– основные характеристики облачных вычислений;
– предоставляемые сервисы;
– границы управляемости.

Получив ответы на вышеперечисленные вопросы, можно перейти к рассмотрению существующих платформ и бизнес-моделей, которые они предлагают.

1.1. Размещение облачных приложений

Обсуждая облачные вычисления, следует обращать внимание на то, где располагаются приложения. В настоящее время существует три основных модели расположения приложений:

– в инфраструктуре заказчика;
– у компании-хостера;
– в облаке.

Расположение в инфраструктуре заказчика (*on premises*). Это наиболее традиционная модель развертывания приложений, существующая уже десятки лет. Размещение приложений в локальной инфраструктуре предполагает существенные начальные инвестиции в аппаратные ресурсы, программное обеспечение, сетевую инфраструктуру и персонал.

Такая модель – оплата, приобретение, владение – напрямую связана с высокими капитальными затратами, но в то же время она обеспечивает полный контроль за инфраструктурой, аппаратным и программным обеспечением.

Расположение у компании-хостера (*hosting*). Такая модель развертывания приложений, называвшаяся ранее *Application Services Prodiver* (*ASP*), а затем – *SaaS*, или просто «хостинг» получила свое развитие несколько лет назад и является одним из наиболее популярных способов снижения расходов на информационные технологии. Она основана на аренде аппаратной платформы, программного обеспечения, соответствующей инфраструктуры и персонала, выполняющего ее обслуживание. Такая модель отличается меньшим контролем за инфраструктурой, аппаратным и программным обеспечением и ба-

Библиотека ГГТУ им. П.О. Сухого

зируется на оплате фиксированного числа ресурсов, что обычно предполагает оплату даже в тех случаях, когда арендуемые ресурсы не используются.

Расположение в облаке (*cloud*). Данная модель появилась совсем недавно. Она предполагает оплату по факту использования, арендуемых аппаратных и программных ресурсов, что приводит к существенному снижению начальных расходов и переходу от капитальных инвестиций к операционным расходам. Такая модель отличается практически отсутствием контроля за инфраструктурой и аппаратным обеспечением, а при аренде программного обеспечения – еще и отсутствием контроля за ним.

Каждый подход имеет свои достоинства и недостатки, но, с точки зрения экономики, самой важной характеристикой является оплата по факту использования, реализуемая именно облачными вычислениями.

Облачные вычисления – это такой подход к размещению, предоставлению и потреблению приложений и компьютерных ресурсов, при котором приложения и ресурсы становятся доступны через Интернет виде сервисов, потребляемых на различных платформах и устройствах. Оплата таких сервисов осуществляется по их фактическому использованию.

1.2. Основные характеристики облачных вычислений

Масштабируемость

Ввод новых продуктов и сервисов, расширение канала продаж и количества заказчиков требуют от информационных систем организации выдерживать растущие нагрузки и обрабатывать большие объемы данных. Быстрая и надежная работа, исключающая отказы в обслуживании, задержки в ответах от системы и сбои позволяют повысить лояльность и удовлетворенность заказчиков. Масштабируемое приложение позволяет выдерживать большую нагрузку, за счет увеличения количества одновременно запущенных экземпляров. Как правило, для одновременного запуска множества экземпляров используется типовое оборудование, что снижает общую стоимость владения и упрощает сопровождение инфраструктуры.

Эластичность

Гибкая реакция на изменяющиеся условия ведения бизнеса является одной из характеристик успешного бизнеса. Например, сложившаяся рыночная конъюнктура и действия конкурентов могут по-

Библиотека ГГТУ им. П.О.Сухого

требовать быстро внедрить новый продукт или услугу, проведя при этом полный цикл планирования, проектирования и разработки информационной системы. Эластичность позволяет быстро нарастить мощность инфраструктуры, без необходимости проведения начальных инвестиций в оборудование и программное обеспечение. Эластичность связана с масштабируемостью приложений, так как решает задачу моментального изменения количества вычислительных ресурсов, выделяемых для работы информационной системы.

Мультитенантность

Мультитенантность – это один из способов снижения расходов за счет максимального использования общих ресурсов для обслуживания различных групп пользователей, разных организаций, разных категорий потребителей и т. п. Мультитенантность может быть особенно привлекательна для компаний-разработчиков приложений, так как позволяет снизить собственные расходы на оплату ресурсов облачной платформы и максимально использовать доступные вычислительные ресурсы.

Оплата за использование

Оплата использованных ресурсов – это еще один атрибут облачных вычислений, позволяющий перевести часть капитальных издержек в операционные. Приобретая только необходимый объем ресурсов, можно оптимизировать расходы, связанные с работой информационных систем организации. А в сочетании с мультитенантностью, разделяя ресурсы между различными потребителями, можно снизить расходы еще больше. Эластичность позволит быстро изменить объем ресурсов в сторону увеличения или уменьшения, тем самым, приведя расходы на ИТ в соответствие с фактическими потребностям организации.

Самообслуживание

Быстрый вывод на рынок нового продукта или услуги в современных условиях сопровождается развертыванием или модификацией информационных систем. Традиционно развертывание информационной системы предваряется определением спецификации оборудования, его закупкой и настройкой. В зависимости от того, кем производится процесс разработки приложения (контрактором или внутренними силами), он может потребовать выделения аппаратных ресурсов и установку программного обеспечения. Все это может занять длительное время: месяцы и даже годы. Самообслуживание позволяет потребителям запросить и получить требуемые ресурсы за считанные минуты.

Библиотека ГГТУ им. П.О. Сухого

Как можно заметить, только сочетание нескольких атрибутов облачных вычислений приводит к достижению задачи повышения доходов и снижения расходов. Так, оплата только использованных ресурсов максимально эффективна в сочетании с эластичностью инфраструктуры.

Эластичность, в свою очередь, предполагает, что приложения масштабируются, в противном случае, быстрое выделение ресурсов не приведет к повышению производительности.

Выше рассмотрели, как основные атрибуты облачных вычислений могут влиять на решение задач повышения доходов и снижения расходов организации. Нужно также понимать, что переход в облако не является тривиальной задачей и часто требует пересмотра и изменения архитектуры существующих решений, а иногда – полного отказа от них в пользу создания новых, реализованных с учетом возможностей, предоставляемых облачными платформами. В зависимости от архитектуры существующих приложений и технологий, на которых они реализованы, их перенос на облачную платформу может привести к получению ряда преимуществ, а может – к появлению дополнительных проблем, связанных, например, с обеспечением совместимости или ограничениями реализации серверной платформы на уровне облака. Как один из шагов по адаптации облачных вычислений можно рассмотреть переход к архитектуре, ориентированной на сервисы.

1.3. Облачные вычисления и предоставляемые ими сервисы

Облачные вычисления и предоставляемые ими сервисы (например, вычислительные мощности или хранилища) можно сравнить с коммунальными услугами. Так же как в жару или холод меняется потребление воды и электричества, так и потребление сервисов, предоставляемых «облачными» платформами, может возрастать или уменьшаться в зависимости от повышения или понижения нагрузок.

Сходность сервисов и коммунальных услуг заключается в нескольких аспектах. Во-первых, и в том и в другом случае потребители платят только за реальную утилизацию. Во-вторых, и те и другие ресурсы берутся в аренду, т. е. в большинстве случаев не нужно подключаться к колодцу для получения воды или непосредственно к электростанции для получения электричества, – поставщики таких сервисов обеспечивают их доступность в виде арендуемых «ресурсов», оставляя за собой вопросы создания и поддержания инфра-

Библиотека ГГТУ им. П.О.Сухого

структуры. В-третьих, заключая договор с соответствующей организацией, подразумевается доступность тех или иных ресурсов. Можно выделить следующие основные сервисы, предоставляемые облачными платформами.

Программное обеспечение как сервис (*SaaS*)

Модель предоставления программного обеспечения как сервиса (*Software as a Service*, *SaaS*) обеспечивает возможность аренды приложений. Программное обеспечение как сервис включает платформу как сервис и инфраструктуру как сервис. Примером приложения как сервиса может быть *Business Productivity Online Suite*.

Модель предоставления программного обеспечения как сервиса является моделью обеспечения доступа к приложениям через Интернет с оплатой по факту их использования. Данная модель является наиболее распространенной на сегодняшний день моделью предоставления облачных сервисов. Организации могут реализовывать подобную модель предоставления сервиса из частных облаков, используя внутренние сетевые каналы, дополнительно защищенные и не связанные с Интернетом.

Потребителями данного типа сервисов являются конечные пользователи, которые работают с приложениями, предоставляемыми в «облаке». Соглашение о предоставлении сервисов (*SLA*) обычно покрывает такие характеристики сервисов, как их доступность (*uptime*) и производительность. Возможности настройки приложений под нужды потребителей минимальны или вообще отсутствуют, их уровень диктуется требованиями рынка или возможностями поставщиков таких приложений.

Оплата конечного сервиса, как правило, производится ежемесячно и рассчитывается на основе количества пользователей приложения.

Платформа как сервис (*PaaS*)

Модель предоставления платформы как сервиса (*Platform as a Service*, *PaaS*) обеспечивает возможность аренды платформы, которая обычно включает операционную систему и прикладные сервисы. Платформа как сервис облегчает разработку, тестирование, развертывание и сопровождение приложений без необходимости инвестиций в инфраструктуру и программную среду. Платформа как сервис также включает и инфраструктуру как сервис. Примером платформы как сервис может служить *Windows Azure*, *Amazon Web Services* (*AWS*).

Библиотека ГГТУ им. П.О.Сухого

Здесь потребителями являются сами компании, разработавшие приложения. Платформа обеспечивает среду для выполнения приложений, сервисы по хранению данных и ряд дополнительных сервисов, например, интеграционные или коммуникационные. Соглашение о предоставлении сервисов (*SLA*) обычно покрывает такие характеристики сервисов, как доступность среды выполнения приложений и ее производительность. Возможности настройки приложений под нужды потребителей практически не ограничены. Ограничением может послужить лишь функциональность сервисов, предоставляемых на уровне платформы. При этом необходимо понимать: для того чтобы воспользоваться возможностями облачной платформы, необходимо значительно модернизировать или вообще написать заново существующие приложения.

Оплата облачной платформы рассчитывается исходя из объема использованных вычислительных ресурсов, таких как:

- время работы приложения;
- объем данных и количество операций с данными (транзакций);
- сетевой трафик.

Провайдер облачной платформы может предоставлять существенные скидки при приобретении определенного объема ресурсов.

Инфраструктура как сервис

Модель предоставления инфраструктуры (аппаратных ресурсов) как сервиса (*Infrastructure as a Service*, *IaaS*) обеспечивает возможность аренды таких инфраструктурных ресурсов, как серверы, устройства хранения данных и сетевое оборудование. Управление всей инфраструктурой осуществляется поставщиком сервисов, а потребитель управляет только операционной системой и установленными приложениями. Такие сервисы обычно оплачиваются по их фактическому использованию и позволяют пользователю увеличивать или уменьшать объем используемой инфраструктуры через специальные порталы, предоставляемые поставщиками сервисов.

Здесь потребителями являются владельцы приложений, ИТ-специалисты, подготавливающие образы операционной системы (ОС) для их запуска в сервисной инфраструктуре. Облачная платформа предоставляет сервисы для запуска виртуальных машин и сервисы хранения данных. Соглашение о предоставлении сервисов (*SLA*) обычно покрывает такие характеристики сервисов, как доступность виртуального сервера, время развертывания образа ОС. В данной сервисной модели могут быть запущены практически любые приложения, установленные на стандартные образы ОС.

Библиотека БГУИР

Как и в случае с *PaaS*, оплата инфраструктуры как сервиса, обычно производится исходя из объема использованных ресурсов.

1.4. Облачные сервисы и границы управляемости

Обсуждая различные типы облачных сервисов: программное обеспечение, платформу и инфраструктуру как сервис, следует обращать внимание на так называемые границы управляемости, т. е. на то, чем, в сравнении с традиционными моделями развертывания в собственной инфраструктуре, можно управлять при переходе на облачную платформу. По понятным причинам инфраструктура как сервис предоставляет большие возможности по настройке отдельных компонентов, тогда как платформа как сервис и программное обеспечение как сервис практически минимизируют эти возможности.

При развертывании собственной инфраструктуры необходимо управлять всеми ее компонентами – от сетевых ресурсов до выполняющихся приложений. Тогда как при использовании модели *IaaS* можно контролировать такие компоненты, как среда исполнения кода, безопасность и интеграция, базы данных, и т. п. При переходе к модели *PaaS* все компоненты платформы предоставляются как сервисы с ограниченными возможностями для управления ими. Это сделано, чтобы предоставить в распоряжение потребителей оптимально сконфигурированную платформу, не требующую дополнительных настроек.

1.5. Существующие облачные платформы

На рынке сегодня существует множество платформ для организации облачных вычислений. Существуют как проприетарные (коммерческие), так и открытые (свободные). На основе открытых платформ, таких как *OpenStack* [2], *Cloud Foundry* [3] многие компании создают свои инфраструктуры и предлагают средства для их управления, в частности, предоставляют комплексы для превращения имеющихся ресурсов в облака.

Для того чтобы выбрать наиболее подходящую платформу и провайдера необходимо четко сформулировать требования, предъявляемые к облаку, а также произвести пробное тестирование всех возможных платформ.

Из наиболее активных и серьезных игроков рынка облачных вычислений следует отметить следующие платформы и компании [4]:

Библиотека ГГТУ им. П. О. Сухого

• ***Amazon Web Services*** [5]

Amazon является пионером рынка облачных платформ и на сегодняшний момент – это безусловный лидер рынка. Особенность *AWS* в том, что это инфраструктурный сервис (*IaaS*), который предоставляет максимум свободы разработчикам в выборе платформы и среды разработки. *AWS* подходит как для хостинга корпоративных приложений и контента, так и для построения *SaaS* сервисов.

• ***Rackspace*** [6]

Rackspace является наиболее близким к *Amazon* (это тоже *IaaS* платформа) и в части стоимости и простоты администрирования – даже обходит своего конкурента. В отличие от *Amazon*, которая концентрирует усилия на развитии инструментов для развертывания и управления облачной инфраструктурой, *Rackspace* стремится быть ближе к прикладным приложениям [7]. Кроме того, *Rackspace* предоставляет базовые сервисы для совместной работы: почтовый сервер (*Rackspace Email*) и файловый сервер (*Rackspace Cloud Drive*), которые можно будет интегрировать в свои облачные приложения.

• ***Windows Azure*** [8]

Это идеальная облачная платформа для *Microsoft*-ориентированных разработчиков и компаний. Впрочем, *Windows Azure* также поддерживает *PHP*, *MySQL*, *Ruby on Rails*, *Python*, *Java*, *Eclipse* и *Zend*. Главным преимуществом *Azure* перед *Amazon Web Services* и *Rackspace Cloud* является высокий уровень автоматизации. Кроме того, эта платформа позволяет легко интегрировать размещаемые на ней приложения с локальной ИТ-инфраструктурой компании с помощью стандартов *SOAP*, *REST* и *XML* (таким образом, поддерживает схему *S* + *S*).

Данная платформа будет рассмотрена подробнее в следующем разделе.

• ***Google App Engine + Google Apps*** [9]

Платформа *Google App Engine* отличается гуманным отношением к стартапам – предоставляет ограниченные бесплатные ресурсы (дисковое пространство и трафик), которые весьма кстати для начинающих *SaaS* сервисов. *GAE* поддерживает пока только два языка программирования – *Python* и *Java*. *GAE* в основном ориентирован на создание *SaaS* сервисов для малого бизнеса. Кроме инфраструктурной платформы *Google* предоставляет набор *API* для интеграции сервиса с популярными приложениями *Google Apps* и супермаркет приложений *Google Apps Marketplace* для вывода вашего сервиса на рынок.

Библиотека ГГТУ им. П.О.Сухого

• ***Force.com*** [10]

Платформа компании *Salesforce* [11] – *Force.com* – претендует на роль монополиста на рынке корпоративных *SaaS* приложений. Платформа построена вокруг самой успешной корпоративной *SaaS* системы – *Salesforce* и позволяет создавать дополнения к этой системе или независимые приложения. *Force.com* предоставляет широкий выбор инструментов разработки (*Apex*, *Flash*, *Java*), конструктор интерфейсов, готовые модули (аутентификация, социальные инструменты, бизнес-процессы, аналитика) и супермаркет приложений с огромной пользовательской базой.

• ***VMWare vCloud*** [12]

VMWare vCloud – не является самодостаточной облачной платформой. Это промежуточный слой, который несколько партнеров *VMWare* предоставляют поверх своей серверной инфраструктуры (последней к списку партнеров присоединилась *Salesforce*). *VMWare* – это мировой лидер на рынке систем виртуализации, поэтому главным преимуществом этой платформы является поддержка виртуальных образов приложений. В частности, это позволяет быстро и просто переносить локальные бизнес-приложения на облачную платформу без проблем, связанных с переносом сопутствующей ИТ-инфраструктуры.

• ***IBM Cloud*** [13]

IBM Cloud в основном ориентирована на крупные компании и ресурсоемкие процессы: разработка и тестирование ПО, хранение и аналитическая обработка огромных массивов данных. Очевидно, после недавнего приобретения сервиса *OmniConnect*, еще одной функцией этого облака станет интеграция разрозненных облачных систем и платформ.

Таким образом, на рынке представлено достаточно платформ, чтобы был выбор.

Библиотека ГГТУ им. П.О. Сухого

Глава 2. Платформа Windows Azure

2.1. Обзор платформы Windows Azure

Windows Azure – это открытая и гибкая облачная платформа, которая позволяет быстро выполнять построение приложений, развертывать их и управлять ими в рамках глобальной сети из центров данных, управляемых корпорацией Майкрософт. Можно осуществлять построение приложений с помощью любого языка, средства или любой платформы, а также интегрировать общедоступные облачные приложения с существующей ИТ-средой.

Высокая готовность

Windows Azure предлагает ежемесячное соглашение об уровне обслуживания на уровне 99,95 %, что позволяет создавать и запускать высокодоступные приложения, не сосредотачивая внимание на инфраструктуре. Эта платформа обладает возможностью автоматического применения исправлений для операционной системы и служб, встроенной балансировкой сетевой нагрузки и устойчивостью к аппаратным сбоям. Она поддерживает модель развертывания, которая позволяет обновлять приложение с нулевым временем простоя.

Открытость

Windows Azure позволяет использовать для построения приложений любой язык, любое средство или любую платформу. Компоненты и службы предоставляются с помощью открытых протоколов *REST*. Клиентские библиотеки *Windows Azure* доступны для нескольких языков программирования, выпускаются по лицензии с открытым исходным кодом и размещаются на сайте *GitHub*.

Неограниченные серверные ресурсы. Неограниченное хранилище

Windows Azure позволяет легко масштабировать приложения до любого размера. Это полностью автоматизированная платформа самообслуживания, которая позволяет подготавливать ресурсы к работе за считанные минуты. Гибко расширяйте или сокращайте использование ресурсов в соответствии со своими потребностями. Оплата производится только за ресурсы, используемые вашим приложением. Платформа *Windows Azure* доступна в нескольких центрах обработки данных по всему миру, что позволяет развертывать приложения ближе к клиентам. Сегодня это шесть датацентров, по два на регион (Северная Америка, Европа, Азия).

Библиотека ГГТУ им. П.О. Сухого

Расширенные возможности

Платформа *Windows Azure* является гибкой облачной платформой, которая способна удовлетворить любые потребности приложений. Она обеспечивает надежное размещение и масштабирование кода в ролях выполнения приложений. Для хранения данных можно использовать реляционные БД *SQL*, хранилища таблиц *NoSQL*, неструктурированные хранилища больших двоичных объектов, а при необходимости использовать компоненты *Hadoop* и службы бизнес-аналитики для интеллектуального анализа данных. Возможности безопасного обмена сообщениями платформы *Windows Azure* позволяют развертывать распределенные приложения и гибридные решения, работающие в смешанной облачной и локальной среде предприятия. Использование распределенного кэширования или сети кэширующих серверов (*CDN*) позволяет сократить задержку и улучшить временные характеристики приложения во всех точках земного шара.

Платформа *Windows Azure* предоставляет набор сервисов, которые, в основной массе, сходны с сервисами, используемыми разработчиками «традиционных» приложений.

Вычислительные сервисы. Представляют собой контейнеры для приложений с поддержкой современных технологий разработки, включая *.NET*, *Java*, *PHP*, *Python*, *Ruby on Rails* и нативный код.

Сервисы хранения данных. Масштабируемая распределенная система хранения данных, поддерживающая ряд моделей хранения, включая табличные структуры, бинарные объекты, асинхронные очереди сообщений, традиционные файловые системы и сети распределения контента (*CDN*, *content distribution networks*).

Коммуникационные сервисы. Доступны через облачную сервисную шину и могут использоваться как средство обмена сообщениями или брокер соединений с другими облачными сервисами или сервисами, находящимися у заказчиков.

Сервисы обеспечения безопасности. Сервисы управления доступом, основанные на политиках, которые поддерживают механизмы федерации и позволяют интегрироваться с существующими системами управления идентификацией.

Прикладные сервисы. Компоненты и сервисы, которые могут использоваться для разработки облачных приложений и прикладных сервисов.

Библиотека ГГТУ им. П.О.Сухого

2.2. Компоненты облачной платформы

Платформа *Windows Azure* состоит из следующих основных компонентов:

– web-сайты;
– виртуальные машины;
– мобильные службы;
– облачные службы;
– большие объемы данных (хранилища);
– мультимедиа.

Для каждого компонента возможны свои сценарии использования, причем они могут включать в себя несколько компонентов.

2.2.1. Web-сайты

Web-разработка является одиним из самых быстрорастущих трендов. Развитие Интернета и технологий, обеспечивающих доступ к нему, требует новых средств и моделей для развертывания сайтов и обеспечения их высокой доступности и надежности. Традиционные хостинги остаются популярными и постоянно обновляются, при этом предоставляют самые последние версии средств для поддержания сайтов.

Облачные платформы позволяют расширить возможности разработки и предоставляют высокую степень масштабируемости. Они предоставляют качественно новые услуги, которые отличаются большей гибкостью, управляемостью и т. д. Это, в свою очередь, позволяет управлять своими затратами и платить лишь за реально необходимые и использованные ресурсы, сокращая издержки. В начале можно начать с небольшого сайта с настройками по умолчанию. Далее, при необходимости, можно подобрать подходящую виртуальную машину под высоконагруженный сайт, увеличить трафик, добавить другие сервисы, такие как кэширование, *CDN*, базы данных *SQL*, хранилище и т. д.

Для создания сайтов можно использовать языки и приложения с открытым исходным кодом по своему усмотрению, а затем выполнить развертывание с помощью *FTP*, *Git* и *TFS*. Использование *Git* и *TFS* дает возможность настроить автоматическую публикацию сайта после того, как его последняя версия обновляется в системе управления версиями (СУВ). Настройка непрерывной интеграции и развертывания снимает необходимость в ручной сборке, тестировании и размещении. Все это будет выполняться автоматически.

Библиотека ГГТУ им. П.О.Сухого

Для создания web-сайта можно выбрать два пути:

– выбрать шаблон сайта (из представленных в галерее);

– создать свой сайт (*Quick Create* или *Create With Database*).

В галерее доступно множество видов сайтов и платформ, таких как *WordPress*, *KentikoCMS*, *Orchard CMS* и др. Во многих случаях выбор приложений из существующих обеспечит более быстрое создание необходимого портала, а также предоставит возможности по управлению им.

При создании сайта самостоятельно в панели управления необходимо подготовить виртуальную машину для него, создать БД (в случае необходимости) и выделить место под хранение. При таком сценарии будет предоставлен экземпляр виртуальной машины, на котором будет развернут сайт. Далее необходимо выбрать способ развертывания или публикации. Среди вариантов можно использовать не только *Git* и *TFS*, но и *Web Deploy* и *FTP Deploy*, доступные в *IDE* после выбора настроек публикации, которые могут быть загружены с портала.

После того как файлы сайта будут загружены, к нему можно получить доступ по адресу, который выдается автоматически в домене третьего уровня (*<your_name>.azurewebsites.net*) и имеет название вашего сайта (который был задан при создании). В случае необходимости *DNS* имя можно сменить на свое (это доступно для режимов работы отличных от *Free*, а также требует фиксированной оплаты).

В случае необходимости повышения производительности сайта, увеличения размера БД, необходимо выбрать более мощную виртуальную машину, а также БД.

2.2.2. Виртуальные машины

В *Windows Azure* можно легко использовать собственные образы *Windows Server* или *Linux*, а также выбрать образы из коллекции. Это позволяет сохранять полный контроль над образами и поддерживать их в соответствии с бизнес-требованиями. *Windows Azure* также помогает переносить приложения и инфраструктуру, не меняя существующий код, что ускоряет переход *SharePoint*, *SQL Server* и *Active Directory* в облако и экономит время и деньги.

Виртуальные машины следует использовать для следующих целей:

– *получение гибкости*. Виртуальные машины дают приложению мобильность, позволяя перемещать виртуальные жесткие диски (*VHD*) между локальной и облачной средой;

Библиотека ГГТУ им. П. О. Сухого

– *выполнение приложений в облаке*. Если компания использует популярные серверные приложения Майкрософт, виртуальные машины помогут применять те же локальные корпоративные приложения и инфраструктуру в облаке. Легко работайте с приложениями, такими как *Microsoft SQL Server*, *Active Directory* и *Microsoft SharePoint Server*;

– *удаленное управление*. С полным административным доступом можно удаленно подключаться к виртуальным машинам и управлять установленными на них приложениями.

Все виртуальные машины управляются расширенной версией гипервизора (*Hyper-V*) и располагаются в глобальных центрах обработки данных. Каждая виртуальная машина может иметь различные характеристики: число процессоров, объем памяти, объем хранилища (жесткого диска).

2.2.3. Облачные службы

Предоставляют возможность создания приложений и интерфейсов *API* с высокой доступностью и бесконечной масштабируемостью.

Облачные службы *Windows Azure* позволяют создавать приложения, которые остаются доступными даже во время обновления программного обеспечения и сбоев оборудования.

Соглашение об уровне обслуживания гарантирует степень доступности 99,95 % для приложения.

Каждому новому облачному приложению требуется мощный набор серверных служб. Облачные службы *Windows Azure* предоставляют все, что нужно для создания самых надежных и масштабируемых интерфейсов *API*.

Облачные службы *Windows Azure* предоставляют наиболее эффективную среду для создания самых современных распределенных вычислительных приложений на планете.

2.2.4. Мобильные службы

Windows Azure Mobile Services – набор сервисов, которые призваны облегчить разработчикам мобильных приложений создание и использование серверного бэкенда. Использование облака *Windows Azure* в качестве такого бэкенда позволит получить готовый функционал *push*-уведомлений, сохранения данных в облачное хранилище, аутентификации и авторизации пользователей без необходимости разворачивать собственную инфраструктуру.

Библиотека БГУИР

Доступ к сервисам доступен из *C#* и *JavaScript*. Команда разработчиков работает над публичным *REST API*, который позволит получать данные и работать с сервисами из любого языка. На сегодняшний день обеспечена официальная инструментальная поддержка *Windows Phone*, *iOS*, *Windows* 10. Также планируется добавление поддержки *Android*.

Сегодня *Windows Azure Mobile Services* предлагает следующий функционал:

– хранение пользовательских данных в облаке;
– аутентификация и авторизация пользователей в облаке;
– прием *push*-уведомлений от облачного сервиса.

Особенности:

– *REST API*, доступ с любого мобильного клиента;
– масштабирование по требованию;
– мониторинг потребления ресурсов и числа запросов в реальном времени;
– реляционное хранилище, поддержка *SQL*-запросов, индексов;
– автоматическое обновление схемы данных;
– разрешения, обработка запросов перед операциями *CRUD*;
– функциональная единая панель управления;
– бесплатно предоставляется 10 экземпляров.

Последнее обновление сервиса предоставило следующие возможности:

• **Поддержка платформы *iOS* и выпуск отдельного *iOS SDK***

Добавлены новые инструментальные средства для разработки *iOS*-приложений для *iPhone* и *iPad*. Эти инструменты выпущены с открытым исходным кодом под свободной лицензией *Apache* 2.0.

Для разработчиков *iOS*-приложений благодаря новому *SDK* упрощается доступ к сервисам хранения информации и авторизации через сторонние сервисы и сервис *Microsoft Account*. Поддержка *push*-уведомлений пока не доступна в новом *iOS SDK* и появится в скором будущем.

• **Поддержка сторонних сервисов авторизации: *Facebook*, *Twitter*, *Google***

В дополнение к уже предложенному сервису авторизации *Microsoft Account*, который можно было использовать для своих приложений ранее, в обновлении представлена поддержка сторонних сервисов авторизации: *Facebook*, *Twitter* и *Google*.

Библиотека ГГТУ им. П.О. Сухого

• **Использование *Windows Azure Tables*, *Blobs* и *Service Bus* внутри *Mobile services***

С обновлением сервиса у разработчиков появилась возможность использовать внутри скриптов *Mobile Services* вызовы к другим сервисам облачной платформы: средствам хранения информации *Tables* и *Blobs* и средству интеграции *Service Bus*.

• **Отправка почтовых и *SMS* сообщений**

В дополнение к использованию облачных сервисов самой платформы из серверных скриптов *Mobile Services* в обновлении добавлен функционал, позволяющий отправлять почтовые сообщения (используя *SendGrid*) и *SMS*-сообщения. Можно бесплатно отправлять до 25000 почтовых сообщений в месяц.

Аналогично отправке почтовых сообщений в обновлении появилась возможность отправлять *SMS*-уведомления. Для этого используется сервис *Twilio*, который предлагает разработчикам *Windows Azure* 1000 бесплатных сообщений.

2.2.5. Данные большого объема

Windows Azure предоставляет множество служб, помогающих управлять данными в облаке, которые называются *Windows Azure Storage*. Каждый сервис подходит для хранения определенного типа данных.

Таблицы – представляют собой структурированное хранилище. Каждая таблица состоит из набора объектов, каждый из которых имеет набор названий свойств и их значений. Один объект может иметь до 256 свойств. Таблицы распределены таким образом, чтобы максимально поддерживать балансировку нагрузок.

Бинарные объекты – используются для хранения больших бинарных объектов (файлов). Предоставляется простой интерфейс для хранения именованных файлов вместе с метаданными и обеспечивается поддержка сети распределения контента. Бинарные объекты располагаются в контейнерах, каждый из которых содержит набор объектов.

Бинарные объекты могут быть двух видов – блочные, оптимизированные для потокового обмена данными и страничные, оптимизированные для случайных операций ввода/вывода. Размер блочного бинарного объекта не может превышать 200 Гб, а размер страничного бинарного объекта – 1 Тб.

***BLOB*-объекты** – это простейший способ хранения больших объемов неструктурированных текстовых или двоичных данных, та-

Библиотека ГГТУ им. П.О.Сухого

ких как видео, музыкальный файлы и изображения. *BLOB*-объекты – это управляемая служба, сертифицированная по стандарту *ISO* 27001, которая может автоматически масштабироваться до объема в 100 ТБ и доступ к которой можно получить практически из любого места с помощью интерфейса *REST* и управляемых *API*.

Очереди – надежное хранилище сообщений. Обычно используется для обеспечения коммуникаций между ролями. Данный сервис оперирует очередями, в которых располагаются сообщения. Допускается использование неограниченного числа очередей, а очереди могут содержать неограниченное число сообщений. Размер сообщения ограничен 8 Кб.

Диски – тома *NTFS*, доступные для приложений, выполняющихся в инфраструктуре *Windows Azure*. Диски (*Windows Azure Drives*) хранятся как отформатированные под *NTFS* виртуальные диски (*Virtual Hard Drives*, *VHDs*) в страничных бинарных объектах. Так как диски поддерживают сохранение информации, они могут использоваться приложениями, которым необходимо сохранять состояния. После того как диск *Windows Azure* смонтирован, он доступен программно через стандартные интерфейсы *NTFS*. Использование дисков *Windows Azure* может существенно упростить миграцию существующих приложений на платформу *Windows Azure*.

Ниже приведем несколько примеров, иллюстрирующих сценарии использования некоторых сервисов хранения данных.

Хранилище бинарных объектов: возможность хранения резервных копий, отчетов для их быстрого получения в случае необходимости.

Табличное хранилище: возможность хранения состояний веб-приложений, например, в случае электронной коммерции – хранение покупательской корзины или текущего состояния заказа.

Очереди: web-приложение может вызывать сервисы, располагаемые на платформе *Windows Azure* и осуществлять коммуникации между web-ролями и прикладными ролями в рамках одного или нескольких приложений.

Диски: за счет поддержки файловой системы *NTFS* могут использоваться сервисами для обеспечения поддержки традиционных файловых операций – чтение/запись, например, для протоколирования операций или сохранения временных данных.

Для хранения реляционных данных, например, при переносе локальной базы данных в облако, следует использовать компонент платформы *Windows Azure – SQL Azure*.

Библиотека ГГТУ им. П.О. Сухого

SQL Azure – это способ предоставления реляционной базы данных *Microsoft* как сервиса. Данный сервер базируется на технологиях *Microsoft SQL Server* и обеспечивает устойчивую к ошибкам, масштабируемую и мультитенантную базу данных, доступную как сервис. Как и в случае с *Windows Azure*, *SQL Azure* – это не просто хостинг *Microsoft SQL Server*.

Работа *SQL Azure* базируется на компоненте *Cloud Fabric*, который управляет экземплярами базы данных и обеспечивает их развертывание, администрирование, обновление, мониторинг и поддерживает весь жизненный цикл работы с данными. От пользователей требуется только выполнение таких задач, как создание схемы и ее поддержание, оптимизация запросов и управление безопасностью.

Основные компоненты *SQL Azure* показаны на рис. 2.1.

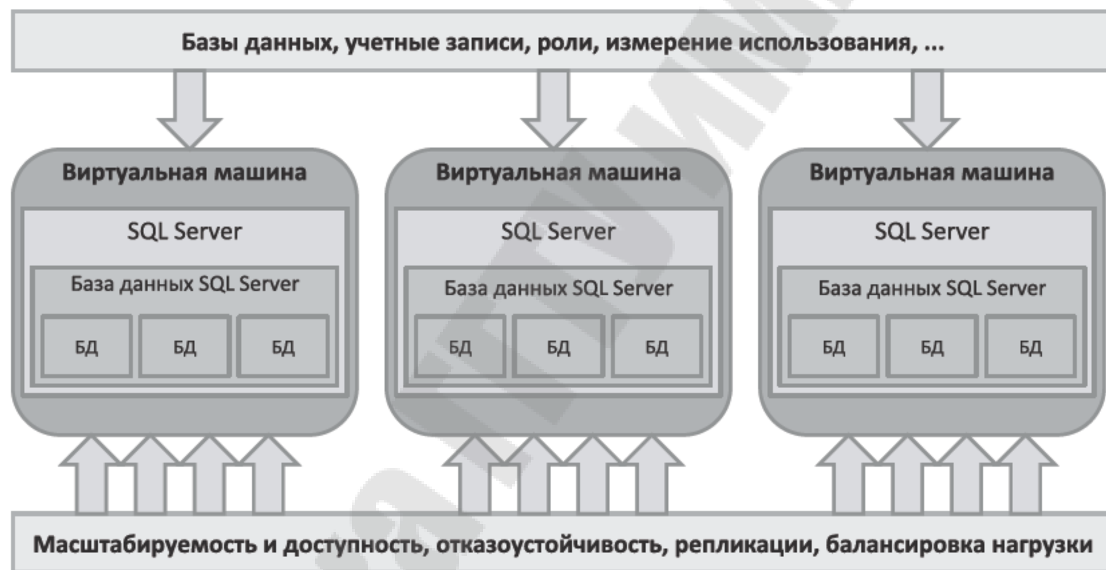


Рис. 2.1. Компоненты *SQL Azure*

Экземпляр базы *SQL Azure* реализован как три реплики в рамках серверной инфраструктуры, поддерживаемой *Cloud Fabric*. Этот компонент обеспечивает высокую надежность, доступность и масштабируемость с помощью автоматической и прозрачной для пользователей репликации, и поддержки отказоустойчивости. Также поддерживается балансировка нагрузки и синхронизация инкрементальных изменений во всех репликах данных. *Cloud Fabric* отслеживает все конфликты при изменениях/обновлениях данных, используя двунаправленную синхронизацию данных между репликами на основе встроенных или задаваемых пользователями политик.

Библиотека ГГТУ им. П.О. Сухого

Так как *SQL Azure* построена на основе *SQL Server*, пользователи получают знакомую реляционную модель данных, которая практически симметрична с серверами *SQL Server*, развернутыми у заказчиков. Поддерживаются многие возможности ядра *SQL Server*, хотя в текущей реализации облачной базы данных *SQL Azure* существует ряд ограничений.

Можно выделить четыре основных высокоуровневых сценария использования *SQL Azure*:

1. Использование *SQL Azure* приложениями, которым требуется обеспечение совместной работы пользователей, находящихся внутри и вне «границ» организации.
2. Использование *SQL Azure* приложениями, расположенными в инфраструктуре *Windows Azure*.
3. Использование *SQL Azure* как основы для создания средств консолидации данных из различных источников и предоставление этих данных пользователям, разделенным географически и работающим с различными устройствами.
4. Использование *SQL Azure* совместно с web-приложениями с высокой нагрузкой, использующими для хранения данных реляционные структуры.

Все эти сценарии решают наиболее часто возникающие перед разработчиками задачи, связанные с выбором в пользу создания приложений, работающих локально, в облаке или гибридных приложений.

2.2.6. Службы мультимедиа

Windows Azure Media Services – это облачное *PaaS*-решение, которое позволяет эффективно строить медиа-сервисы и доставлять медиа-контент вашим потребителям. Решение предлагает набор готовых к применению сервисов, которые позволяют производить быстрое получение медиа-материала, кодирование, конвертирование формата, хранение, защиту контента и доставку видео как в *live*-формате, так и по требованию. *Windows Azure Media Services* также поддерживают доставку контента на любое устройство или платформу, включая: *HTML5*, *Silverlight*, *Flash*, *Windows* 8, *iPad*, *iPhone*, *Android*, *Xbox* и *Windows Phone*. Кроме того, *Windows Azure Media Services* – это технологии платформы *Microsoft Media Platform*.

Windows Azure Media Services предоставляют следующие сервисы для построения собственных медиа-сервисов и приложений:

– загрузку контента;
– перекодирование;

Библиотека ГГТУ им. П.О. Сухого

– конвертацию форматов;

– защиту контента;

– вещание по запросу и живое вещание, а также аналитику и рекламу.

Этот функционал доступен разработчикам через *REST API*, что позволяет создавать решения на базе *Windows Azure Media Services* с использованием любой привычной, удобной или, например, являющейся стандартом в организации технологии. Разработчикам на платформе *.NET* доступен *Windows Azure Media Services SDK for .NET*, который в удобной форме оборачивает предоставляемый *REST API*.

Службы мультимедиа предоставляют ряд встроенных и готовых к использованию компонентов Майкрософт и сторонних производителей (от отправки мультимедиа до распределения контента), которые можно объединить для выполнения ваших требований. К числу доступных возможностей относится отправка, хранение, кодирование данных, преобразование форматов, защита и доставка контента.

Службы мультимедиа можно вызывать отдельно через стандартные *API REST* для облегчения интеграции с внешними приложениями и службами.

2.3. Развитие платформы

Платформа постоянно обновляется и в ней появляются все новые функции. Среди последних новинок и обновлений, последнее из которых было в октябре 2018 г., можно отметить следующее:

– поддержка новой платформы *.NET* 4.7;

– новый портал управления, который вышел из состояния *Preview*, построенный на основе *HTML*5 вместо *Silverlight* (портал, который был с самого начала появления платформы);

– *App Store* для дополнительных компонентов платформы (*Addons*);

– обновлен *SDK* до версии 1.8, который в свою очередь приносит поддержку *IIS*8, *C*#5.0, преимущества *.NET*4.7, *WebSockets*;

– обновления для мобильных сервисов;

– обновления для web-сайтов;

– обновления для облачных сервисов.

Одним из направлений развития платформы является предоставление мобильных сервисов, позволяющих создавать универсальные приложения в облаке, которые будут работать с мобильными платформами. Например, сервис *Push*-уведомлений, хранение данных

Библиотека ГГТУ им. П.О.Сухого

в облаке и др. Самой важной особенностью является то, что *Azure* по сути на сегодня является больше платформой для разработчиков и компаний, которые создают свои системы в облаках.

2.4. Магазин Windows Azure

Сайт *Windows Azure Marketplace* – это глобальный интернет-магазин приложений *SaaS* и лучших наборов данных. Представив свои приложения для *Windows Azure* на этом сайте, их можно продавать по всему миру. Кроме того, имеется возможность подписаться в своих приложениях на целый ряд лучших наборов данных, представленных на этом сайте, либо распространять собственные наборы данных, извлекая прибыль.

Сценарии использования магазина:

– *получение всемирного охвата*. Возможность распространения приложений во многих странах и прием оплаты в разных валютах.

– *единая модель безопастности*. Возможность использования единой модели безопасности, выставления счетов, аудита и проверки подлинности, поддерживающую возможность единого входа с помощью *OAuth v*2.

– *простое управление предложениями.* Возможность использования отчетов о трафике, подписках и продажах.

Библиотека ГГТУ им. П.О.Сухого

Глава 3. Разработка приложений с Windows Azure Cloud Services

Размещаемые в *Windows Azure* сервисы состоят из одной или более web-ролей (*web-role*) и (или) прикладных ролей (*worker-role*). Web-роль является доступным через конечные точки *HTTP* и *HTTPS* web-приложением *ASP.NET* и предоставляет пользовательский интерфейс. Прикладные роли, как правило, используются для фоновой обработки данных. Сервисы *Windows Azure* могут включать любые комбинации ролей любого типа, каждая из которых может существовать в одном или более экземплярах. Экземпляры ролей могут добавляться и удаляться без перезапуска приложения, что позволяет масштабировать сервис по мере необходимости.

Сервисы хранения данных *Windows Azure* (*Windows Azure storage services*) включают в себя хранилище двоичных объектов (*Blob services*), табличное хранилище (*Table services*) и очереди (*Queue services*), позволяющие организовывать взаимодействие между службами.

Создадим простое приложение *GuestBook*, в котором демонстрируется ряд возможностей платформы *Windows Azure*, в том числе использование web- и прикладных ролей, хранилища двоичных объектов и табличного, а также очередей.

В приложении *GuestBook* web-роль используется для создания пользовательского интерфейса, предоставляющего возможность как просмотра содержимого гостевой книги, так и добавления в нее новых записей. Каждая запись включает имя, текстовое сообщение и изображение. В приложении также используется прикладная роль, генерирующая миниатюры для добавленных пользователями изображений.

Когда пользователь добавляет новую запись, web-роль загружает соответствующее изображение в хранилище двоичных объектов, после чего добавляет в табличное хранилище новую сущность, содержащую введенную пользователем информацию и ссылку на изображение в хранилище двоичных объектов. При обращении веб-роль форматирует данную информацию таким образом, чтобы пользователь мог просмотреть содержимое гостевой книги.

После сохранения изображения и добавления сущности в табличное хранилище web-роль помещает в очередь рабочий элемент, указывающий на необходимость обработки изображения. Прикладная роль извлекает рабочий элемент из очереди, а также изображение из хранилища двоичных объектов и создает миниатюру – уменьшенный вариант оригинального изображения. Использование очередей явля-

Библиотека БГУИР

ется рекомендованным подходом при организации взаимодействия сервисов в облаке. Преимущества организации слабосвязанных элементов заключаются в возможности их раздельного тестирования и масштабирования.

3.1. Создание проекта в Visual Studio

Для создания проекта в *Visual Studio* необходимо выполнить следующие действия:

1. Откройте меню **Пуск | Все программы|** ***Microsoft Visual Studio | Microsoft Visual Studio***.

2. В меню ***File*** выберите ***New*** и затем ***Project.***

3. В диалоговом окне ***New Project*** разверните узел ***Visual C#*** и в списке ***Installed Templates*** выберите ***Cloud***.

4. В списке ***Templates*** (рис. 3.1) выберите ***Windows AzureCloud Service***. Введите ***Name «GuestBook»***, имя *solution* ***«Begin»***. Затем выберите расположение внутри папки ***Ex1-BuildingYourFirstWindowsAzureApp***. Убедитесь, что опция ***Create directory for solution*** выбрана и нажмите ***OK*** чтобы создать проект.

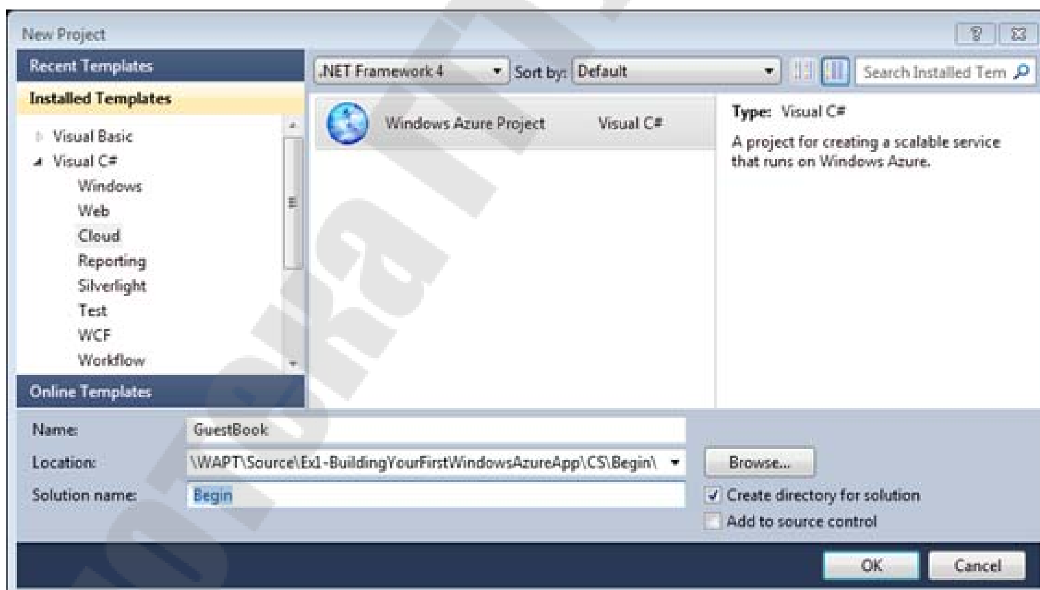


Рис. 3.1. Создание нового проекта*Windows Azure Cloud Service*

5. В диалоге ***New Cloud Service Project*** (рис. 3.2) разверните узел ***Visual C#*** и выберите ***ASP.NET Web Role***. Переместите выбранную роль в проект. Выберите роль в проекте и введите имя ***Guest-Book_WebRole***. Нажмите ***OK*** для завершения.

Библиотека ГГТУ им. П.О. Сухого

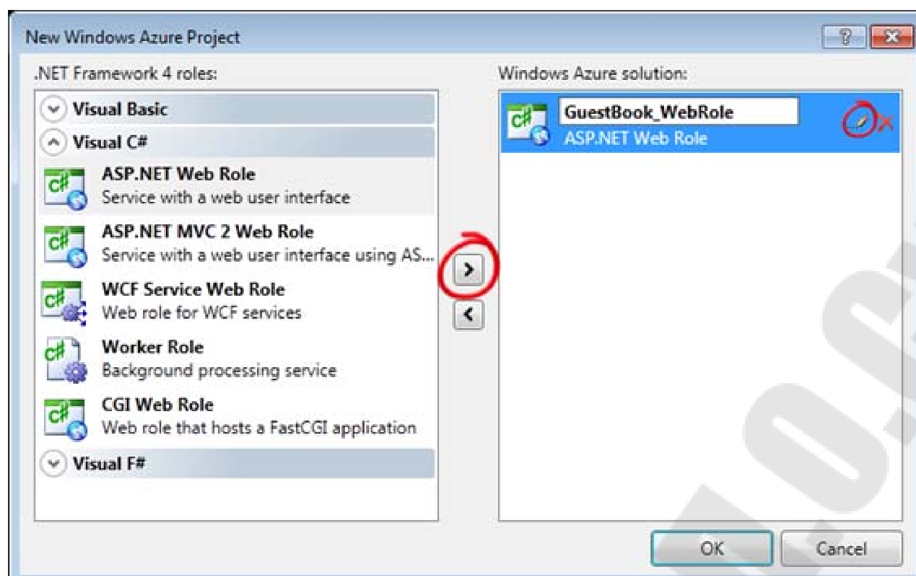


Рис. 3.2. Добавление ролей в проект

6. Обратите внимание на структуру проекта в ***Solution Explorer*** (рис. 3.3).

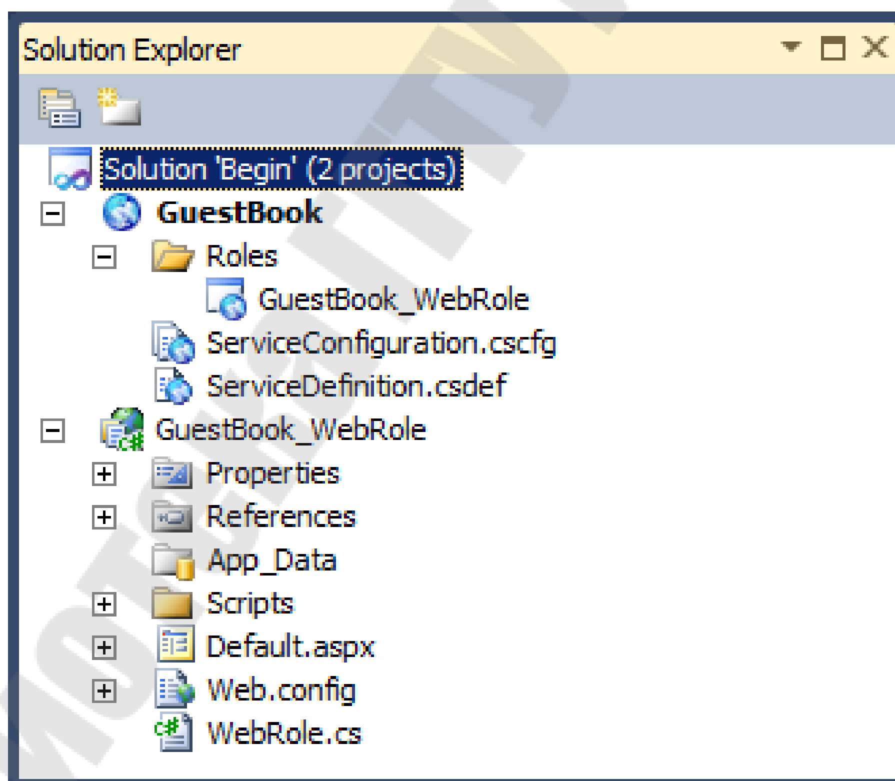


Рис. 3.3. Структура проекта в *Solution Explorer*

Библиотека ГГТУ им. П.О. Сухого

3.2. Создание модели данных для элементов в *Table Storage*

Для создания модели данных элементов в ***Table Storage*** выполните следующие действия:

1. В ***Solution Explorer*** нажмите правой кнопкой мыши по ***Begin***, выберите ***Add*** | ***New Project***.

2. В диалоге ***Add New Project*** (рис. 3.4) разверните узел ***Visual C#*** в списке ***Installed Templates***, выберите категорию ***Windows*** и выделите ***Class Library*** в списке шаблонов. Убедитесь, что выбран. ***NET Framework*** **3.5.** Введите имя ***GuestBook_Data*** и нажмите ***OK***.

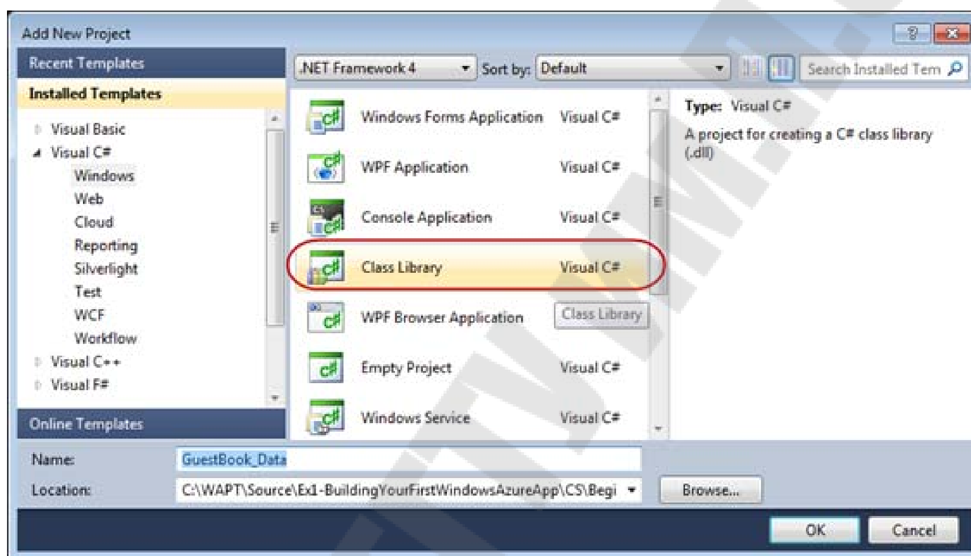


Рис. 3.4. Создание библиотеки классов

3. Удалите файл класса по умолчанию. Нажмите правой кнопкой по ***Class*1*.cs*** и выберите ***Delete***. Нажмите ***OK***.

4. Добавьте ссылку на библиотеку *.NET* для *ADO.NET* в проект ***GuestBook_Data***. В ***Solution Explorer*** нажмите правой кнопкой по проекту ***GuestBook_Data***, выберите ***Add Reference***, затем выберите закладку ***.NET***, выделите компонент ***System.Data.Service.Client*** и нажмите ***OK***.

5. Выполните пункт 4, добавив ссылку на библиотеку ***Microsoft.WindowsAzure.StorageClient*** (рис. 3.5).

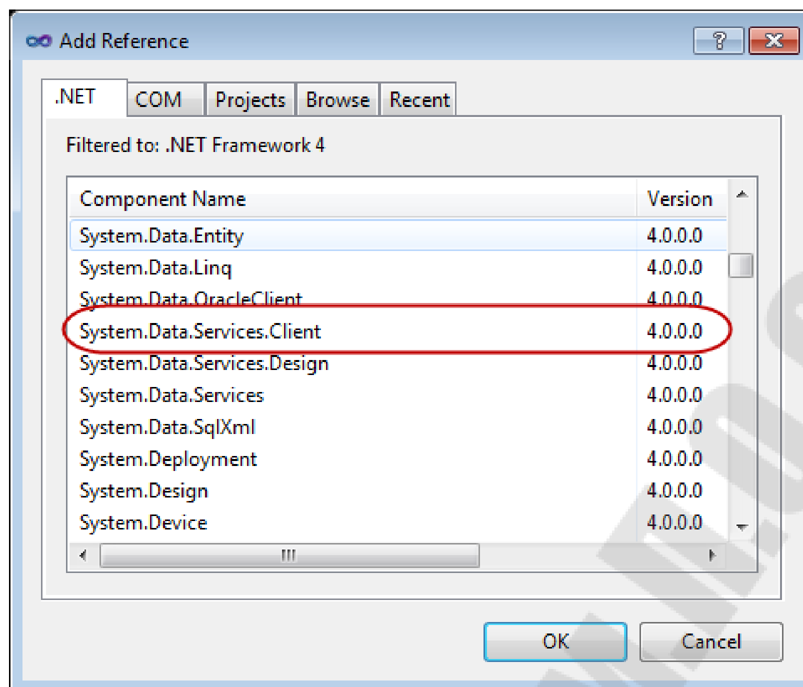


Рис. 3.5. Добавление ссылки на библиотеку

6. Нажмите правой кнопкой мыши по ***GuestBook_Data*** в ***Solution Explorer***, выберите ***Add***, затем ***Class***. В диалоге ***Add New Item*** (рис. 3.6) введите имя ***GuestBookEntry.cs*** и нажмите ***Add***.

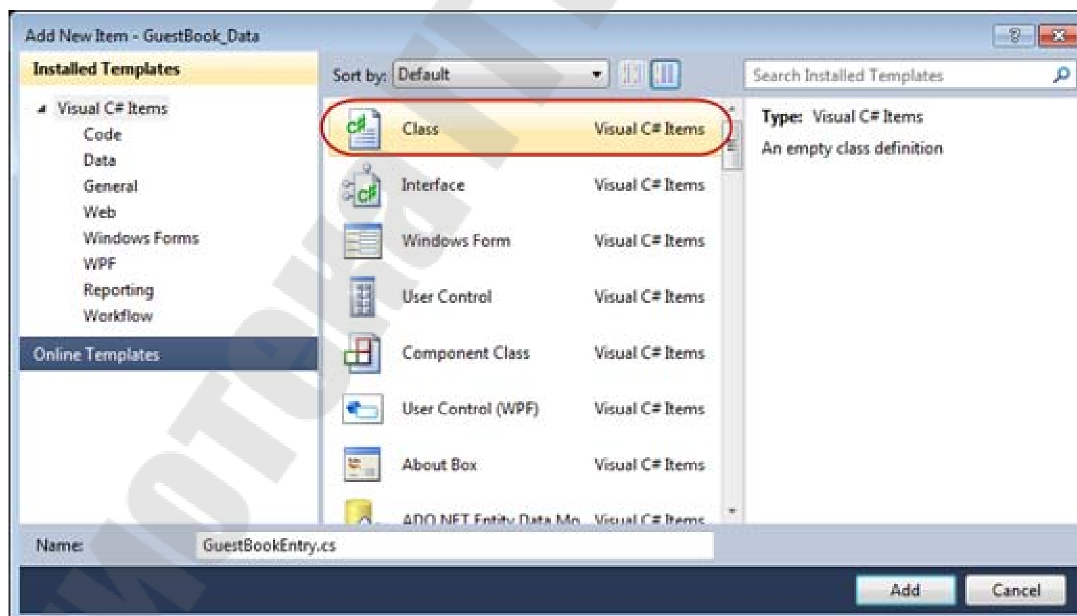


Рис. 3.6. Добавление класса

7. Откройте файл ***GuestBookEntry.cs***, добавьте в начало файла *using Microsoft.WindowsAzure.StorageClient*;

8. Измените объявление класса ***GuestBookEntry***

```
public class GuestBookEntry :
 Microsoft.WindowsAzure.StorageClient.TableServiceEntity
{
}
```

9. Добавьте конструктор по умолчанию:

```
public GuestBookEntry()
{
 PartitionKey = DateTime.UtcNow.ToString("MMddyyyy");
 // Row key allows sorting, so we make sure the rows come back in
//time order.
 RowKey = string.Format("{0:10}_{1}", DateTime.MaxValue.Ticks - DateTime.Now.Ticks, Guid.NewGuid());
}
```

10. Добавьте свойства:

```
public string Message { get; set; }
public string GuestName { get; set; }
public string PhotoUrl { get; set; }
public string ThumbnailUrl { get; set; }
```

11. Сохраните файл ***GuestBookEntry.cs***.

12. Нажмите правой кнопкой мыши по ***GuestBook_Data*** в ***Solution Explorer***, выберите ***Add***, затем ***Class***. В диалоге ***Add New Item*** введите имя ***GuestBookDataContext.cs*** и нажмите ***Add***.

13. Откройте файл ***GuestBookDataContext.cs***, добавьте в начало файла:

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
```

14. Измените объявление класса ***GuestBookDataContext*** и добавьте конструктор:

```
public class GuestBookDataContext : TableServiceContext
{
 public GuestBookDataContext(string baseAddress, StorageCredentials credentials)
  : base(baseAddress, credentials)
 { }
}
```

15. Добавьте свойство:

```
public class GuestBookDataContext : TableServiceContext
{
```


Библиотека ГГТУ им. П.О. Сухого

```
...
public IQueryable<GuestBookEntry> GuestBookEntry
{
get
{
return this.CreateQuery<GuestBookEntry>("GuestBookEntry");
}
}
}
```

16. Нажмите правой кнопкой мыши по ***GuestBook_Data*** в ***Solution Explorer***, выберите ***Add***, затем ***Class***. В диалоге ***Add New Item*** введите имя ***GuestBookDataContext.cs*** и нажмите ***Add***.

17. Откройте файл ***GuestBookEntryDataSource.cs***, добавьте в начало файла:

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.StorageClient;
```

18. Далее измените класс:

```
public class GuestBookEntryDataSource
{
 private static CloudStorageAccount storageAccount;
 private GuestBookDataContext context;
}
```

19. Добавьте конструктор:

```
public class GuestBookEntryDataSource
{
 private static CloudStorageAccount storageAccount;
 private GuestBookDataContext context;
 static GuestBookEntryDataSource()
 {
 storageAccount = CloudStorageAccount.FromConfigurationSetting-
("DataConnectionString");
 CloudTableClient.CreateTablesFromModel(
 typeof(GuestBookDataContext),
 storageAccount.TableEndpoint.AbsoluteUri,
 storageAccount.Credentials);
 }
}
```

20. Добавьте конструктор для класса ***GuestBookData-EntrySource***:

```
public GuestBookEntryDataSource()
```

Библиотека ГГТУ им. П.О. Сухого

```
{
 this.context = new GuestBookDataCotext(storageAccount.Table-Endpoint.-AbsoluteUri, storageAcount.Credentials);
 this.context.RetryPolicy = RetryPolicies.Retry(3, TimeSpan.From-Seconds(1));
}
```

21. Добавьте методы:

```
public IEnumerable<GuestBookEntry> Select()
{
 var results = from g in this.context.GuestBookEntry
 where g.PartitionKey == DateTime.UtcNow.ToString("MMddyyyy")
 select g;
 return results;
}

public void UpdateImageThumbnail(string partitionKey, string rowKey, string thumbUrl)
{
 var results = from g in this.context.GuestBookEntry
 where g.PartitionKey == partitionKey && g.RowKey == rowKey
 select g;

 var entry = results.FirstOrDefault<GuestBookEntry>();
 entry.ThumbnailUrl = thumbUrl;
 this.context.UpdateObject(entry);
 this.context.SaveChanges();
}
```

22. Сохраните файл ***GuestBookEntryDataSource.cs.***

3.3. Создание Web-роли для отображения гостевой книги

Для создания web-роли для отображения гостевой книги выполните следующие действия:

1. В ***Solution Explorer*** нажмите правой кнопкой по проекту ***GuestBook_WebRole***, выберите ***Add Reference***, затем выберите закладку ***.NET***, выделите компонент ***System.Data.Service.Client*** и нажмите ***OK***.

2. В ***Solution Explorer*** нажмите правой кнопкой по проекту ***GuestBook_WebRole***, выберите ***Add Reference*****,** затем выберите закладку ***Project***, выделите ***GuestBook_Data*** и нажмите ***OK***.

Библиотека ГГТУ им. П.О. Сухого

3. Нажмите правой кнопкой по ***Default.aspx*** и выберите ***Delete***. Нажмите ***OK***.

4. В ***Solution Explorer*** нажмите правой кнопкой по проекту ***GuestBook_WebRole***, выберите ***Add***, выделите ***Existing Item*.**

5. В диалоге ***Add Existing Item*** выберите директорию ***Source**Ex*1-*BuildingYourFirstWindowsAzureApp**CS**Assets***, выберите ***Add***.

6. В ***Solution Explorer*** нажмите правой кнопкой по ***Default.aspx***, выберите ***View Code***, объявите следующие пространства имен:

```
using System.IO;
using System.Net;
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
using Microsoft.WindowsAzure.StorageClient;
using GuestBook_Data;
```

7. В классе укажите:

```
private static bool storageInitialized = false;
private static object gate = new Object();
private static CloudBlobClient blobStorage;
```

8. Найдите событие ***SignButton_Click*** и добавьте следующий код:

```
protected void SignButton_Click(object sender, EventArgs e)
{
 if (FileUpload1.HasFile)
 {
 InitializeStorage();

 // upload the image to blob storage
 CloudBlobContainer container = blobStorage.GetContainerReference("guestbookpics");
 string uniqueBlobName = string.Format("image_{0}.jpg", Guid.NewGuid().ToString());
 CloudBlockBlob blob = container.GetBlockBlobReference(uniqueBlobName);
 blob.Properties.ContentType = FileUpload1.PostedFile.ContentType;
 blob.UploadFromStream(FileUpload1.FileContent);
 System.Diagnostics.Trace.TraceInformation("Uploaded image '{0}' to blob storage as '{1}'", FileUpload1.FileName, uniqueBlobName);
 // create a new entry in table storage
 GuestBookEntry entry = new GuestBookEntry() { GuestName =
```

Библиотека ГГТУ им. П.О. Сухого

```
NameTextBox.Text, Message = MessageTextBox.Text, PhotoUrl = blob.-
Uri.ToString(), ThumbnailUrl = blob.Uri.ToString() };
        GuestBookEntryDataSource ds = new GuestBookEntryDataSource();
        ds.AddGuestBookEntry(entry);
        System.Diagnostics.Trace.TraceInformation("Added entry {0}-{1} in
table storage for guest '{2}'", entry.PartitionKey, entry.RowKey, en-
try.GuestName);
        }
        NameTextBox.Text = "";
        MessageTextBox.Text = "";
        DataList1.DataBind();
       }
```

9. Обновите метод ***Timer*1*_Tick***:

```
protected void Timer1_Tick(object sender, EventArgs e)
{
 DataList1.DataBind();
}
```

10. Обновите событие ***Page_Load***:

```
protected void Page_Load(object sender, EventArgs e)
{
 if (!Page.IsPostBack)
 {
 Timer1.Enabled = true;
 }
}
```

11. Произведите изменения в методе ***InitializeStorage***:

```
private void InitializeStorage()
{
 if (storageInitialized)
 {
 return;
 }
 lock (gate)
 {
 if (storageInitialized)
 {
 return;
 }
 try
```


Библиотека ГГТУ им. П.О.Сухого

```
{
// read account configuration settings
var storageAccount = CloudStorageAccount.FromConfiguration-Setting("DataConnectionString");

// create blob container for images
blobStorage = storageAccount.CreateCloudBlobClient();
CloudBlobContainer container = blobStorage.GetContainerRefe-rence("guestbookpics");
container.CreateIfNotExist();

// configure container for public access
var permissions = container.GetPermissions();
permissions.PublicAccess = BlobContainerPublicAccessType.Con-tainer;
container.SetPermissions(permissions);
}
catch (WebException)
{
throw new WebException("Storage services initialization failure. " + "Check your storage account configuration settings. If running locally, "+ "ensure that the Development Storage service is running.");
}
storageInitialized = true;
}
}
```

12. В ***Solution Explorer*** разверните узел ***Roles*** в проекте ***GuestBook***. Нажмите два раза по ***GuestBook_WebRole***, откроется свойства данной роли, выберите закладку ***Setting***. Нажмите ***Add Setting*** (рис. 3.7), наберите "*DataConnectionString*" в колонке ***Name***, измените ***Type*** не *ConnectionString* и нажмите ***Add Setting***.

Библиотека ГГТУ им.П.О.Сухого

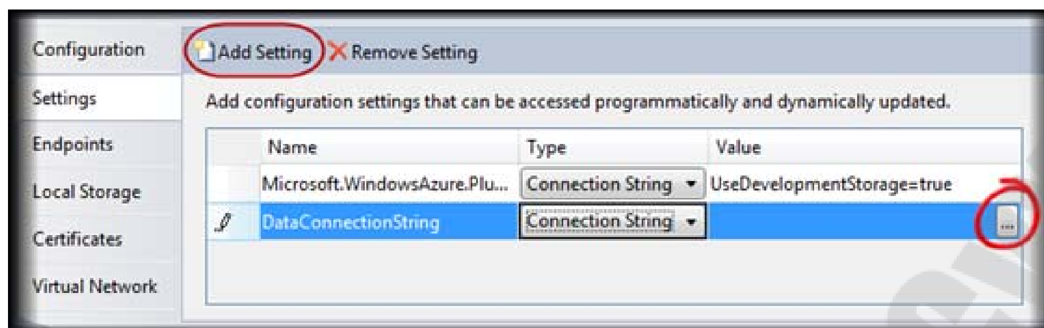


Рис. 3.7. Добавление строки подключения

13. В диалоге ***Storage Connection String*** выберите ***Use development storage*** и нажмите ***OK*** (рис. 3.8).

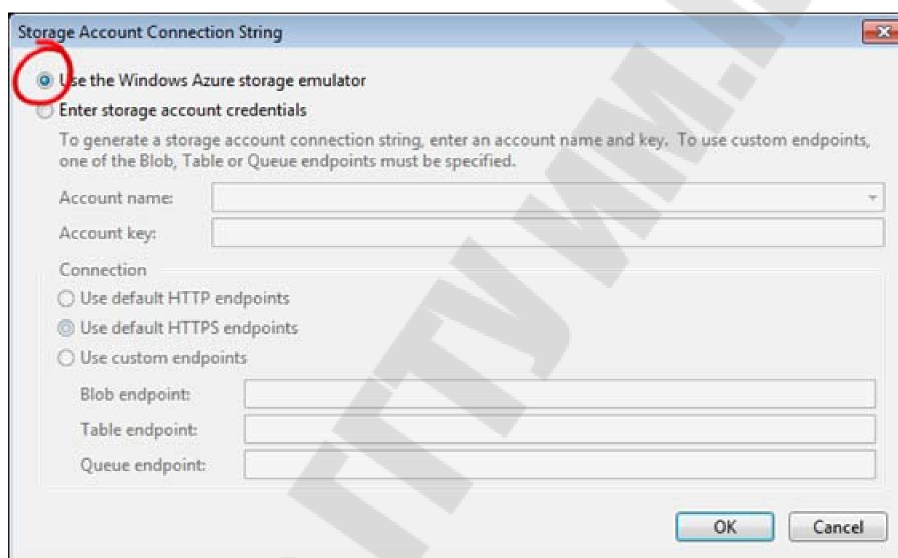


Рис. 3.8. Настройка эмулятора локального хранилища

14. Сохраните изменения.

15. В проекте ***GuestBook_WebRole*** откройте файл *Global.asax.cs*.

16. Объявите пространства имен:

```
using Microsoft.WindowsAzure;
using Microsoft.WindowsAzure.ServiceRuntime;
```

17. Вставьте следующий код внутрь метода ***Application_Start***, заменим содержимое по умолчанию:

```
void Application_Start(object sender, EventArgs e)
{
Microsoft.WindowsAzure.CloudStorageAccount.SetConfiguration-
SettingPublisher((configName, configSetter) =>
```


Библиотека ГГТУ им. П.О. Сухого

```
{
configSetter(RoleEnvironment.GetConfigurationSettingValue(config-
Name));
});
}
```

3.4. Организация очереди рабочих элементов

Для организации очереди рабочих элементов выполните следующие действия:

1. В ***Solution Explorer*** нажмите правой кнопкой по ***Default.aspx***, выберите ***View Code*,** объявите элемент клиента очереди:

```
public partial class _Default : System.Web.UI.Page
{
 private static bool storageInitialized = false;
 private static object gate = new Object();
 private static CloudBlobClient blobStorage;
 private static CloudQueueClient queueStorage;
 ...
}
```

2. Найдите метод ***InitializeStorage*** и вставьте следующий код внутрь данного метода:

```
public partial class Default : System.Web.UI.Page
{
 ...
 private void InitializeStorage()
 {
 ...
 try
 {
 ...
 // configure container for public access
 var permissions = container.GetPermissions();
 permissions.PublicAccess = BlobContainerPublicAccessType.Con-
tainer;
 container.SetPermissions(permissions);
 // create queue to communicate with worker role
 queueStorage = storageAccount.CreateCloudQueueClient();
 CloudQueue queue = queueStorage.GetQueueReference("guestthumbs");
```


Библиотека ГГТУ им.П.О.Сухого

```
queue.CreateIfNotExist();
}
catch (WebException)
{
...
}
```

Проверка

1. Нажмите ***F5*** для запуска сервиса. Сервис запустится в *development fabric*. Для открытия пользовательского интерфейса необходимо нажать правой кнопкой мыши на значке в области уведомления панели задач и выбрать ***Show Development Fabric UI***.

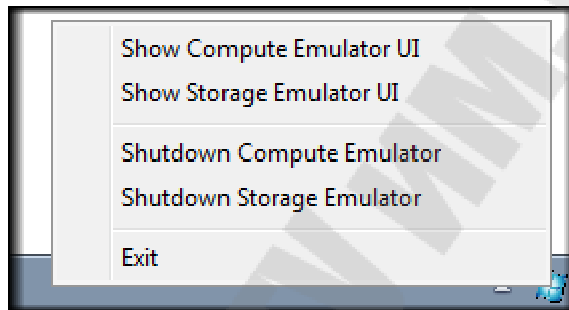


Рис. 3.9. Запуск сервиса

2. Переключитесь на *Internet Explorer* для просмотра приложения *GuestBook.*

3. Добавьте новую запись в гостевой книг (рис. 3.10).

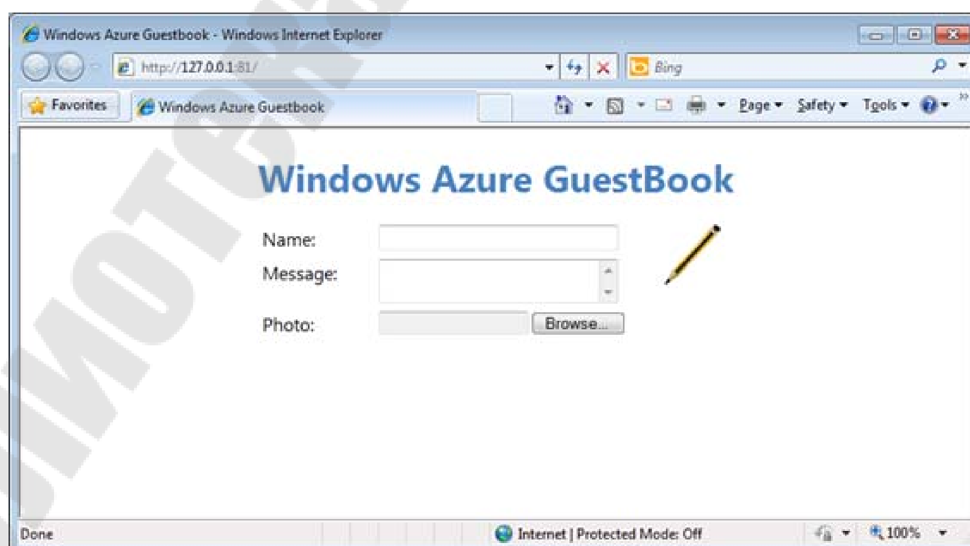


Рис. 3.10. Главная форма разработанного приложения

Библиотека ГГТУ им. П.О. Сухого

Глава 4. Авторизация и безопасность с Windows Azure Active Directory

4.1. Аутентификация на базе утверждений

Сервис *Windows Azure Active Directory* предлагает возможность управления процессом аутентификации, доступом к приложениям с помощью аутентификации на основе утверждений. С помощью этого сервиса можно обеспечить единый вход (*Single SignOn*), повышенную безопасность и простое взаимодействие с уже развернутыми в *Active Directory* приложениями, а также выполнить интеграцию приложения с другими популярными провайдерами аутентификации (*Microsoft Account*, *Google*, *Facebook* и т. д.). *WindowsAzure Active Directory* – это полноценная реализация каталога в облаке. При этом сервис поддерживает популярные открытые стандарты обеспечения федеративной аутентификации: *SAML* 2.0, *OData*, *WS-FED*, *OAuth* 2.0/*OpenID*, *JSON Web Token* (*JWT*), *SWT*. Авторизация происходит иначе, *Windows Azure Active Directory* предоставляет возможность аутентификации. Авторизация в этом случае подразумевает под собой определение прав объекта, аутентификация – определение, имеет ли объект право на вход в систему.

Методы обеспечения аутентификации на основе утверждений позволяют реализовать различные сценарии доверия между несколькими системами для осуществления обмена информацией, в том числе конфиденциальной, например, *Email*-адресами или информацией о заказах, используя для этого набор стандартов *WS*-*, описывающих токен *Security Assertion MarkupLanguage*.

Аутентификация на основе утверждений предлагает упрощение процесса аутентификации за счет использования привычных пользователю провайдеров идентификации, таких как *Windows Live ID*, *Facebook* и локальный доменный каталог *Active Directory*.

Любая система, использующая аутентификацию на основе утверждений, пользуется определенным набором терминов. Утверждение является некоторой информацией об объекте, которую предоставляет провайдер идентификации. Например, есть некоторое предложение, что «Уважаемая компания А, мое имя – Александр, фамилия – Иванов, и это подтверждается паспортом, который был выдан компанией Б». В данном случае утверждением будет являться набор данных Имя–Фамилия. Вторая часть о паспорте – это токен

Библиотека ГГТУ им. П.О. Сухого

безопасности (*Security token*), объект с электронной подписью, который и содержит утверждение (-я). Третьей частью является провайдер идентификации (*Security Token Service*) – доверенная компания/объект, который имеет право выпускать токены безопасности, содержащие утверждения. «Компания А» – это объект, которому требуется получить токен безопасности для предоставления входа в систему либо доступа к защищенной функциональности системы (*Relying Party*).

В отдельных сценариях может также присутствовать провайдер федерации (*Federation Provider*) – провайдер идентификации и брокер между приложением и личностью в провайдере идентификации (*Facebook*, *Live ID* и т. д.). Разработчик настраивает аутентификацию таким образом, что конечное приложение, которым пользуется пользователь, доверяет только провайдеру федерации (в контексте отношений приложение-провайдер федерации является провайдером идентификации), а провайдер федерации доверяет набору провайдеров идентификации, также настроенному разработчиком, и в контексте данных отношений провайдер федерации является приложением *relying party*, получающим токен безопасности от провайдеров идентификации. Таким образом, конечное приложение знает только о провайдере федерации, который скрывает от него детали реализации аутентификации с использованием реальных провайдеров идентификации.

4.2. Федеративная аутентификация в Windows Azure

Рассмотрим использование федеративной аутентификации в *Windows Azure* с использованием публичных провайдеров идентификации и доменного каталога *Active Directory*.

Для того чтобы лучше понять федеративную аутентификацию с использованием *Windows Azure*, разберем пример с уже настроенной инфраструктурой. В инфраструктуре установлены: домен уровня *Windows Server* 2008 *R*2, *AD FS* 2.0, *WIF*, *WIF SDK* 4.0, *Windows Azure Access Control Service*, приложение в облаке *Windows Azure* с адресом *https*://*techdaysdemo.cloudapp.net*.

В данном сценарии возможны две ситуации: пользователь входит на свой компьютер, находящийся в домене, и вводит свои учетные данные либо заходит с компьютера, не находящегося в домене, и таким образом должен ввести свои доменные учетные данные. Различаются ситуации тем, на каком этапе в процесс аутентификации подключается *Active Directory*.

Библиотека ГГТУ им. П.О.Сухого

Придя на работу, пользователь ввел свои учетные данные на компьютере в домене. В этот момент происходит взаимодействие с контроллером домена *Active Directory* – проверяются учетные данные, и контроллер домена отправляет пользователю два сообщения – *Client*/*TGS Session Key*, зашифрованный секретным ключом пользователя, и *Ticket Granting Ticket* (который включает в себя служебные данные – сетевой адрес, период жизни тикета и т. д.), зашифрованный секретным ключом *Ticket Granting Service* (*TGS*, сервиса выдачи тикетов). Пользователь, получив оба сообщения, расшифровывает первое сообщение секретным ключом, сгенерированным из пароля, и получает *Clients*/*TGS Sesion Key*, который используется для дальнейших коммуникаций с *TGS*. Второе сообщение он прочитать не может, так как не знает ключа *TGS*. Если в процессе аутентификации не произошло ошибок, то пользователь становится аутентифицирован в домене.

Через несколько часов пользователь запускает браузер и переходит на *https*://*techdaysdemo*.*cloudapp*.*net*.

Приложение настроено для *Windows Azure Access Control Service* как *Relying Party*. Для приложения *Windows Azure Access Control Service* является провайдером идентификации. Однако *Windows Azure Access Control Service* не является провайдером идентификации по своей сути, – для него настроены доверенные отношения с сервером *AD FS* 2.0, расположенным в организации пользователя. *Windows Azure Access ControlService* выступает для *AD FS* 2.0 как *Relying Party*, таким образом *AD FS* выступает в роли провайдера идентификации.

Теперь, когда пользователь заходит на сайт *https*://*techdaysdemo*.*cloudapp*.*net*, его автоматически перенаправляют сначала на *WindowsAzure Access Control Service*, который запускает процесс *Home Realm Discovery*. У *Windows Azure Access Control Service* есть список *HomeRealm*, с которыми у него есть доверенные отношения. Определяющим для *Home Realm Discovery* является аргумент, который предоставляется пользователем в *URL*.

Строка запроса при редиректе (302) на сервис *Windows Azure Access Control Service* выглядит так: *https*://*ahrimanacs*.*accesscontrol*.*windows*.*net*/*v*2/*wsfederation*?*wa*=*wsignin*1.0&*wtrealm*=*https*%3*a*%2*f*%2*ftechdaysdemo*.*cloudapp*.*net*%2*f*&*wctx*=*rm*%3*d*0%26*id*%3*dpassive*%26*ru*%3*d*%252*f*&*wct*=2012-05-07*T*11%3*a*06%3*a*15*Z GET* 302.

Где ***wa*** – описание действия (*signin* – либо вход, либо *signout* – выход), ***wtrealm*** – *URL* доверенного *Realm*, куда направляется токен-приложение, ***wctx*** – контекст, передающийся странице (в данном слу-

чае контекст является пассивным). В данном запросе отсутствует дополнительное значение ***wreply***, содержащее указание на то, куда пользователь должен быть перенаправлен с полученным *RSTR*. Часть ссылки до *wsfederation* – точка вхождения для пассивного *STS WS-Federation* в *Windows Azure Access Control Service*.

Далее *Windows Azure Access Control Service* перенаправляет пользователя на сервер *AD FS* 2.0.

На сервер *Active Directory* отправляется выданный пользователю *TGT*, сервер создает *Service Ticket* и передает пользователю. Этот тикет используется в коммуникациях с сервером *AD FS* 2.0. Сервер *AD FS* 2.0 определяет, что пользователь аутентифицирован, и обращается к серверу *Active Directory* за атрибутами для этого пользователя.

На сервере *AD FS* 2.0 происходит создание *SAML*-токена с полученными атрибутами (утверждениями) и его подписание. *SAML*-токен выдается web-браузеру пользователя для передачи его *Windows Azure Access Control Service*. Теперь *Windows Azure Access Control Service* может определить, что подпись верна, и создать собственный *SAML*-токен путем либо копирования из первичного *SAML*-токена атрибутов, либо с использованием механизма под названием *Rules Engine*, который копирует атрибуты, но позволяет производить над ними манипуляции. *Windows Azure Access ControlService* пересылает свой *SAML*-токен web-браузеру, и только после этого web-браузер попадает снова в web-приложение (с *SAML*-токеном).

Рис. 4.1. Федеративная аутентификация в *Windows Azure* с использованием публичных провайдеров идентификации и доменного каталога *Active Directory*

На уровне приложения используется *WIF HTTP* модуль web-***WSFederationAuthentificationModule*** (***FAM***), который перехватывает *POST*-запрос от web-браузера с токеном и производит проверку этого токена, проверяя список слушателей и дату истечения токена. Список слушателей – *Audience Restriction*, определяется в *web.config* и содержит список *URL*, который может получать приложение (рис. 4.1).

4.3. Программная реализация проверки токенов безопасности на стороне клиента

Проверкой токенов на уровне приложения занимается набор классов *Windows Identity Foundation* (*WIF*), который позволяет создавать приложения, работающие на основе утверждений, и обрабатывать соответствующим образом запросы *WS-Trust*, *WS-Security* и *WS-Federation*, основанный на первых двух.

WIF позволяет рассматривать аутентификацию на основе утверждений как аутентификацию на основе форм – при попытке аутентификации *WIF* перенаправляет запрос на страницу, предоставленную *STS*, где пользователь аутентифицируется и возвращается уже вошедшим в систему к веб-приложению.

Проверка токенов безопасности происходит с использованием специального *WIF HTTP*-модуля *WSFederationAuthentificationModule* (*FAM*). Модуль перехватывает пришедший в приложение *POST*-запрос, прослушивая событие *AuthenticateRequest*, и, перехватив запрос, начинает проверку токена – периода действия, списка слушателей и целостности токена (используя публичный ключ *STS* для проверки того, является ли доверенным *STS* и не был ли изменен в процессе передачи токен). Модуль разбирает утверждения в токене и использует *HttpContext.User.Identity* для того, чтобы представить *IClaimsPrincipal*, после чего выдает *cookie* для начала сессии. В течение сессии этот процесс не повторяется.

Процесс проверки и разбора токена безопасности на утверждения состоит из следующих этапов:

1. Определение обработчика для токена согласно его типу – *SAML* 1.1, *SAML* 2.0 и т. д.

2. Создание объекта типа *ClaimsPrincipal* с утверждениями.

3. При необходимости изменения утверждений производится вызов *ClaimsAuthentificationManager*.

Библиотека ГГТУ им. П.О. Сухого

4. Создание объекта *SessionSecurityToken*.

5. Создание сессии. В процессе утверждения в виде *ClaimsPrincipal* транслируются в коллекцию *cookie FedAuth[x]*, при этом *cookie* разбиваются на части с целью не превышать определенных лимитов.

6. Перенаправление на *URL*, если он указан.

После завершения всех процедур по проверке и обработке утверждений и токена и перенаправления токена на web-приложение запускается модуль *SessionAuthentificationModule*, который перехватывает коллекцию *cookie* и преобразовывает их в объект *ClaimsPrincipal*. Процесс состоит из нескольких этапов:

1. Проверка наличия *cookie*. Если коллекция обнаружена, она используется для создания *SessionSecurityToken*. При этом коллекция расшифровывается и распаковывается.

2. Проверка периода действия токена.

3. Создание объекта *ClaimsPrincipal* с утверждениями.

4. Определение контекста *HttpContext.User* как *ClaimsPrincipal*.

4.4. Программная модель Windows Identity Foundation

IClaimsPrincipal

В модели утверждений несколько пользователей или каких-либо объектов, обладающих утверждениями, могут быть объединены в единую сущность. Интерфейс *IClaimsPrincipalInterface* определяет данные и поведение личностей одного аутентифицированного объекта в рамках текущего контекста выполнения.

IClaimsPrincipal предоставляет метод *IsInRole*, возвращающий булевый тип в зависимости от того, есть ли в утверждениях объекта утверждение типа *Role* и является ли значение этого утверждения равным чему-то (например, *Domain Admins* в контексте *Windows*-домена). Также интерфейс предоставляет коллекцию личностей, каждая из которых реализует *IClaimsIdentity*. В общем случае в контексте выполнения находится один издатель и один токен, полученный от него, и коллекция *Identity* содержит только один элемент.

ClaimsIdentity

Интерфейс *ClaimsIdentity* определяет базовую функциональность объекта *ClaimsIdentity* и рекомендуется для использования и обеспечения доступа к методам и свойствам *ClaimsIdentity* вместо использования непосредственно *ClaimsIdentity*. Все объекты типа

Библиотека ГГТУ им.П.О.Сухого

ClaimsIdentity реализуют интерфейс *IClaimsIdentity*, который расширяет стандартный *IIdentity*, оставляя при этом базовую функциональность *IIdentity*. *IClaimsIdentity* содержит коллекцию *Claims* – утверждений, привязанных к личности (например, имя или *e-mail*).

Claim

Claim определяет свойство аутентифицированного объекта, которое было выпущено провайдером идентификации – например, членство в определенной группе, возраст и т. д. Содержит поля *Claim*.*ClaimType* (строка, обычно *URI* на описание типа утверждения. например, *http*://*schemas*.*xmlsoap*.*org*/*ws*/2005/05/*identity*/*claims*/*name* – тип, описывающий значение имени) и *Claim*.*ValueType* (и *Claim*. *Value*), содержащие тип значения и значение соответственно.

ClaimsPrincipal

ClaimsPrincipal предоставляет статическое свойство *Current*, которое является *IClaimsPrincipal*, ассоциированным с текущим контекстом выполнения.

4.5. Windows Azure Access Control Service

Одним из главных преимуществ аутентификации на основе утверждений является возможность децентрализации – вместо того, чтобы возлагать на *STS* обязанность аутентифицировать удаленных пользователей (в компании Б), настраиваются отношения доверия с *STS* в компании Б, что будет означать, что *STS* будет доверять *STS* компании Б в вопросах аутентификации в *Realm* компании Б. Таким образом можно обойтись без дополнительных учетных данных для удаленных пользователей – они продолжат пользоваться учетными данными своего домена. Преимущества подобного подхода очевидны – архитектура не претерпевает принципиальных изменений, специалистам, занимающимся *STS* в компании А, нет необходимости отслеживать активность учетных данных, для новых *Realm*-ов просто настраиваются отношения доверия с соответствующими *STS* (в терминологии федеративной аутентификации настройка доверия заключается в обмене специального формата генерируемыми *XML*-файлами).

Таким образом, ***Windows Azure Access Control Service*** предоставляет сервис для обеспечения федеративной безопасности и контроля доступа к облачным или локальным приложениям. ***Windows Azure Access Control Service*** предоставляет возможность создания об-

Библиотека ГГТУ им. П.О.Сухого

лачных и локальных приложений, работающих с множеством провайдеров идентификации, с использованием открытых стандартов и протоколов (*WS-Federation*, *WS-Trust*, *SAML*, *OAuth* 2.0, *SWT*, *JWT*).

Windows Azure Access Control Service состоит из следующих компонентов:

1. ***Security Token Service***. Набор конечных точек, которые могут выдавать токены для *RP*-приложений, реализовывая федеративную аутентификацию для этих приложений. Поддерживаются самые популярные протоколы и платформы – *.NET Framework*, *WCF*, *Silverlight*, *ASP.NET*, *Java*, *Python*, *Ruby*, *PHP*, *Flash*, *OAth WRAP*, *OAuth* 2.0, *WS-Trust*, *WS-Federation*.

Адреса конечных точек для *Windows Azure Access Control Service STS* можно получить с портала управления *Windows Azure Access Control Service* – для точки *WS-Federation Metadata*, которая используется для интеграции web-приложений с *Windows Azure Access Control Service* (эти метаданные могут быть использованы приложением с *WIF*) – и для точки *Windows Azure Access Control Service Management Service*, которая используется для программного управления *Windows Azure Access Control Service*.

2. **Портал управления *Windows Azure Access Control***. Портал, с которого происходит управление пространством имен *Windows Azure Access Control Service*. Предоставляет основную функциональность.

3. **Сервис управления (*Management Service*)**. Сервис для программного управления *Windows Azure Access Control Service* с использованием протокола *OData*.

4. **Движок трансформации токена по правилам (*Token Transformation Rule Engine*)**. В своей работе *Windows Azure Access Control Service* может манипулировать состоянием утверждений. Основная задача – получить утверждения во входящем токене, обработать их согласно настроенным правилам, выпустить выходящие утверждения и инкапсулировать их в токен для *RP*-приложения.

5. **Провайдеры идентификации**. Токены безопасности для *Windows Azure Access Control Service* предоставляются заранее настроенным набором провайдеров идентификации, которые могут быть как публичными, так и локальными для организации (с использованием *AD FS*).

Windows Azure Access Control Service создает список преднастроенных провайдеров идентификации, из которого разработчик вы-

Библиотека ГГТУ им. П.О.Сухого

бирает те, через которые хочет аутентифицироваться, и может создать собственную страницу с этим список и список правил, по которым *Windows Azure Access Control Service* будет обрабатывать утверждения провайдера идентификации.

При первом запросе на приложения определяется, что пользователь не аутентифицирован, и его браузер перенаправляется на *WindowsAzure Access Control Service*, который создает и отдает браузеру список доступных провайдеров идентификации, который может содержать любых провайдеров идентификации – от публичных до локальных каталогов *Active Directory*, предоставляющих данные через *AD FS* 2.0. Пользователь выбирает необходимого провайдера идентификации, после чего перенаправляется на страницу входа в систему соответствующего провайдера идентификации. После входа в систему пользователя провайдер идентификации возвращает токен *Windows Azure Access Control Service* с утверждением о том, что пользователь аутентифицирован. *Windows Azure Access ControlService* выпускает утверждения на основе того, что передал ему провайдер идентификации и токен безопасности, после чего перенаправляет пользователя на приложение. Приложение использует полученный от *Windows Azure Access Control Service* токен безопасности для определения набора привилегий пользователя. Таким образом, в реальном режиме меняется приоритет участников – сначала провайдер идентификации (*Live ID*) определяет то, что пользователь имеет право на вход в систему, и возвращает утверждения, после этого приоритет переходит к *Windows Azure Access Control Service*, и браузер пользователя «предъявляет» токен безопасности *Live ID*, и *Windows Azure Access Control Service* создает новый токен безопасности на основе этих же утверждений (режим *copy claim*), при этом *Windows Azure Access Control Service* может в процессе произвести некоторые преобразования утверждений, в том числе, например, добавив некоторые сведения о принадлежности пользователя к определенной группе.

4.6. Active Directory Federation Services 2.0

Центральное место в *ADFS* 2.0 занимает сервис токенов безопасности (*Security Token Service*, *STS*), который использует *Active Directory* как хранилище, из которого получаются учетные данные пользователей, *Lightweight Directory Access Protocol* (*LDAP*), *SQL* или собственный источник данных как хранилище атрибутов. *STS* в *ADFS* 2.0 может выда-

Библиотека ГГТУ им. П.О.Сухого

вать маркеры защиты по различным протоколам, в том числе *WS-Trust*, *WS-Federation* и *Security Assertion Markup Language* (*SAML*) 2.0. *ADFS* 2.0 *STS* также поддерживает форматы маркеров *SAML* 1.1 и *SAML* 2.0.

ADFS 2.0 может применяться в нескольких распространенных случаях, например, использование *ADFS* 2.0 в качестве провайдера идентификации. *AD FS* прозрачно интегрируется с платформой *Windows Azure*, позволяя аутентифицировать внутренних пользователей любым набором их учетных данных. С помощью *AD FS* можно реализовать принцип *Single Sign-On* как с собственным приложением, размещенным в облаке, так и интегрировав его с другими продуктами *Microsoft* – *Office* 365 и *Sharepoint*. *AD FS* использует аутентификацию на основе утверждений, позволяя манипулировать ими – принимать на вход, например, пользователя с ролью администратор (согласно ролевой модели в *Active Directory*) и выдавая на выход в токен утверждение о том, что это – пользователь с уровнем доступа, например, *Privileged*. Лицензия на *AD FS* 2.0 автоматически предоставляется владельцу лицензии на *Windows Server*2008 *R*2 *Enterprise Edition*.

Принципиально *AD FS* не отличается от обсужденных выше провайдеров идентификации, добавляя лишь дополнительную инфраструктурную и программную гибкость. Например, если сервер *AD FS* 2.0 расположен во внутренней сети и не может получать запросы извне, можно расположить в демилитаризованной зоне сервер *Feder-ation Service Proxy*, который будет маршрутизировать запросы на утверждения на внутренний *AD FS* сервер (серверы) либо другие провайдеры идентификации.

4.7. Многофакторная проверка подлинности Windows Azure

Летом 2013 г. в систему безопасности *Windows Azure* была внедрена новая опциональная функциональность – многофакторная проверка подлинности. Данная функциональность предназначена для защиты учетных записей и облачных сервисов *Microsoft*, решений сторонних компаний или приложений и сервисов, которые используют в качестве системы аутентификации сервис *Windows Azure ActiveDirec tory*. Многофакторная проверка подлинности *Windows Azure* предоставляет дополнительный слой проверки подлинности в дополнение к учетным данным пользователя. При этом многофакторную проверку

Библиотека ГГТУ им. П.О.Сухого

подлинности можно использовать для защиты доступа как к расположенным *on-premise*, так и облачным приложениям. К возможным опциям многофакторной проверки подлинности относятся мобильные приложения, телефонные звонки и текстовые сообщения, при этом пользователи могут выбрать то, что наиболее удобно для них. В контексте защиты локальных приложений многофакторную проверку подлинности можно использовать для защиты *VPN* удаленного доступа и *Web*-приложений с помощью специального *SDK*. Если облачное приложение использует *Active Directory Federation Services*, то разработчик может настроить синхронизацию с *Windows Server Active Directory* или другим каталогом *LDAP*. Что касается других сервисов *Microsoft*, то многофакторная проверка подлинности будет полезна для защиты доступа к *Windows Azure*, *Microsoft Online Services*, *Office* 365 и *Dynamics CRM Online*.

Одной из наиболее сложных задач при разработке новых приложений или создании инфраструктур является, безусловно, обеспечение безопасности. Многослойная безопасность позволяет с определенной степенью уверенности гарантировать, что система не будет взломана с помощью стандартных средств или проблем с кодом. Миграция в облако позволяет решить многие вопросы – многослойная безопасность, которая доступна по умолчанию на платформе, ограничивает возможность внешних злоумышленников к осуществлению, например, *DDOS*-атак, внутренняя архитектура виртуальных машин гарантирует, что даже если соседняя виртуальная машина была скомпрометирована, она не сможет повлиять на другие и будет изолирована.

Программные средства обеспечения аутентификации и авторизации, такие как *Windows Identity Foundation*, *Active Directory*, *ActiveDirectory Federation Services*, *Windows Azure Active Directory* и *MFA*, обеспечивают еще один слой безопасности, которым может воспользоваться разработчик для предоставления опций пользователю.

Библиотека ГГТУ им. П.О.Сухого

Глава 5. Хранение и обработка данных с Windows Azure Storage и Windows Azure SQL Databases

Сервис платформы *Windows Azure Storage* предоставляет масштабируемое хранилище, доступно как сервис *REST* – таким образом взаимодействие с сущностями, хранящимися в *Windows Azure Storage*, может быть произведено с любой платформы, которая поддерживает *HTTP*, что обеспечивает кроссплатформенность и универсальность.

Сервис *Windows Azure Storage* может выступать как альтернативой, так и дополняющим решением для *SQL Azure Databases*, масштабируемой «облачной» версией *SQL Server*. Каждая сущность в *Windows Azure Storage* хранится в трех экземплярах: сущности, хранящиеся в сервисах таблиц и блобов, записываются в еще один триплет в другом датацентре в этом же регионе. Все сущности трех сервисов хранятся в едином контейнере первого уровня, который называется **аккаунтом хранилища**. На одну подписку *Windows Azure* можно создать до **20** аккаунтов, каждый из которых может содержать до 200 ТБ данных. Максимальное количество аккаунтов может быть расширено обращением в техническую поддержку на портале управления *Windows Azure*. Необходимо отметить, что подписка *Windows Azure* имеет собственное ограничение в 10 петабайт данных (200 ТБ – максимальное количество аккаунтов на подписку, равное 50).

Каждый аккаунт хранилища защищается от несанкционированного доступа двумя (первичным и резервным) 512-битными ключами доступа, с помощью которых сервис аутентифицирует входящие запросы. Ключи могут быть регенерированы в любой момент по запросу пользователя.

При хранении данных *Windows Azure Storage* использует одну из трех абстракций:

1. **Блобы (*Blobs*, *Binary Large Objects*)** – простые именованные файлы + метаданные. В блобах могут храниться любые бинарные данные: изображения, текстовые файлы и т. д.
2. **Таблицы** – структурированное хранилище.
3. **Очереди** – сервис-брокер, предоставляющий надежное хранение и доставку сообщений для приложения.

В *Windows Azure Storage* есть еще одна абстракция, которая является надстройкой над хранилищем блобов – диски – долговечные тома *NTFS*, используемые приложениями *Windows Azure*. Диски хранятся в блобах.

Библиотека ГГТУ им. П.О.Сухого

Программная модель *Windows Azure Storage* состоит из следующих классов:

– *CloudStorageAccount*: класс, ассоциирующийся с аккаунтом хранилища, является точкой вхождения в сервисы хранилища.

– *Cloud[Service]Client*: класс, содержащий функциональность, относящуюся к одному из трех сервисов: *CloudTableClient*, *CloudQueueClient*, *CloudBlobClient*.

– *Cloud[Object]*: класс, отображающий конкретную сущность одного из трех сервисов, например, *CloudBlob*, *CloudQueue*.

Сервис блобов также имеет дополнительный класс *CloudBlobContainer*, отображающий контейнер блобов, в котором находятся сами блобы.

5.1. Блобы

Сервис блобов в *Windows Azure Storage* имеет простую структуру. Являясь *REST*-сервисом, хранилище предоставляет к каждой из хранимых сущностей гиперссылку, которая состоит из нескольких сегментов:

1. **Аккаунт** – весь доступ к хранилищу происходит через аккаунт хранилища, и имя аккаунта является первым сегментом в ссылке на блоб.

2. **Контейнеры блобов** аналогичны директории верхнего уровня традиционных файловых систем. Могут содержать только блобы, до 100 ТБ. Каждый контейнер имеет права доступа: приватный уровень (нужен ключ аккаунта), полное публичное чтение и публичное только чтение. Также внутри каждого аккаунта имеется специальный корневой контейнер $*root*, действующий как корень диска в традиционной файловой системе. К контейнеру можно привязать словарь метаданных, несущих дополнительную информацию.

3. **Блобы**, хранящие в себе любые бинарные данные, могут иметь ассоциированные с ними метаданные в виде пар «ключ – значение» и размером до 8 килобайт на блоб.

Блобы делятся на два типа – блочные и страничные.

Блочные блобы используются для потоковых нагрузок. Блочный блоб выглядит как логическая последовательность блоков, каждый из которых определяется специальным идентификатором и максимальным размером 200 Гб на один блоб. Для блочного блоба используется оптимистичный параллелизм через *ETags*. Модификация

блочного блоба состоит из двух этапов: сначала блоки должны быть загружены в хранилище как неподтвержденные блоки для конкретного блоба, после чего для создания обновленного блоба используется метод *PutBlockList*.

Страничные блобы используются для операций случайного чтения и записи. Страничный блоб выглядит как массив страниц, при этом каждая страница определяется отступом от начала блоба. Размер ограничен 1 Тб на блоб. Для страничных блобов используется либо оптимистичный, либо пессимистичный (блокировка) параллелизм через *Leases*. Обновление происходит сразу же в завершение запроса на запись последовательного набора страниц, поэтому блоки не надо подтверждать (*commit*). Важная особенность страничных блобов заключается в том, что оплата за хранение страничных блобов производится только за непустые страницы.

К сценариям использования блобов можно отнести:

– доставку статического контента в браузер;
– хранение контента для распределенного и глобального доступа;
– стриминг видео и аудио;
– резервирование данных.

Рассмотрим базовые операции, доступные для управления блобами в хранилище *Windows Azure*:

1. **Синхронное создание контейнера блобов**

Стандартным способом создания контейнера блобов является синхронная операция. Для выполнения этой операции должен быть создан экземпляр класса *CloudBlobClient*, необходимый для предоставления операций, доступных для управления блобами и контейнерами блобов, после чего, на основе этого экземпляра, получается ссылка на контейнер блобов (ссылка на контейнер может быть получена даже в том случае, если его не существовало ранее). С помощью выполнения метода *CreateIfNotExists* контейнер создается, если его ранее не было. Для создания контейнеров также доступен метод *Create*, однако в случае существования контейнера с таким именем будет выброшено исключение, поэтому с позиции безопасности лучше использовать *CreateIfNotExists*.

При создании контейнера блобов необходимо учитывать следующие ограничения:

– имя контейнера должно начинаться с буквы или цифры и может содержать только буквы, цифры и символы дефиса;
– все буквы должны быть низкого регистра;
– длина имени должна быть от 3 до 63 символов;
– имя не может содержат дефис после точки.

Библиотека БГУИР им. П.О. Сухого

Асинхронный вариант операции создания использует метод *BeginCreateIfNotExist*.

2. **Удаление контейнера блобов**

Для выполнения операции удаления должен быть создан экземпляр класса *CloudBlobClient*, необходимый для предоставления операций, доступных для управления блобами и контейнерами блобов, после чего, на основе этого экземпляра, получается ссылка на контейнер блобов (ссылка на контейнер может быть получена даже в том случае, если его не существовало ранее). Контейнер может быть удален с помощью метода ***Delete***.

3. **Создание блоба**

Для создания блоба в указанном контейнере достаточно воспользоваться методом *GetBlobReference* объекта контейнера, передав ему имя блоба. После этого будет создан пустой блоб, который необходимо заполнить данными.

4. **Создание и привязка метаданных к контейнеру блобов и блобу**

Блобы могут иметь ассоциированные с ними метаданные. Заголовки метаданных могут быть определены по запросу при создании нового контейнера или блоба, или при «привязке» метаданных к уже существующим ресурсам. Получение метаданных производится с помощью свойства *Metadata*.

5. **Определение прав доступа для контейнера**

Класс *BlobContainerPublicAccessType* является перечислением (*enumeration*), состоящим из следующих возможных значений прав доступа:

- ***Blob***. Публичный доступ на уровне блоба. Анонимные пользователи могут получать контент и метаданные блобов внутри этого контейнера, но не могут получать метаданные контейнера и список блобов внутри него.
- ***Container***. Публичный доступ на уровне контейнера. Анонимные пользователи могут получать контент и метаданные блоба и контейнера и список блобов внутри контейнера.
- ***Off***. Анонимный доступ отключен, только владелец аккаунта имеет доступ к каким-либо ресурсам внутри этого контейнера.

6. **Управление доступом с помощью *Shared Access Signatures* и *Shared Access Policies***

Shared Access Policies и *Shared Access Signatures* могут быть использованы для создания некоторых правил доступа к блобу или контейнеру, регламентирующих права доступа и период доступа.

Shared Access Policy. Политика определяет время начала дейст-

вия политики, время истечения и набор разрешений для *Shared Access Signatures*.

Shared Access Signature. *URL*, предоставляющий доступ к контейнеру и блобу.

5.2. Таблицы

Данные в сервисе таблиц *Windows Azure* хранятся в виде коллекций сущностей. Сущность имеет первичный ключ и набор свойств. Свойства являются парами ключ-значения, что аналогично столбцам. Сервис таблиц в *Windows Azure* не является реляционным, поэтому нельзя выполнять следующие операции:

– создание внешних ключей между таблицами;
– выполнение операции объединения на стороне сервера;
– создание произвольных индексов.
– выполнение таких функций, как, например, *Count*() на стороне сервера.

Сервис таблиц подходит в том случае, если разработчику не требуется реляционное хранилище или не нужны операции объединения (*joins*) на стороне сервера, а также в тех случаях, когда набор данных не очень велик и операции объединения можно обрабатывать на стороне клиента с помощью *LINQ*. Для доступа к сервису таблиц разработчик может использовать *REST API*, совместимый с *WCF Data Services* (ранее *ADO.NET Data Services Framework*).

Сущность в таблицах состоит из следующих компонентов:

1. **Свойство (столбец)** – значение сущности. Имена свойств чувствительны к регистру. Свойства эквивалентны столбцам в классических СУБД, одна сущность может иметь до 255 свойств.
2. ***PartitionKey*** – первое обязательное свойство каждой таблицы, используемое системой для автоматической балансировки нагрузки и распределения сущностей таблицы между серверами.
3. ***RowKey*** – второе обязательное свойство каждой таблицы, являющееся уникальным *ID* сущности внутри партиции, в которой оно расположено, и вторым компонентом в комбинации с *RowKey*, уникально идентифицирующей сущность в таблице.
4. ***Timestamp*** – каждая сущность имеет версию, управляемую системой, необходимую для оптимистического параллелизма.

Для того чтобы создать модель в клиентском приложении, которая будет отображаться из хранилища таблиц, необходимо создать

Библиотека ГГТУ им. П.О.Сухого

класс, реализующий *TableServiceEntity*. Приведенный ниже код содержит определение класса, реализующего *TableServiceEntity*.

```
public class MyClass : TableServiceEntity
 {
 public MyClass()
 {
 base.PartitionKey = "Clients";
 base.RowKey = Guid.NewGuid().ToString();
 }
 public MyClass(string PartitionKey, string RowKey)
 {
 base.PartitionKey = PartitionKey;
 base.RowKey = RowKey;
 }
 public string FName { get; set; }
 public string LName { get; set; }
 }
```

Класс *TableServiceEntity* определяет системные свойства *PartitionKey*, *RowKey* и *TimeStamp*, необходимые для каждой сущности в таблице *Windows Azure*. В примере выше для свойства *PartitionKey* было определено фиксированное значение. В реальной ситуации необходимо выбирать такое значение, которое обеспечит балансировку нагрузки между узлами хранилища, динамическое вычисляемое значение, например, день добавления сущности.

Характерные особенности сервиса таблиц включают в себя:

1. **Порядок сортировки** – есть единственный индекс в таблицах, когда таблицы сортируются сначала по *PartitionKey*, после чего по *RowKey*. Это означает, что записи, указывающие данные ключи, будут более эффективны, и результаты будут сортироваться сначала по *PartitionKey* и потом по *RowKey*.

2. **Типы** – *PartitionKey* и *RowKey* должны быть типа *string*, остальные свойства: *Binary*, *Bool*, *DateTime*, *Double*, *GUID*, *Int*, *Int*64, *String*.

3. **Нет фиксированной схемы** – в таблицах *Windows Azure Storage* нет схем, поэтому все свойства хранятся в парах «ключ – типизированное значение». Это означает, что две сущности в одной таблице могут иметь разные свойства. Например, в таблице может быть два свойства с одинаковым именем, но разными типами для значения свойства. В пределах одной сущности имена свойств должны быть уникальными.

Библиотека ГГТУ им. П.О.Сухого

Ссылка на таблицу выглядит стандартно для именования сущностей в сервисах хранилища *Windows Azure*:

http://<*account*>.*table*.*core*.*windows*.*net*/<*TableName*>

Рекомендация: так как используется локальный контекст данных, то таблицы данных должны создаваться только один раз. Обычно этот процесс выполняется на стадии подготовки и реже – в коде приложения. *Application_Start* в классе *Global* является рекомендуемым местом для помещения логики инициализации.

Для использования и управления содержимым таблицы необходимо создать объект типа *CloudTableClient*, предоставляющий следующие базовые методы:

– *CreateTable* – создание таблицы с определенным именем. В случае наличия таблицы с таким именем будет выброшено исключение *StorageClientException*. Наиболее предпочитаемым спосо-бом создания таблиц является их создание с помощью класса контекста.

Пример

```
CloudTableCli-
ent.CreateTablesFromModel(typeof(MyClassDataContext),
 account.TableEndpoint.AbsoluteUri, account.Credentials);
 }
```

– *CreateTableIfNotExist* – создание таблицы с определенным именем только в том случае, если ее не существует.

– *DoesTableExist* – проверка на существование таблицы с определенным именем.

– *DeleteTable* – удаление таблицы и ее содержимого из хранилища. Если этой таблицы не существует, будет выброшено исключение *StorageClientException*.

– *DeleteTableIfExist* – удаление таблицы и содержимого из хранилища только в случае, если она существует.

– *ListTables* – получение списка всех таблиц. Возможно указание префикса для фильтрации имен таблиц.

– *AddObject* – добавление сущности в таблицу.

В том случае, если в приложении происходит создание множества сущностей, дешевле (с позиции количества транзакций) и быстрее будет выполнить пакет запросов. Для этого необходимо передать в *SaveChangesWithRetries* соответстующий аргумент *SaveChangesOptions*.*Batch*. При этом необходимо учитывать, что в пакете можно совершать запросы *update*, *delete* и *insert*; один пакет может содержать до 100 сущностей; все сущности в одном пакете должны иметь одно

Библиотека ГГТУ им.П.О.Сухого

значение *PartitionKey*.

Получение объектов может быть реализовано с помощью *LINQ*-утверждения. Для возвращения всех записей в пределах одной партиции и имеющих определенное значение поля, можно воспользоваться следующим *LINQ*-утверждением:

```
CloudTableQuery<MyClass> qry =
 (from e in ctx.CreateQuery<MyClass>("mytable")
 where e.PartitionKey == "Belotserkovskiy" &&
e.FName == "Alexander"
 select e).AsTableServiceQuery<MyClass>();

foreach (MyClass unit in qry)
{
 Console.WriteLine("{0}, {1}", unit.PartitionKey, unit.FName);
}
```

LINQ-утверждения можно также использовать и для любых других операций, в том числе обновления и удаления сущностей.

```
MyClass unit = (from e in ctx.CreateQuery<MyClass>("mytable")
where e.PartitionKey == "Belotserkovskiy" && e.FName == "Alexander"
 select e).AsTableServiceQuery<MyClass>().FirstOrDefault();
unit.FName= "Alex";
ctx.UpdateObject(unit);
ctx.SaveChangesWithRetries();
ctx.DeleteObject(unit);
ctx.SaveChangesWithRetries();
```

5.3. Очереди

Механизм очередей *Windows Azure Storage* используется для надежного хранения и дальнейшей доставки сообщений. Очереди *Windows Azure Storage* являются стандартным *Middleware*-обеспечением, который используется для обеспечения однонаправленных коммуникаций между *Web*- и *Worker*-ролями в *Windows Azure CloudServices* – например, *Web*-роль кладет сообщения в очередь, которую в бесконечном цикле опрашивает *Worker*-роль и, забрав сообщение, приступает к его обработке. Если экземпляры *Worker*-роли не могут забрать или обработать сообщение, оно будет храниться некоторое время в очереди.

Библиотека ГГТУ им. П.О. Сухого

Важным замечанием про очереди является то, что имя очереди должно быть именем в нижнем регистре и *URL-friendly* для удобства использования его в *REST*.

В очередях хранятся сообщения. Сообщения характеризуются:

– очередь может содержать неограниченное количество сообщений;

– сообщения должны быть сериализуемы как *XML*;

– размер ограничен 64 Кб;

– обычно используется паттерн *work ticket*, суть которого заключается в передаче внутри сообщения не непосредственно сущности, но ссылку на нее и метаданные, содержащие в себе указания для обработки этой сущности;

– сборщик мусора для очереди запускается раз в неделю.

Ссылка на очередь выглядит стандартно для именования сущностей в сервисах хранилища *Windows Azure*:

http://<*account*>.*queue.core.windows.net*/<*QueueName*>

К основным операциям над очередями с использованием клиента облачных очередей *CloudQueueClient* относятся получение ссылки на очередь (*GetQueueReference*, возвращает объект *CloudQueue*) и получение списка очередей (*ListQueues*).

У объекта *CloudQueue* доступно несколько базовых методов по управлению очередью:

– *SetMetadata*: определяет метаданные для очереди;

– *FetchAttributes*: загружает метаданные и атрибуты;

– *Clear*: очищает очередь;

– *Create*: создает очередь. Если очередь уже существует, будет выброшено исключение;

– *CreateIfNotExists*: создает очередь в том случае, если ее не существует;

– *Delete*: удаляет очередь;

– *Exists*: проверяет существование очереди;

– *AddMessage*: добавляет сообщение в очередь;

– *DeleteMessage*: удаляет сообщение из очереди;

– *GetMessage*: возвращает следующее сообщение из очереди. В это время сообщение становится «невидимым» для других обработчиков;

– *GetMessages*: возвращает определенное количество сообщений из очереди;

Библиотека ГГТУ им. П.О. Сухого

– *PeekMessage*: возвращает сообщение из очереди, оставляя его видимым для других обработчиков («подсматривает»);

– *PeekMessages*: возвращает определенное количество сообщений из очереди, оставляя их видимыми для других обработчиков;

– *UpdateMessage*: изменяет содержимое или время видимости сообщения;

– *RetrieveApproximateMessageCount*: возвращает примерное количество сообщений в очереди. Примерное по той причине, что сообщения могут быть добавлены или удалены после того, как отправляется запрос на количество сообщений очереди.

Усеченный экспоненциальный алгоритм отката

Одним из подходов к опросу очереди и уменьшению нагрузки на сервис является использование усеченного экспоненциального алгоритма отката. Суть данного подхода заключается в том, что каждый «пустой» опрос увеличивает интервал опроса вдвое, не пустой же опрос ставит интервал обратно в 1. Код, приведенный ниже, показывает возможную реализацию данного подхода.

```
while (true)
 {
 var msg = queue.GetMessage();
 if (msg != null)
 {
 string retrievedMsg = msg.AsString;
 //логика
 query.DeleteMessage(msg);

 if (gradualDecrease)
 if (curInterval > intervalFloor) curInterval = curInterval /2;
 else curInterval = intervalFloor;
 else curInterval = intervalFloor;
 }
 else
 {
 if (curInterval <intervalCeiling) curInterval = curInterval * 2;

 }
```

Долгие очереди

Очередь, собирающая сообщения в период отключенного потребителя, называется долгой. В этом случае потребитель очереди

Библиотека ГГТУ им. П.О. Сухого

может выходить в онлайн раз в день и забирать все сообщения, затем снова отключаться. Полезно для обмена сообщениями с внешними вендорами, а также для сокращения оплаты за время выполнения сервиса (в том случае, если потребитель является, например, *Worker*-ролью в *Windows Azure*).

Разбиение процесса

В качестве рекомендаций можно посоветовать разбиение сложного процесса на состояние, при этом каждое состояние является отдельным процессом-потребителем, опрашивающим определенную очередь. Подход, при котором один большой потребитель обрабатывает одну очередь, является негибким. Архитектура, являющаяся более оптимальной с позиции гибкости, разделена на различных потребителей и очередей, каждый из которых занимается собственной задачей.

5.4. Партиции

Windows Azure Storage использует специальное системное поле *PartitionKey* для того, чтобы группировать сущности в партиции. Партиции – одно из важнейших понятий на платформе *Windows Azure*, так как именно оно позволяет виртуально неограниченно масштабировать хранилище, логически разделяя данные на группы. Для того чтобы лучше понимать, как работают партиции, рассмотрим принципы работы всего хранилища *Windows Azure Storage*. *Windows Azure Storage* – это система, имеющая трехслойную архитектуру. На первом уровне находится *VIP* (*Virtual IP*), конечная точка, по которой доступна система. Далее все запросы масштабируются на три экземпляра фронтендов (*FE*), серверов, принимающих запросы на сущности. На данном слое происходит логика определения прав, аутентификация клиента и маршрутизация запроса на следующий уровень. Следующий уровень, уровень партиций (*Partition Layer*), управляется серверами партиций и мастерами партиций. Каждому серверу партиций принадлежит определенный набор партиций, за который он «ответственен» – мастер партиций также контролирует загрузку всех серверов. На этом слое происходит балансировка нагрузки между серверами партиций и партициями.

На самом низком уровне находится распределенная файловая система – *Distributed File System* (*DFS*). На этом слое хранятся данные, доступные всем серверам партиций. *DFS* балансирует нагрузку

Библиотека ГГТУ им. П.О.Сухого

на систему, распределяя запросы, и состоит из множества узлов, по которым распределены наборы данных (*extents*), причем каждый из наборов, данных назначается одному ***primary server*** (первичному серверу) и нескольким ***secondary server*** (вторичным серверам). Каждый экстент имеет данных от 100 Мб до 1 Гб, одна сущность может быть распределена по нескольким экстентам. При операциях записи сущностей эти операции сначала обрабатываются первичным сервером (но не выполняются), после чего передаются на выполнение операции записи какому-то из вторичных серверов. Первичный сервер в данном случае выполняет роль контроллера, распределяющего операции и следящего за их выполнением. Первичный сервер не подтверждает операцию записи до тех пор, пока как минимум два вторичных сервера не сообщат об успешной записи данных в три домена обновлений и неисправностей.

Данные реплицируются в нескольких экземплярах внутри *DFS* (минимальное количество копий должно быть равно 3). Большие сущности могут быть разбиты на части и распределены по нескольким экстентам, что приводит к тому, что одна сущность может храниться на нескольких дисковых устройствах.

Размер партиции

Размер партиции равен количеству сущностей, которое хранится в партиции. Разреженность значения *PartitionKey* является важным показателем, так как, если для всех сущностей используется одно значение *PartitionKey*, то это значит, что будет существовать одна партиция, либо, если значений много, каждая сущность может содержаться в отдельной партиции.

Цели масштабирования по партициям

Цель масштабирования, показатели, которые определяют, сколько может «выдержать» одна партиция, равно 500 сущностей в секунду. Партиции переносятся между серверами хранилища для обеспечения высокой степени эластичности и максимальной производительности. «Горячие», т. е. испытывающие высокую степень нагрузки, партиции могут быть вертикально масштабированы – *Windows Azure Fabric Controller* способен выделить больше ресурсов для партиций с большим количеством транзакций. Таким образом, если на одном сервере расположены три партиции, то, если одна из партиций начинает испытывать высокие нагрузки и большое количество запросов, она может быть перемещена на второй сервер. После того как нагрузка на нее спадет, она может быть возвращена на исходный сервер.

Библиотека ГГТУ им. П.О.Сухого

Каждый тип хранилища определяет свою партицию:

1. **Очередь -> Одна очередь = Одна партиция**

Сервис очереди использует имя очереди в качестве ключа партиции, и все сообщения для данной конкретной очереди будут находиться в одной партиции. Это означает, что каждая очередь имеет собственную цель масштабирования. Необходимо, однако, учитывать виртуальное ограничение в 500 сущностей в секунду, поэтому, если очередь испытывает нагрузку более 500 сущностей в секунду, необходимо определить, каким образом можно разделить систему на несколько очередей для реализации высокопроизводительного механизма обмена сообщениями.

2. **Таблица -> Одна партиция таблицы = Одна партиция**.

В сервисе таблиц разработчик сам определяет, как будет партиционироваться таблица, с помощью определения системного свойства *PartitionKey*. Рекомендуется создавать маленькие многочисленные партиции, так как именно в этом случае будет организована наиболее эффективная масштабируемость на уровне платформы, а вместе с этим и скорость обращения к данным. Каждая сущность будет принадлежать партиции, определенной в ее *PartitionKey*. Если разработчик использует распространенную практику инкрементирования (или декрементирования) значения *PartitionKey*, он можете столкнуться с характерным поведением платформы *Windows Azure* – созданием партиций-диапазонов. Это необходимо для повышения эффективности запросов на диапазоны, которые без партиций-диапазонов должны были бы выходить за пределы партиции или сервера, что понизило бы производительность. При использовании инкрементирования может произойти следующая ситуация: платформа объединит первые три сущности в партицию-диапазон и, если разработчик выполнит запрос на диапазон с использованием *PartitionKey* и запросит сущности с *PartitionKey* между 1 и 3, запрос будет выполнен быстро, так как партиция будет находиться в пределах одного сервера.

3. **Блоб -> Один блоб = Одна партиция**.

При записи в партицию операция считается завершенной по записи на все три реплики.

5.5. Избыточность хранилища Windows Azure Storage

В *Windows Azure Storage* доступно две опции избыточности: *Locally Redundant Storage* и *Geo Redundant Storage*.

***Locally Redundant Storage* (*LRS*)** предоставляет хранилище с высокой степенью долговечности и доступности внутри одной географической локации (региона). Платформа хранит три реплики каждого элемента данных в одной первичной географической локации, что гарантирует, что эти данные можно будет восстановить после сбоя, несущего общий характер (например, выхода из строя диска, узла, корзины и так далее, что является нормальным событием в центре обработки данных) без потери функциональности для аккаунта хранилища и, соответственно, не влияя на доступность и долговечность хранилища. Все операции записи в хранилище выполняются синхронно в три реплики в трех различных доменах ошибок (*fault domain*), код об успешном завершении транзакции возвращается после успешного завершения всех трех операций. В случае использования локального избыточного хранилища, если центр обработки данных, в котором размещены реплики данных, подвергнется сбою, несущему характер катастрофы (стихийной, техногенной или любой другой, которая приведет к потере подключения к центру обработки данных либо потере самого центра), *Microsoft* свяжется с клиентом и сообщит о возможной потере информации и данных, используя контакты, приведенные в подписке клиента.

Вторая опция, ***Geo Redundant Storage* (*GRS*),** предоставляет гораздо более высокую степень долговечности и безопасности. Реплики данных в этом режиме размещаются не только в первичной географической локации, но и во вторичной, находящейся в том же регионе, но за сотни километров. Таким образом, в географически избыточном режиме платформа сохраняет три реплики, но в двух локациях. Это гарантирует то, что, если центр обработки данных подвергнется сбою, несущему характер катастрофы, то данные будут доступны из вторичной локации. Как и в случае первой опции избыточности, операции записи данных в первичной географической локации должны быть подтверждены перед тем, как система вернет код успешного завершения операции. По подтверждению операции в асинхронном режиме будет инициирован процесс репликации в другую географическую локацию. Рассмотрим подробнее процесс географической репликации.

Библиотека ГГТУ им. П.О. Сухого

Когда осуществляются операции создания, удаления, обновления, транзакция полностью реплицируется в три узла хранилища в трех доменах ошибок и обновлений в первичной географической локации, после чего клиенту возвращается код успешного завершения операции и в асинхронном режиме подтвержденная транзакция реплицируется во вторичную локацию, где полностью реплицируется в три узла хранилища в разных доменах ошибок и обновлений. Общая производительность при этом не падает, так как все совершается асинхронно. Если происходит серьезный сбой в первичной географической локации, применяются правила географической отказоустойчивости, то клиент оповещается о возникшей катастрофе в первичной локации, после чего соответствующие ресурсные *DNS*-записи изменяются и начинают вести не на первичную локацию, но на вторую (*DNS*-запись *account.service.core.windows.net*). В процессе перевода *DNS*-записей могут наблюдаться сбои в работе, но по его завершению существующие блобы и таблицы становятся доступными по их прежним *URL*-ам. После завершения процесса перевода вторичная локация повышается в статусе до первичной; в завершение процесса повышения статуса центра обработки данных инициируется процесс создания новой вторичной географической локации в этом же регионе и дальнейшей репликации данных.

Рассмотрим реальный пример, взятый из блога разработчиков *Windows Azure Storage*. В аккаунте хранилища размещено несколько блобов, *foo* и *bar*. Для блобов полное имя блоба равно значению *PartitionKey*. Разработчик выполняет две транзакции *A* и *B* на блобе *foo*, после чего выполняет две транзакции *X* и *Y* на блобе *bar*. Система гарантирует, что транзакция А будет географически реплицирована перед транзакцией *B*, и, соответственно, транзакция *X* будет географически реплицирована перед транзакцией *Y*.

Географически избыточное хранилище включается по умолчанию для всех создающихся аккаунтов хранилища. Можно отключить географическую репликацию на портале управления *Windows Azure* либо указать эту опцию при создании аккаунта хранилища.

Локально избыточное хранилище полезно в том случае, если необходимо сократить затраты на хранение некритичных либо легковоспроизводимых данных, тогда как географически избыточное может быть полезно при наличии важных данных.

Библиотека ГГТУ им. П.О.Сухого

5.6. Диагностика хранилища

Сервис *Windows Azure Storage Analytics* предоставляет клиенту возможность отслеживать и анализировать использование данных, храня-щихся в аккаунте хранилища (блобы, таблицы и очереди), используя эти данные для адаптации приложения к реальным нагрузкам. Аналитические данные состоят из **логов** (выполненные запросы к аккаунтам хранилища) и **метрик** (статистика по объему и запросам блобов, таблиц и очередей).

Логи хранятся в виде блочных блобов в специальном контейнере блобов ($*logs*). Каждая запись-лог в блобе отображает один сделанный запрос и содержит служебную информацию (*ID* запроса, *URL* запроса. *HTTP*-статусы, имя аккаунта клиента, имя аккаунта хозяина, *IP*-адрес клиента и т. д.). Логи позволяют выяснить:

– сколько анонимных запросов было выполнено от определенного диапазона *IP*-адресов;

– к каким контейнерам блобов был наиболее активный доступ;

– сколько раз и каким образом был запрошен конкретный *URL* с *Shared Access Signature*;

– кто запросил удаление контейнера блобов;

– пришел ли запрос на сервер или нет.

Метрики сводят воедино основную статистику для блобов, таблиц и очередей. При этом статистику можно подразделить на два типа – информацию о запросах (количество запросов по часам, средняя задержка на стороне серверов, средняя задержка *E2E*, средняя полоса пропускания, общее количество успешных и неуспешных запросов и т. д., и эта информация предоставляется на уровне сервиса и *API* и доступна для блобов, таблиц и очередей) и информацию об объеме (ежедневная статистика потребленного сервисом пространства, количество контейнеров и количество объектов, хранящихся внутри сервиса, доступна только для блобов). Метрики хранятся внутри специальной служебной таблицы в сервисе таблиц.

Все аналитические логи, статистика и данные метрик хранятся внутри пользовательского аккаунта, и получить к ним доступ можно через стандартное *REST API* сервисов блобов и таблиц и с помощью портала управления *Windows Azure*. Кроме этого логи и метрики могут быть потреблены из любого сервиса, размещенного как в *Windows Azure*, так и в интернете или локального приложения, которое умеет работать и обрабатывать *HTTP*/*HTTPS*-запросы. Выключить или включить логи и метрики можно как с помощью *REST API*, так и с

Библиотека ГГТУ им.П.О.Сухого

помощью портала управления *Windows Azure*. Также можно настроить срок, после которого данные будут автоматически удаляться.

5.7. Обеспечение безопасности с использованием Shared Access Signatures

Shared Access Signatures доступны для всех сервисов *Windows Azure Storage*. Функциональность *Shared Access Signature* заключается в предоставлении возможности детального контроля доступа к ресурсам; определения, какие операции может совершить пользователь над ресурсом, имея *Shared Access Signature*. К списку доступных для определения *SAS* операций относятся:

– чтение и запись контента – в случае блобов дополнительно их свойства и метаданные, а также блок-списки;

– удаление, лизинг, создание снапшотов блобов;

– получение списков элементов контента;

– добавление, удаление, обновление сообщений в очередях;

– получение метаданных очередей, включая количество сообщений в очереди;

– чтение, добавление, обновление и вставка сущностей в таблицу.

Параметры *Shared Access Signature* включают в себя всю информацию, необходимую для выдачи доступа к ресурсам хранилища – параметры запроса в *URL* определяют временной промежуток, через который *Shared Access Signature* будет деактивирована, разрешения, предоставляемые данной *Shared Access Signature*, ресурсы, к которым предоставляется доступ и сигнатуру, с помощью которой происходит аутентификация. Кроме этого в *URL Shared Access Signature* можно включить ссылку на хранимую политику доступа, с помощью которой можно обеспечить более гибкий контроль доступа.

Shared Access Signature должны распространяться с использованием *HTTPS* и разрешать доступ на максимально короткий временной промежуток, необходимый для выполнения операций.

Строение *Shared Access Signature*

В составе *URL Shared Access Signature* находятся компоненты, специфичные для *REST* и специфичные для *Shared Access Signature*:

1. **Параметр *signedversion***

Содержит версию сервиса, обеспечивающего *Shared Access Signature*. При этом, если *Shared Access Signature* указывает на сервис хранилища версии, более ранней, чем 2012-02-12, этот *Shared Access Signature* может оперировать только блобом или контейнером.

Библиотека ГГТУ им. П.О.Сухого

Сервисы *Windows Azure Storage* могут принимать запросы, которые указывают конкретную версию каждой операции (сервиса), при этом разработчик может указать версию, используя заголовок *x-ms-version*. Функциональность различных версий и механизмы работы (не концептуальные) могут различаться.

Request Headers:

x-ms-version: 2011-08-18

Если *URL Shared Access Signature* использует этот параметр со значением версии, которая является более новой, чем клиентское приложение, использующее этот *URL*, могут возникнуть функциональные различия. Например, в версиях, более ранних, чем 2011-08-18, операция *Get Blob* не возвратит заголовок *Accept-Ranges*.

2. **Параметр *Signed Resource* (только для блобов)**

Значение параметра определяет ресурсы, к которым обеспечивается доступ данной *SAS*. Имеет значение либо ***b*** (блоб), что обеспечивает доступ к контенту и метаданным блоба, либо ***c*** (контейнер), что обеспечивает доступ к контенту и метаданным любого блоба в контейнере и получение списка блобов в этом контейнере.

3. **Параметр *Table Name* (только для таблиц)**

Определяет имя таблицы.

4. **Параметр *Access Policy***

Определяет значения временного промежутка, в течение которого будет работать данная *Shared Access Signature*, и набор разрешений. Состоит из следующих частей:

– *signedstart* – начальный момент времени в *ISO* 8061 (например, так – *YYYY-MM-DDThh*:*mm*:*ssTZD*), когда сигнатура будет активирована. Если опущено, сигнатура будет активирована в момент получения запроса;

– *signedexpiry* – конечный момент времени в *ISO* 8061, когда сигнатура станет неактивной. Опускается, если соответствующее значение указано в хранимой политике доступа;

– *signedpermissions* – набор разрешений, ассоциированных с сигнатурой (*r*, *w*, *d* для блобов *r* и *w* для контейнеров блобов, *r a* (*add*) *и p* (*process*, забор и удаление и *r a*(*add*) *и d* (*delete*) для сущностей таблиц) для сообщений в очередях). Опускается, если соответствующее значение указано в хранимой политике доступа. Пример: *sp* = *rwdl*, *sp* = *rw*, *sp* = *r*. Нельзя выдавать разрешения на создание и удаление контейнеров, очередей и таблиц, получения списка сущностей в них, получения и редактирования метаданных и свойств кон-

тейнеров блобов и очистки метаданных и очистки очередей от сообщений.

Startpk, *startrk* – только для таблиц. Минимальное значение *partition key* и *row key*, которое доступно пользователю.

Endpk, *endrk* – только для таблиц. Максимальное значение *partition key* и *row key*, которое доступно пользователю.

5. **Параметр Signed Identifier**

Определяет соответствующую *Shared Access Signature* хранимую политику доступа. Хранимые политики доступа предоставляют определенную степень контроля над одной или более *Shared Access Signature*, включая отзыв *Shared Access Signature*. Каждый контейнер блобов, очередь или таблица может иметь до пяти ассоциированных с ней хранимых политик доступа.

К сценариям использования *Shared Access Signature* можно отнести два примера:

1. Сервис должен давать доступ клиентам к каким-то частям аккаунта хранилища с определенными разрешениями, например, приложение *Windows Phone* для сервиса, который запущен в *Windows Azure*. Токен *Shared Access Signature* должен распространяться клиентам (приложениям *Windows Phone*) для обеспечения доступа к хранилищу *Windows Azure*.

2. Сервис должен выдавать доступ к ресурсам по необходимости, также быстро закрывая его после окончания срока действия токенов.

Генерация токенов может проходить согласно следующим моделям:

- Токен генерируется специальным сервисом на ограниченный период времени. Успешнее всего эта модель подходит к первому сценарию использования, описанному выше. Непосредственно перед истечением периода действия токена приложение запрашивает новый токен, сервис решает по каким-либо правилам выдавать этот токен или нет.

- Канал коммуникаций между клиентом и сервисом, генерирующим токены, может использовать механизм аутентификации (например, *OAuth*) и клиент должен сначала аутентифицироваться, после чего запросить токен *Shared Access Signature*.

- Для второго описанного выше сценария можно использовать сгенерированные токены *Shared Access Signature*, привязанные с хранимой политикой доступа.

Библиотека ГГТУ им. П.О.Сухого

Локальный эмулятор *Windows Azure Storage*

Локальный эмулятор хранилища *Windows Azure* эмулирует сервисы *Windows Azure Storage*. Эмулятор поставляется в комплекте *Windows Azure SDK* и его можно использовать для разработки и тестирования приложений, использующих блобы, очереди и таблицы, в локальной инфраструктуре. Между локальным эмулятором и реальными сервисами хранилища есть некоторые различия.

К общим различиям относятся количество аккаунтов и ключ аутентификации: локальный эмулятор хранилища поддерживает только один фиксированный аккаунт и один ключ аутентификации, при этом они не меняются и равны, например:

Имя аккаунта: *devstoreaccount*1

Ключ аккаунта:

*Eby*8*vdM*02*xNOcqFlqUwJPLlmEtlCDXJ*1*OUzFT*50*uSRZ*6*IFsuFq*2 *UVErCz*4*I*6*tq*/*K*1*SZFPTOtr*/*KBHBeksoGMGw*==

Локальный эмулятор хранилища не поддерживает большое количество параллельных подключений и клиентов и не поддерживает возможности масштабирования, имея отличную схему *URI* от схемы *URI* реальных сервисов хранилища *Windows Azure*. *URI* локального эмулятора определяет имя аккаунта как часть пути в *URI*, но не как часть доменного имени.

Необходимо учитывать различия между локальным эмулятором и облачными сервисами хранилища перед тем, как переводить приложение в *Windows Azure*.

5.8. Windows Azure SQL Databases

Windows Azure SQL Databases – это облачный сервис баз данных, основанный на технологиях, использующихся в классическом *SQL Server*. *Windows Azure SQL Databases* предоставляет доступную глобально инфраструктуру и функциональность базы данных как сервис (*Database As A Service*). Основанный на облачных технологиях сервис *Windows Azure SQL Databases* предлагает большое количество характерных преимуществ, включающих быстрое развертывание, экономичное масштабирование, высокую доступность и сокращение издержек на управление. С точки зрения разработчика, *Windows Azure SQL Databases* предлагает знакомую реляционную модель программирования и методы доступа к данным и простые варианты развертывания, которые уже используются разработчиком. Как и любой об-

Библиотека ГГТУ им.П.О.Сухого

лачный сервис, *Windows Azure SQL Databases* устраняет проблемы с инфраструктурой, тем самым предоставляя разработчикам больше свободы. С точки зрения *IT*, *Windows Azure SQL Databases* предлагает надежное и безопасное облачное решение, которое может быть интегрировано локальной инфраструктурой, с гибкими механизмами контроля и управления. Ключевым достоинством сервиса является простота масштабирования решения. При возрастании требований есть возможность горизонтального и вертикального масштабирования.

При разработке локальных приложений с использованием *SQL Server* разработчики используют для взаимодействия с источником данных различные клиентские библиотеки (протокол *TDS*). *Windows Azure SQL Databases* имеет аналогичный интерфейс *TDS*, поэтому разработчики могут использовать те же самые инструменты и библиотеки в процессе создания клиентских приложений с использованием данных в облаке.

Windows Azure SQL Databases оперируют стандартными терминами, за исключением того, что «контейнером» верхнего уровня, который должен быть создан, является сервер *Windows Azure SQL Databases*. Внутри каждого сервера *Windows Azure SQL Databases* могут быть созданы стандартные объекты *SQL Server* – таблицы, представления, хранимые процедуры, индексы и т. д. Подобная модель позволяет использовать существующую архитектуру реляционной БД и *Transact SQL* и упрощает процесс миграции существующих приложений в *Windows Azure SQL Databases*. Для задач миграции существует набор утилит, упрощающих процесс (например, встроенная функция *SQL Server Management Studio* или *AzureMW*). Серверы и базы данных *Windows Azure SQL Databases* являются виртуальными объектами, не соответствующими физическим сущностям.

Редакции *Windows Azure SQL Databases*

В свободном доступе для разработчика имеется две редакции баз данных *Windows Azure SQL Databases*, которые можно свободно менять на портале управления либо с помощью *SQL*-запроса: *Web* и *Business*. Основная разница между редакциями заключается в максимально доступном размере каждой базы данных в данной редакции (*Web* – до 5 Гб, *Business* – до 150 Гб). Каждая из редакций несет собственное назначение – например, базы данных *Web*-редакции подходят скорее для проектов, которые оперируют небольшим количеством данных, базы данных *Business*-редакции предназначены для корпоративных баз, данных, которым необходимые большие объемы хранилища.

Библиотека ГГТУ им. П.О.Сухого

В июле 2013 г. было объявлено о дополнительном предложении *Windows Azure SQL Databases Premium*, в рамках которого предлагается использование более мощных ресурсов для обслуживания серверов *Windows Azure SQL Databases*. Это предложение подразумевает под собой предоставление пользователю эксклюзивно зарезервированных под его нужды ресурсов. Использование базы данных редакции *Premium* позволяет пользователям масштабировать программную инфраструктуру, основываясь на пиковых нагрузках без того, чтобы учитывать при этом важную особенность облачных ресурсов – разделяемое использование одних и тех же ресурсов несколькими клиентами. Подобное предложение значительно упрощает разработку или миграцию локальных приложений в облако, которые проектируются с учетом пиковых нагрузок, и дает пользователю возможность масштабировать зарезервированные мощности таким образом, чтобы они удовлетворяли потребностям настоящего времени, например, в случае сезонных нагрузок приложению может понадобиться больше ресурсов; приложению необходимо большое количество ресурсов (*CPU*, *RAM*, *IO*) для выполнения операций с эксклюзивным доступом к ресурсам; приложение выполняет большое количество параллельных операций (запросов к базе данных); приложению должна гарантироваться минимальная латентность и т. д.

Миграция в *Windows Azure SQL Databases*

При миграции или разработке гибридной инфраструктуры с использованием *Windows Azure SQL Databases* разработчик может оставить клиентское приложение в локальной инфраструктуре и перенести только данные, к которым можно получить доступ с помощью библиотек доступа к данным. Однако, если используется подобная архитектура, необходимо учитывать проблемы с задержками, которые происходят в связи с географической удаленностью и коммуникационной инфраструктурой и могут привести к более сложному коду в приложении. Рекомендацией в данном случае может служить перемещение логики доступа к данным в *Windows Azure*. Таким образом приложение, использующее данные, находится там же, где и данные, в том же центре обработки данных. Например, в *Windows Azure* может быть расположен Web-интерфейс приложения.

Синхронизация данных

Сервис *Windows Azure SQL Databases* позволяет решать задачи по синхронизации данных. Например, в центральной базе данных в центре обработки данных на стороне клиента хранятся критичные

Библиотека ГГТУ им. П.О.Сухого

данные, сотрудники же организации используют приложение, которое работает на мобильных устройствах, и хранилище данных в *SQL Server Express*. В целях безопасности администратор организации не имеет права открывать брандмауэр на локальном центре обработки данных. В этом случае, с помощью сервиса *Windows Azure SQL Databases*, разработчик может обеспечить безопасное и полностью синхронизированное решение, создав базу данных в *Windows Azure SQL Databases*, провайдера *Sync Framework* для центра обработки данных, который позволит синхронизировать информацию между центром обработки данных и базой данных в *Windows Azure SQL Databases*, и второго провайдера *Sync Framework* для мобильных устройств сотрудников, который позволит синхронизировать информацию между этими устройствами и данными в *Windows Azure SQL Databases*.

Серверы *SQL Azure*

Каждый аккаунт *Windows Azure* может содержать несколько серверов *SQL Azure*. Эти сервера не являются экземплярами *SQL Server*, но являются абстракцией, использующейся для единой точки администрирования множества серверов. Каждый сервер включает в себя логины (аналогично *SQL Server*) и может быть расположен в регионе, указанном пользователем.

Для создания и управления сервером баз данных используется портал управления *Windows Azure*.

База данных *Windows Azure SQL Databases*

Каждый сервер *Windows Azure SQL Databases* может содержать до 148 баз данных. Каждый сервер содержит служебную базу данных *master*. В базах данных можно создавать стандартные объекты: таблицы, представления, индексы, хранимые процедуры и т. д. База данных может быть создана на портале управления аналогично созданию сервера. Фактически пользователю доступна только операция создания – далее присутствует выбор, необходимо ли использовать существующий сервер или создать новый.

Базы данных реплицируются в центре обработки данных, что обеспечивает автоматическую обработку ошибок и балансировку нагрузки. Данные, хранящиеся в базе, распределяются между множеством физических серверов внутри определенной пользователей для сервера географической локации. Таким образом для приложений различного уровня достигается высокая доступность и масштабируемость.

Архитектура доступа к данным

База данных *Windows Azure SQL Databases* работает по стандартному протоколу *TDS*. Также *TDS* используется в *SQL Server*. Та-

Библиотека БГТУ им. П.О. Сухого

ким образом, приложения, использующие *SQL Server*, могут быть подключены к базе данных *Windows Azure SQL Databases*. Для этого, однако, необходима поддержка *Secure Sockets Layer* (*SSL*).

Подробнее про архитектуру *Windows Azure SQL Databases* можно прочитать в специально посвященном этому вопросу документе по адресу: *http://rutechnet.blob. core.windows.net/files/SQL_Azure_Data-base.pdf*.

Модель безопасности *Windows Azure SQL Databases*

Многие базы данных содержат важные данные, поэтому необходимо правильно осуществлять контроль доступа, который особенно сложен в мультитенантном приложении, использующимся пользователями из различных организаций. Принципы безопасности *Windows Azure SQL Databases* аналогичны *SQL Server* с *SQL ServerAuthen-tication*:

– логины *SQL Server*: аутентификация доступа к *SQL Azure* на уровне сервера;

– пользователи БД: выдача доступа на уровне БД;

– роли БД: объединение группы пользователей и выдача прав доступа на уровне БД.

SQL Azure Databases использует стандартный протокол *SQL Server Tabular Data Stream* (*TDS*), при этом разрешены исключительно зашифрованные коммуникации. В *SQL Server* 2008 было введено новое средство: прозрачное шифрование данных (*transparent data encryption*, *TDE*), позволяющее полностью шифровать данные с минимальными усилиями. На данный момент *SQL Azure Databases* не поддерживает шифрование на уровне базы данных. Следует заметить, что в настоящее время *SQL Azure Databases* доступен только через *TCP*-соединения и только через порт 1433, поэтому необходимо учитывать возможности шифрования *ADO.NET* и сертификаты. Второе средство защиты *SQL Database Azure* – брандмауэр *SQL Azure Databases*, которым изначально блокируется весь доступ к серверу *SQL Azure Databases*. Попытки подключения до соответствующей настройки будут безуспешны. Брандмауэром *Windows Azure SQL Databases* можно управлять через портал *WindowsAzure SQL Databases* или напрямую в главной базе данных с помощью хранимых процедур, таких как *sp_set_firewall_rule* и *sp_delete_firewall_rule*.

При первой попытке управления созданным сервером портал *Windows Azure* предложит автоматически добавить *IP*-адрес пользователя в список разрешенных.

Как и в любой реализации *SQL Server*, управление пользовательскими учетными записями – еще один аспект, который нужно контроли-

Библиотека ГГТУ им. П.О.Сухого

ровать. Благодаря этим средствам *Windows Azure SQL Azure* является высокозащищенной управляемой платформой для приложений в облаке.

Между *SQL Server* и *SQL Azure Databases* в контексте безопасности существует список различий:

- *Microsoft SQL Server* поддерживает аутентификацию *Windows Integrated* с использованием параметров доступа из *Active Directory*; *SQL Azure Databases* поддерживает только *SQL Server Authentication*.
- *Microsoft SQL Server* и *SQL Azure Databases* используют одинаковую модель авторизации на основе пользователей и ролей, создаваемых в каждой базе данных и связываемых с логинами пользователей.
- *Microsoft SQL Server* имеет стандартные роли уровня сервера, такие как *serveradmin*, *securityadmin* и *dbcreator*, однако в *SQL Azure Databases* этих ролей нет. Вместо этого в *SQL Azure Databases* доступна роль *loginmanager* (создание логинов) и *dbmanager* (создание и управление базами данных). Эти роли могут быть привязаны к пользователям в базе данных *master*.
- Доступ к *SQL Server* и *Windows Azure SQL Databases* происходит по протоколу уровня приложения *Tabular Data Stream* (*TDS*), защищенному протоколом *SSL*, через порт *TCP*/1433. Использование *SSL* опционально для *Microsoft SQL Server* и обязательно для *Windows Azure SQL Databases*.
- В *SQL Server* контроль доступа на основе *IP* должен быть осуществлен на уровне хоста или сети, с использованием брандмауэров. В *Windows Azure SQL Databases* есть встроенный брандмауэр, ограничивающий весь доступ к серверу *Windows Azure SQL Databases* до определения клиентов компьютеров, имеющих право доступа. Брандмауэр выдает доступ, основываясь на *IP*-адресе каждого запроса.
- *SQL Server* предоставляет шифрование в режиме реального времени всех хранящихся данных на страничном уровне с использованием функциональности *Transparent Data Encryption* (*TDE*). Подобное шифрование не поддерживается в *Windows Azure SQL Databases*.

Масштабирование баз данных

Каждая база данных ограничена размером в 150 Гб (в том случае, если используется режим *Business*). Для хранения более чем 150 Гб данных разработчик должен реализовывать механизм партиционирования или шардинга данных. Шардингом называется методика, используемая разработчиками для увеличения производительности, масштабирования и снижения стоимости в тех случаях, когда, например, приложение использует модель данных с четкими критериями

Библиотека ГГТУ им. П.О.Сухого

индексирования, например, хранящие и обрабатывающие данные о продажах (дата и время). В этом случае методика шардинга может помочь оптимизировать и масштабировать решение, позволяя также параллельно обрабатывать данные – приложения могут располагать несколько партиций данных на несколько наборов вычислительных ресурсов и обрабатывать данные одновременно.

Windows Azure SQL Databases предоставляет встроенный механизм для шардинга данных – *Windows Azure SQL Databases Federations*. Подробнее про *SQL Azure Federations*: *http://msdn.microsoft.com/en-us/library/windowsazure/hh597452.aspx.*

Возрастание количества и сложности хранимых данных приводит к неизбежной проблеме нехватки ресурсов для осуществления их хранения. В сценарии, в котором важна производительность при доступе к таким данным, необходимо реализовывать сложные механизмы масштабирования, производить настройку, поддерживать работоспособность построенной инфраструктуры. *Windows Azure* предоставляет два типа хранилища, которые можно использовать в различных ситуациях и инфраструктурах (включая гибридные): *Windows Azure Storage* и *Windows Azure SQL Databases*. *Windows Azure Storage* предлагает три сервиса – таблицы, блобы и очереди, которые являются масштабируемыми, долговечными механизмами, реализующими глобально-доступное хранилище данных. *Windows Azure SQL Databases* является решением, позволяющим реализовывать более сложные сценарии хранения данных, в том числе реляционных.

Библиотека ГГТУ им. П.О.Сухого

Глава 6. Бизнес-аналитика и анализ данных с SQL Reporting и Hadoop

6.1. Обзор Business Intelligence

Бизнес-аналитика, или *Business Intelligence* (*BI*), является важным процессом, во время которого происходит интеграция и консолидация данных в масштабе организации в едином хранилище, к которому пользователи могут выполнять специальные запросы и формировать отчеты для анализа существующих данных. Фактически целью бизнес-аналитики является именно хранение данных, к которым могут обращаться пользователи, ответственные за принятие бизнес-решения на основе анализа полученных данных. Например, типичным вопросом к системе бизнес-аналитики может быть: «Какая категория продуктов является наихудшей в плане реализации в первом квартале 2013 г.?» Наиболее важными свойствами системы бизнес-аналитики являются:

- периодические операции записи в результате запросов;
- небольшое количество пользователей;
- большой размер данных, хранимых в базе данных.

Пользователи систем бизнес-аналитики занимаются генерацией отчетов, отражающих различные факторы, связанные с важными процессами внутри организации (например, финансовыми), или запросов сравнения данных. Системы бизнес-аналитики должны отслеживать историю данных, так как часто пользователи производят сопоставление данных, собранных в различные периоды времени. По этой причине объем данных в хранилище обычно очень велик и механизм хранения данных должен обеспечивать достаточное для задач пользователя и оптимальное с позиции выполнения запросов время. На данный момент существует большое количество решений для бизнес-аналитики, лидерами являются корпорации *Microsoft* и *Oracle*, предоставляющие мощные движки и механизмы – *SQL Server*, *SQL Reporting* и т. д. Локальная версия системы бизнес-аналитики от *Microsoft* называется *SQL Server Reporting Services* (*SSRS*), которая предлагает программную серверную систему создания отчетов. В *SQL Server Reporting Services* есть возможность генерировать как печатные, так и динамические отчеты, в том числе с использованием *Web*-служб. *SQL Server Reporting Services* входит в состав *Express*, *Workgroup*, *Standard* и *Enterprise* версий *Microsoft SQL Server* в каче-

Библиотека ГГТУ им. П.О. Сухого

стве устанавливаемого дополнения. Отчеты, создаваемые в *SQL Server Reporting Services*, должны быть описаны с помощью специального подъязыка *Report Definition Language* (*RDL*) и могут проектироваться при помощи *Visual Studio* с установленным дополнением *Business Intelligence Projects* либо при помощи входящего в комплект *Report Builder*. Отчеты, определенные при помощи *RDL*, могут быть созданы и сохранены в различных файловых форматах, например, *XLS*, *PDF*. Начиная с версии *SQL Server* 2008 *SQL Server Reporting Services*, предлагает возможность генерации отчетов в формате *DOC*. *RDL*-отчеты можно просматри-вать программным способом с помощью элемента управления *ASP.NET ReportViewer*, что позволяет встраивать отчеты прямо в приложение или сайт.

С развитием облачных вычислений стало возможным переносить как хранилища данных, так и системы бизнес-аналитики на мощности вендора, таким образом, оплачивая только те ресурсы, которые используются. Облачные особенности предоставляют важное преимущество эластичности и возможности обрабатывать большие массивы данных. Системы бизнес-аналитики в облаках можно условно разделить на два типа:

– данные в облаке, бизнес-аналитика локально;
– движок и приложение в облаке.

Размещение движка и отчетов в облаке позволяют размещать отчеты таким образом, что их можно встраивать как в приложение, так и иметь глобальный доступ с помощью *URL*.

К основным сценариям облачной системы бизнес-аналитики, которая называется *Windows Azure SQL Databases Reporting*, относятся следующие сценарии:

• *Сезонные нагрузки*. Например, организация, реализующая некоторые продукты, имеет периодическую задачу генерации отчетов из хранилища данных. Размещение ресурсов, на которых выполняется решение этой задачи, в локальном центре обработки данных влечет за собой необходимость в расчете максимальных нагрузок, эффективного использования и т. д. После этого в какие-то моменты осуществляется нагрузка, но большую часть времени ресурсы могут простаивать. В этом случае, когда есть периодическая необходимость генерации отчетов, может быть использован облачный сервис *Windows Azure SQL Databases Reporting*. После спада пика нагрузки, как и в случае с любыми другими сервисами, можно отказаться от оплачиваемых ресурсов.

• *Сценарий, в котором происходит разделение отчетности*

Библиотека ГГТУ им. П.О. Сухого

между несколькими партнерами, возникает, когда сгенерированную отчетность необходимо предоставить партнерской организации. В случае размещения систем аналитики в локальном центре обработки данных партнеру, который должен обеспечить доступ к отчетам, необходимо проводить конфигурацию, настройку безопасности, портов, предоставлять аккаунты и т. д. Если поместить отчеты в облако, процесс значительно упрощается – клиенты и поставщики могут получить доступ к отчетам тогда, когда им это необходимо.

• *Гибридная инфраструктура*. Часть данных в локальном центре обработки данных, часть, которую можно публично выставлять, расположена в облаке.

6.2. Создание отчетов в Report Builder

Рассмотрим последовательность действий по созданию отчетов с помощью *Report Builder* и базы данных *AdventureWorks*, предоставляемой корпорацией *Microsoft* в тестовых целях.

Создадим сервер баз данных *Windows Azure SQL Databases* с помощью портала управления *Windows Azure*.

База данных *AdventureWorks* расположена на *CodePlex* по адресу:

http://*msftdbprodsamples*.*codeplex*.*com*/*releases*/*view*/37304.

В данном примере используется версия *AdventureWorks*2012*For WindowsAzureSQLDatabase*.

После распаковки архива, содержащего базу данных, необходимо выполнить следующую команду для развертывания базы данных в *Windows Azure*:

CreateAdventureWorksForSQLAzure.cmd <servername>.database.-windows.net <username> <password>

Развертывание базы данных может занять некоторое время.

Создадим сервис *Windows Azure SQL Reporting* (рис. 6.1) с помощью портала управления *Windows Azure*. Расположение сервиса в одном регионе с созданным сервером *Windows Azure SQL Databases* значительно уменьшит задержки при их взаимодействии.

Библиотека ГГТУ им. П. О. Сухого

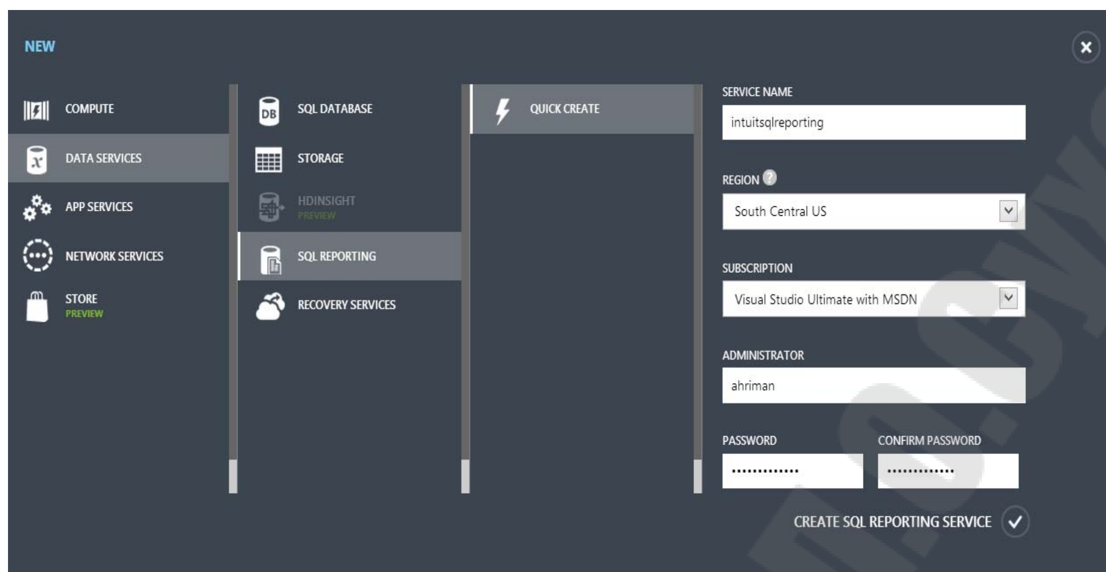


Рис. 6.1. Создание сервиса *Windows Azure SQL Reporting* с помощью портала управления *Windows Azure*

Для генерации отчетов необходимо связать сервис *SQL Reporting* с источником данных. Добавим источник данных в сервис *SQL Reporting*.

В открывшемся диалоговом окне необходимо задать логическое название источника данных и его описание. Базу данных для источника данных можно выбрать в выпадающем меню ***DATABASE*** (рис. 6.2).

CREATE DATA SOURCE

Create a data source, and then provide the credentials you want to use to connect to the data source.

DATA SOURCE NAME

AdventureWorksTestForIntuit

DATA SOURCE DESCRIPTION

Description

DATABASE

AdventureWorks2012 (Server=trj9xfuf8q, Locati

Рис. 6.2. Создание базы данных

Библиотека ГГТУ им. П.О. Сухого

На следующей странице выберем опцию *PROVIDE CREDENTIALS TO BE STORED SECURELY IN THE REPORT SERVER*. Введем логин и пароль администратора сервера баз данных (рис. 6.3).

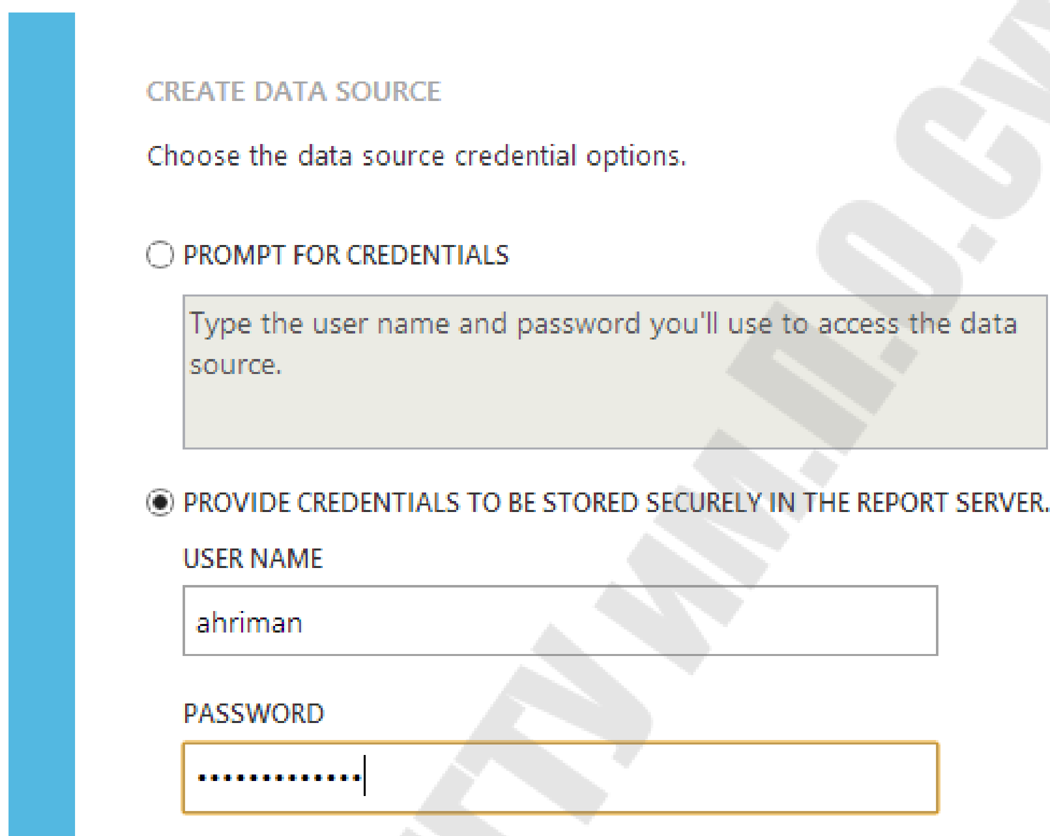


Рис. 6.3. Добавление администратора базы данных

Для взаимодействия с сервисом *SQL Reporting* установим *Report Builder*, дистрибутив которого можно загрузить с сайта *Microsoft*: *http*://*www.microsoft.com*/*ru-ru*/*download*/*details.aspx*?*id*=29072.

После запуска *Report Builder* (в зависимости от языка установки он может называться «Построитель Отчетов 3.0») подключимся к серверу, нажав «Соединение» в нижней части интерфейса.

Ссылку, которую необходимо использовать для подключения, можно скопировать на панели управления сервисом.

После ввода логина и пароля и подключения к серверу изменится статус подключения в нижней части интерфейса.

Добавим источник данных в *Report Builder* (рис. 6.4).

Библиотека ГГТУ им. П.О. Сухого

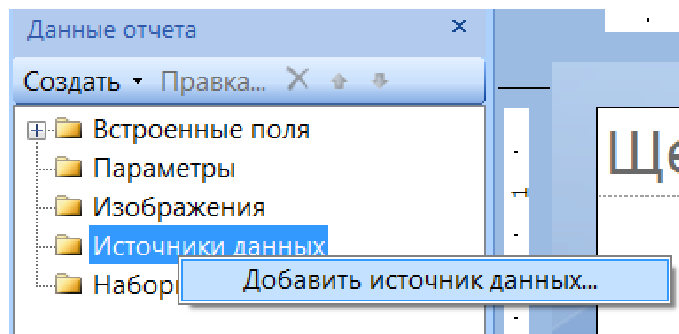


Рис. 6.4. Добавление данных в Report Builder

В открывшемся окне необходимо выбрать подключенный сервер.

Введем заголовок отчета, например, «Отчетность по продажам».

Добавим еще один источник данных, нажав правой кнопкой мыши на уже созданном источнике данных *DataSource* (рис. 6.5).

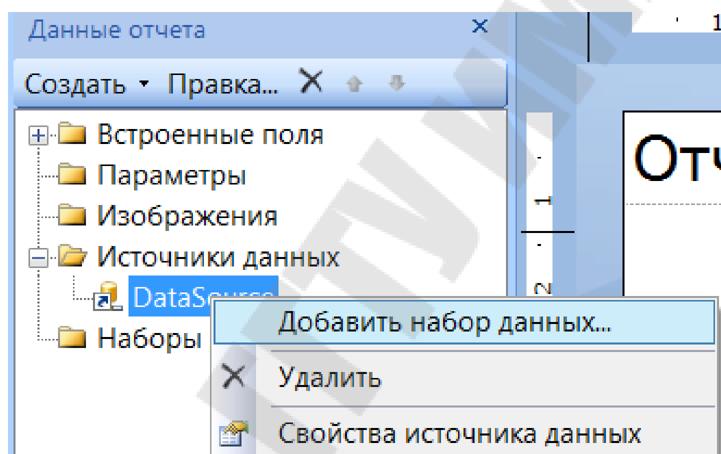


Рис. 6.5. Добавление источника данных

В открывшемся окне создания набора данных вставим в текстовое поле «Запрос» *SQL*-запрос для получения данных по продажам:

SELECT soh.[SalesOrderID],DATEPART(year, soh.[OrderDate]) AS 'Year'

,soh.[CustomerID],soh.[TerritoryID],terr.[Name] as 'TerritoryName'

,terr.[CountryRegionCode] as 'Country',soh.[TotalDue] as 'TotalSales'

FROM [Sales].[SalesOrderHeader] AS soh

JOIN [Sales].[SalesTerritory] AS terr

ON terr.[TerritoryID] = soh.[TerritoryID]

ORDER BY 'Year'

Вставим матрицу для отображения отчета.

Выберем созданный набор данных.

Библиотека ГГТУ им. П.О. Сухого

Сформируем форму отчета: перенесем *Year* в «Группы столбцов», *Country* и *TerritoryName* – в «Группы строк» и *TotalSales* – в «Значения».

Нажмем «Выполнить» для запуска формирования отчета (рис. 6.6).

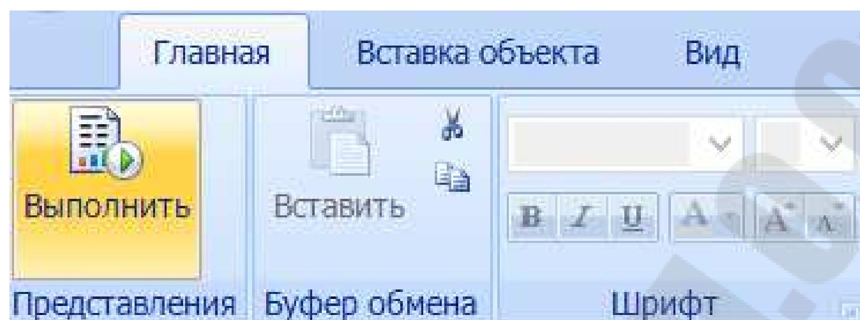


Рис. 6.6. Запуск формирования отчета

После выполнения будет выведен отчет по продажам за несколько лет (рис. 6.7).

Отчётность по продажам

Country	Territory Name	2005	2006	2007	2008	Итог
⊞ AU	Итог	1446497.1744	2380484.8387	4313294.8365	3674099.2456	11814376.0952
⊞ CA	Итог	1866734.9221	6130230.7354	6964947.2034	3437016.3271	18398929.1880
⊞ DE	Итог	262752.4184	575960.0974	2432549.8252	2208557.2345	5479819.5755
⊞ FR	Итог	199531.7230	1535232.8960	3815005.2509	2569979.4761	8119749.3460
⊞ GB	Итог	322207.5294	1602371.3205	3873251.7497	2776218.1086	8574048.7082
⊞ US	Итог	8595526.8591	22239568.5473	25772440.6803	14222327.1163	70829863.2030
Итог		12693250.6264	34463848.4353	47171489.5460	28888197.5082	123216786.1159

Рис. 6.7. Пример отчета по продажам

Сохраните отчет, перейдя обратно в **Конструктор**. Так как активное подключение принадлежит серверу в *Windows Azure*, сохранение можно произвести на него.

После сохранения любой пользователь с необходимыми правами может получить доступ к отчету, перейдя по ссылке *https://[server ame].reporting.windows.net/reportserver* и введя логин и пароль.

В сервисе *SQL Reporting* может быть несколько пользователей, имеющих доступ к сервису, которые должны быть добавлены на вкладке *Users* на панели управления сервисом.

Таким образом, облачный сервис *SQL Reporting* можно использовать для различных сценариев, подразумевающих глобальный удобный доступ к отчетам, виртуально-неограниченные ресурсы и эластичность.

Библиотека ГГТУ им.П.О.Сухого

6.3. Анализ данных с Hadoop

Сервис *Windows Azure HDInsight* (*Hadoop*) – это облачный сервис, предлагающий экосистему и создание кластеров *Hadoop* по запросу пользователя. С помощью портала *Windows Azure* могут быть созданы и развернуты кластеры *Hadoop* размером до 32 узлов (необходимо уточнить, что на момент написания сервис *Windows Azure HDInsight* находится в закрытой *Preview*-версии). Кроме создания задач *MapReduce*, разработчик имеет доступ к интерактивной консоли, которая позволяет писать запросы к данным на *JavaScript* и *Hive*.

MapReduce – это параллельное и распределенное решение, разработанное корпорацией *Google* для обработки больших массивов данных и активно использующееся в таких сферах, как, например, поисковые движки. *MapReduce*, или *M*/*R*, состоит из двух функций-компонентов – *Map* и *Reduce*. Первая функция, *Map*, используется для вычисления наборов ключ-значение. *Reduce* – функция, получающая результаты вычисления функции *Map* и применяющая к ним другую функцию. В подходе *M*/*R* подразумевается, что между данными нет прямых зависимостей, что упрощает процесс параллелизации. Узел-мастер распределяет задачи *M*/*R* по обработчикам *Map*и *Reduce*, собирая информацию. Все пары «ключ – значение» с идентичным значением ключа отправляются на обработку на один обработчик.

Типичным примером использования *MapReduce*, а значит, и *Hadoop*, является анализ файлов логов. Для файлов логов обычна ситуация разрастания до очень больших размеров, при этом все файлы соблюдают строгий синтаксис данных, что позволяет применять логику обработки к этим файлам. Несмотря на простоту, однако, обработки, если файлы логов содержат очень большое количество записей, их обработка на одном компьютере может занять долгое время.

С помощью *MapReduce* можно разделять большие файлы логов на части. Функция *Map* ищет уникальные вхождения записей (например, *Web*-страниц). Каждый раз, когда *Web*-страница находится в файле, в функцию *Reduce* поступает «ключ – значение», где ключ – *Web*-страница, а значение – 1. Обработчики функции *Reduce* агрегируют количество для каждой из *Web*-страниц, и в результате пользователь получает общее количество входов для каждой из *Web*-страниц.

Сервис основан на распределенной файловой системе *HDFS* (*Hadoop Distributed File System*), реализованной на кластере узлов *Hadoop* двух видов – *Data Node* и *Name Node*. Подобный кластер мо-

Библиотека ГГТУ им. П.О. Сухого

жет быть как гомогенной структуры (с унифицированными характеристиками узлов), так и гетерогенной (большим количеством узлов разных характеристик). *Name Nodes* содержат информацию о том, на каком из узлов данных (*Data Node*) содержится конкретная реплика (реплики используются для обеспечения высокой надежности и избыточности), и представляют клиенту эту информацию. При выходе из строя реплики фрагмента одна из его вторичных реплик назначается основной. Масштабируемость достигается за счет параллельной обработки задач. *Task Tracker* называются узлы, хранящие входные фрагменты задач (файлов) и запускающие экземпляры выполнения *Map*/*Redurce*, координирует эти экземпляры выполнения узел, носящий название *Job Tracker*. В различных реальных задачах экосистема *Hadoop*/*HDInsight* состоит из гораздо большего количества компонентов и модулей интеграции с другим программным обеспечением и форматами данных (рис. 6.8).

Рис. 6.8. Экосистема Hadoop/HDInsight

Данные, используемые *HDInsight* в задачах, хранятся в *Windows Azure Storage* в блобах и получаются для обработки по парадигме *MapReduce* узлом-мастером, который затем распределяет задачи согласно заданной логике по обработчикам (рис. 6.9). По этой причине для работоспособности кластера необходимо создать и настроить аккаунт хранилища *Windows Azure* и создать кластер максимально близко к географическому расположению хранилища (для избежания увеличения латентности).

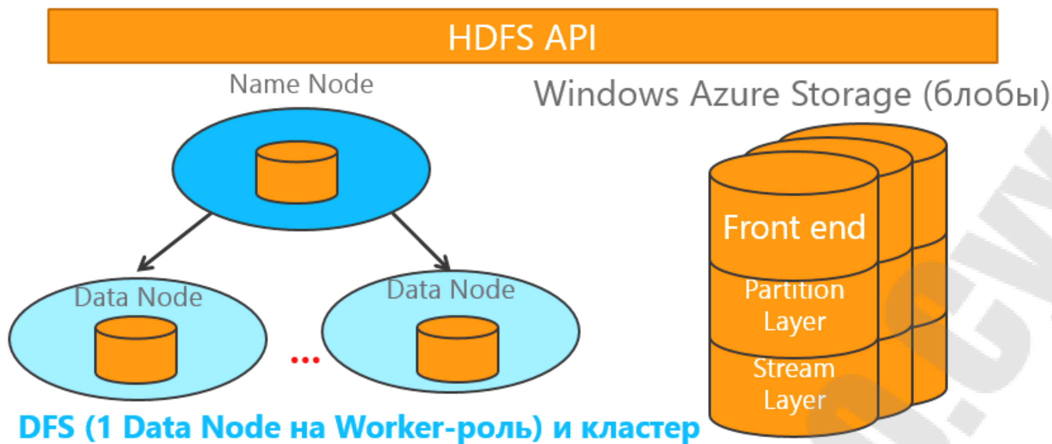


Рис. 6.9. Доступ к хранилищу *HDInsight*

Экосистема *HDInsight*

Как было изложено выше, сервис *HDInsight* предоставляет облачный сервис по управлению большими данными. *HDInsight* имеет собственные реализации *Hive* и *Pig*, которые используются для обработки данных и хранения соответственно, а также способен интегрироваться с существующими мощными средствами *BI*, разработанными *Microsoft*: *SQL Server Analysis Services*, *Reporting Services*, *PowerPivot*, *Excel* (рис. 6.10).

Говоря о том, что представляет собой *Pig*, необходимо уточнить, что *Pig* – это высокоуровневая платформа, которая обеспечивает возможность обработки больших данных в кластерах *Hadoop*. *Pig* состоит из специального языка запросов *Pig Latin*, которые выполняются к массивам данных в консоли, и способен интегрироваться *User Defined Functions* (*UDF*) на *Java*, *Python*, *C#* и *JavaScript*.

Hive является распределенным сервисом хранения данных «над» *HDFS* и предоставляет интерфейс для взаимодействия с данными с помощью *SQL* запросов на языке *HiveQL* (поддиалекте *SQL*) и реляционной модели. Аналогично *Pig*, *Hive* трансформирует запрос на собственном языке в задачи *Map*/*Reduce*.

Библиотека ГГТУ им. П.О.Сухого

Экосистема для разработчика	Hive, Pig, Mahout, Cascading, Scalding, Scoobi, Pegasus
.NET	C#, F# Map/Reduce, LINQ to Hive, .NET
JavaScript	JavaScript Map/Reduce, интерактивная консоль, Node.js
DevOps / IT Pro	PowerShell, кросс-платформенные CLI

Рис. 6.10. Экосистема Hadoop/HDInsight

Функциональность *HDInsight* гарантирует также то, что разработчик может использовать уже привычные ему средства, такие как, например, *Powershell*, *.NET*, *Java*, *F#*, *Javascript* и *Node.js* для разработки, управления и мониторинга, происходящего на кластере. Также *HDInsight* может быть установлен в его локальной версии на серверный продукт *Microsoft Windows Server*.

Использование *HDInsight*

Для того чтобы начать использование *HDInsight*, необходимо оставить заявку на портале управления, выбрать *HDInsight*, нажать на ссылку *Preview Program* и на кнопку *Try it now*.

После активации сервиса можно начать создание кластера *Hadoop*. Для этого нажмем *DATA SERVICES | HDINSIGHT | CUSTOM CREATE* (рис. 6.11).

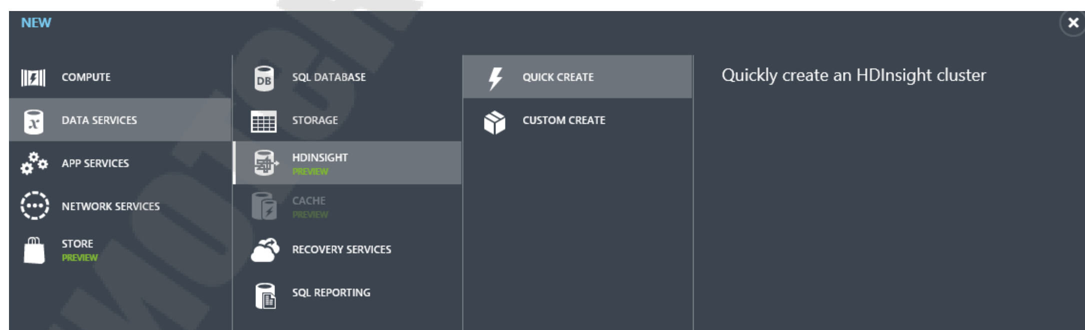


Рис. 6.11. Создание кластера HDInsight

Введем название нового кластера, все остальные настройки оставим по умолчанию.

Библиотека ГГТУ им. П.О.Сухого

На следующей странице введем учетные данные администратора.

На следующей странице выберем опцию создания нового аккаунта хранилища и введем соответствующие данные.

После создания кластера нам станет доступна панель управления кластером.

Из панели управления можно инициировать сессию удаленного доступа (*CONNECT*), внутри виртуальной машины создать задачу, посмотреть историю выполнения задач, изучить информацию по *Hadoop*. Кнопка *Manage Cluster* позволяет контролировать размер использованного дискового пространства, а также задать папки в *Windows Azure* блобах, которые можно рассматривать как хранилище (*Azure Storage Vault*).

Войдем на панель управления непосредственно кластером, нажав *MANAGE CLUSTER*. Введем учетные данные, настроенные при создании кластера.

Создадим задачу *Map*/*Reduce* из тестового набора. Для этого нажмем кнопку *Samples* (рис. 6.12).

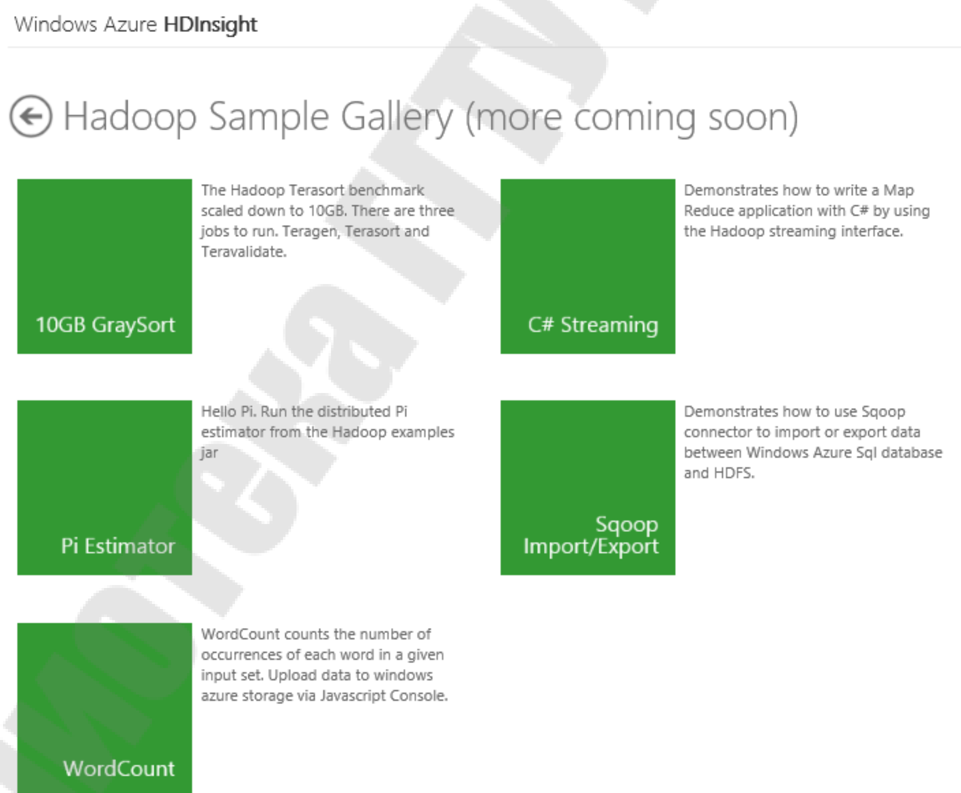


Рис. 6.12. Набор тестовых задач *HDInsight*

Библиотека ГГТУ им. П.О.Сухого

Нажмем кнопку *Pi Estimator*. На странице тестовой задачи нажмем *Deploy to your cluster*. Это приведет к странице, которую можно также вызвать, нажав кнопку *Create Job*. Будет открыта страница добавления новой задачи (рис. 6.13), на которой необходимо ввести название задачи, указать *JAR*-файл задачи и настроить параметры (если такие есть). В случае тестовой задачи эти данные заполняются автоматически.

Рис. 6.13. Добавление задачи на расчет

Задача, которая будет запущена, инициирует расчет числа Пи, исходя из 16 *Maps* (первый параметр в команде) и 10 млн сэмплов на каждый *Map* (второй параметр).

Нажмем *Execute Job* и дождемся выполнения задачи. После окончания выполнения будут выведены результаты и служебная информация о задаче (рис. 6.14).

Библиотека ГГТУ им.П.О.Сухого

Command

```
c:\apps\dist\hadoop-1.1.0-SNAPSHOT\bin\hadoop.cmd jar c:\apps\Jobs\templates\635153786227414232.hadoop-
examples.jar pi 16 10000000
```

Output (stdout)

```
Number of Maps = 16
Samples per Map = 10000000
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Wrote input for Map #8
Wrote input for Map #9
Wrote input for Map #10
Wrote input for Map #11
Wrote input for Map #12
Wrote input for Map #13
Wrote input for Map #14
Wrote input for Map #15
Starting Job
Job Finished in 44.654 seconds
Estimated value of Pi is 3.14159155000000000000
```

Рис. 6.14. Результат выполнения задачи по вычислению числа Пи

Воспользуемся интерактивной консолью для расчета скрипта на *Javascript* (*Map*/*Reduce*). В качестве примера возьмем тестовую задачу по подсчету слов. Для этого загрузим скрипт *WordCount.js* и текстовый файл *davinci.txt*, зайдя на страницу *Samples* и нажав на кнопку *WordCount*.

Вернемся на главную страницу панели управления кластером и нажмем на кнопку *Interactive Console*. Обратите внимание, что ту же задачу можно выполнить, нажав *Deploy to your cluster*.

Введем в интерактивную консоль команду *fs.put*() для загрузки файла *WordCount.js*. Выберем загруженный локально файл. Значение *Destination* укажем равным *./WordCount.js/*. Повторим процедуру для загрузки файла *davinci.txt*. Значение *Destination* для *davinci.txt* укажем равным *./example/data/*.

Выполним команду, указанную ниже, и, после выполнения задачи, нажмем *View Log* для просмотра информации о задаче.

pig.from("/example/data/davinci.txt").mapReduce("WordCount.js",
"word, count:long").orderBy("countDESC").take(10).
to("*DaVinciTop*10*Words*")

Увидеть результаты можно, введя команду *fs.read*("*DaVinciTop 10Words*").

Библиотека ГГТУ им. П.О. Сухого

С появлением тенденции быстрого увеличения количества данных, существующей в сегодняшнем мире, и распространением термина *Big Data* (Большие данные), локальные центры, которые часто не могут покрыть потребности в обработке все возрастающих массивов данных, могут быть как заменены, так и дополнены (в зависимости от сценариев) ресурсами, хранящимися в облаке, для того, чтобы оптимизировать затраты и увеличить эффективность производства.

Библиотека ГГТУ им. П.О. Сухого

Глава 7. Доступ к сервисам предприятия с Windows Azure Service Bus

Сервис *Windows Azure Service Bus* представляет собой интеграционную шину (*Middleware*) с функциональностью переноса, безопасного обмена сообщениями и создания распределенных и слабосвязанных приложений в облаке, а также гибридных приложений, размещенных одновременно в частных и общедоступных облачных службах. *Service Bus* – это программная прослойка, которая позволяет интегрировать некоторые сущности, находящиеся внутри предприятия, и внешних клиентов. *Windows Azure Service Bus* является одним из вспомогательных компонентов облачной платформы *Windows Azure* (рис. 7.1), и основная его роль как вспомогательного компонента – помощь в реализации сложных сценариев взаимодействия, например, таких как создание безопасного канала коммуникаций между web-сервисом внутри *NAT* или за брандмауэром и глобально-доступного сервиса.

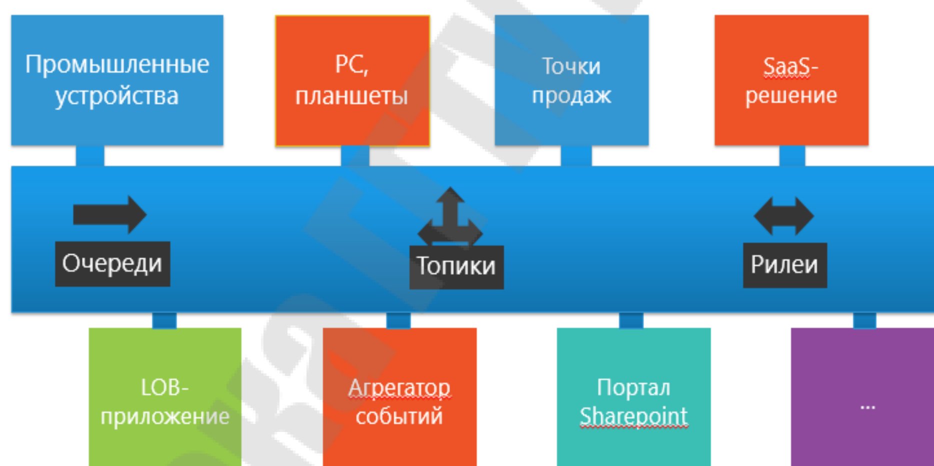


Рис. 7.1. Сервисы Windows Azure Service Bus

Когда рассматриваются интеграционная шина и обмен сообщениями, нельзя обойти вниманием один из основополагающих принципов построения распределенных систем, который называется связанностью. Суть этого принципа заключается в том, что выход из строя или возникновение неполадки в одной из частей системы не должна приводить к неполадкам или выходу из строя других частей или всей системы.

Рассмотреть применение этого принципа на практике можно на примере приложения, состоящего из трех частей – *Web*-интерфейса, с

Библиотека ГГТУ им. П.О. Сухого

помощью которого пользователи оставляют заказы, сервиса трекинга заказов и сервиса, осуществляющего доставку заказов.

В случае тесной связанности (рис. 7.2) заказы хранятся и вращаются внутри двух сервисов – трекинга и логистики. Если один из этих сервисов выйдет из строя, то это затронет не только его работу, но и работу других частей системы. Если выйдет из строя сервис логистики, то сервис трекинга не сможет переправлять заказы от пользователей (если внутри сервиса трекинга не реализован механизм брокера). Если выйдет из строя сервис трекинга, то заказы пользователей не смогут попасть в систему в принципе – при каждом запросе пользователь будет получать ошибки.

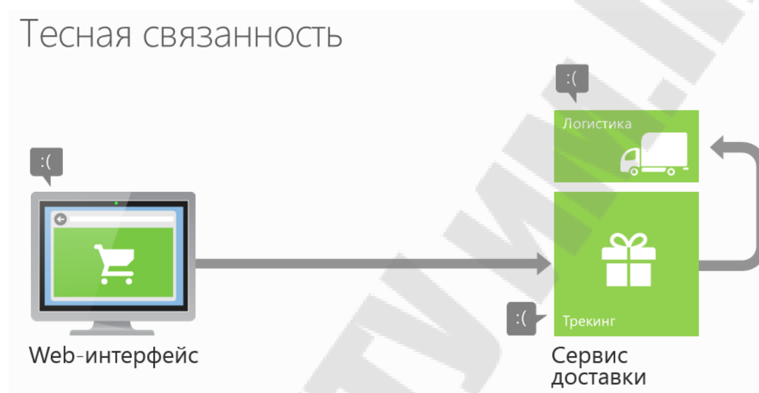


Рис. 7.2. Пример связанности

Для того чтобы решить проблемы тесной связанности в данном сценарии, необходимо интегрировать в приложение еще одну часть, которая будет отвечать за хранение сообщений. Этой частью может стать механизм очереди, который рассматривался в главе 5, посвященной хранилищу *Windows Azure*. *Windows Azure Service Bus* предоставляет механизм очередей с дополнительной функциональностью (рис. 7.3).

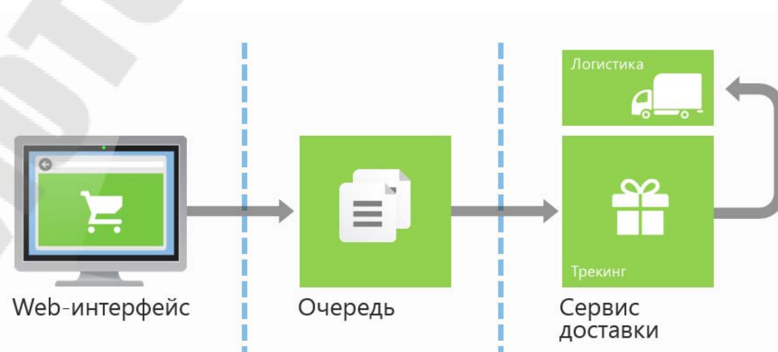


Рис. 7.3. Пример распределенной архитектуры

Библиотека БГУИР / ГГТУ им. П.О. Сухого

Добавлением в приложение очереди будет решена проблема доставки сообщений – даже в случае полного выхода из строя серверной части приложения (сервисов трекинга и логистики) сообщения пользователей не будут потеряны – они будут храниться в очереди до тех пор, пока не истечет время их жизни либо не станет доступна серверная часть.

Windows Azure Service Bus оперирует тремя основными понятиями – очередями, топиками и рилеями. Основа любой сервисной шины и *Windows Azure Service Bus* в том числе – создание пула сообщений и передача их тем клиентам, которым они интересны и для которых они предназначены. Соответственно, с помощью сервисной шины разработчик может реализовать высокомасштабируемые сценарии по осуществлению коммуникаций, подключить необходимое количество клиентов и т. д. Возможными клиентами сервисной шины могут быть *SaaS*-решения, десктопные клиенты, порталы, железные устройства типа телефонов, промышленные, клиенты с переменным доступом к сети, которым необходимо собрать данные и потом отправить их в главный офис. Например, в случае клиентов с переменным доступом к сети сообщения накапливаются в кэше у клиента и, когда появляется подключение, сообщения отправляются в базу.

Первым компонентом *Windows Azure Service Bus* является рилей-маршрутизатор, который позволяет реализовать сценарий «прохода» запросов через *NAT*/брандмауэр и вынести сервис наружу. Это типичный сценарий, в котором за брандмауэром в корпоративном центре обработки данных находится сервис, который нужно вынести наружу без нарушений регламентов безопасности. С помощью рилея этот сценарий решается наиболее безопасным образом.

Второй компонент *Windows Azure Service Bus* – это очереди. Очереди *Windows Azure Service Bus* – это сервис, реализующий стандартную абстракцию программирования и позволяющий обеспечить работу с переменным доступом к сети, и в то же время создать приложения с выравниванием нагрузки, при получении большого количества сообщений разработчик может выделить несколько очередей и балансировать таким образом нагрузку.

Последним компонентом являются топики и подписки (рис. 7.4). Этот компонент необходим для решения сценария распространения сообщений среди нескольких клиентов согласно заданным разработчиком правилам. Например, разработчик может создать подписку, подписать на нее определенный набор клиентов, устройств, после чего сообщения, которые придут в эту подписку, будут поступать только этим клиентам.

Библиотека ГГТУ им. П.О.Сухого

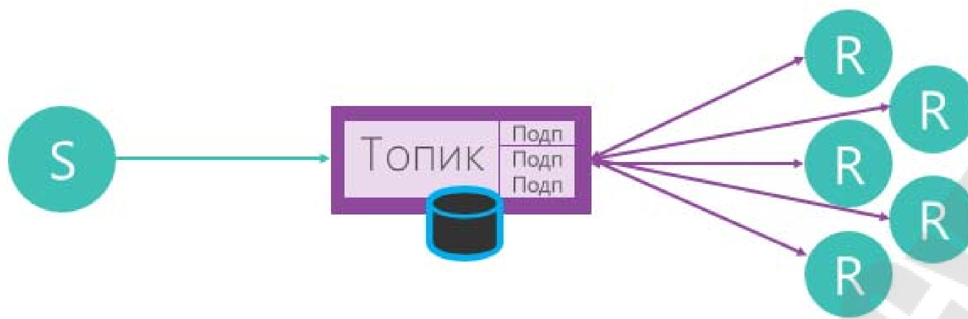


Рис. 7.4. Топики и подписки

Более сложным сценарием использования топиков и подписок является маршрутизатор на базе контента, когда нужно направить сообщение не всем подряд, а передать ее конкретным клиентам (рис. 7.5). Для этого существуют фильтры, на основании которых сообщения фильтруются и направляются в соответствующие очереди. Фильтры могут иметь один из трех типов: *True*/*NotTrue* (когда выполняется или не выполняется заданное условие), *SQL*-фильтр (когда фильтр задается в *SQL*-синтаксисе) и фильтр *CorrelationId* (когда фильтр отрабатывает, учитывая идентификатор корреляции каждого сообщения).

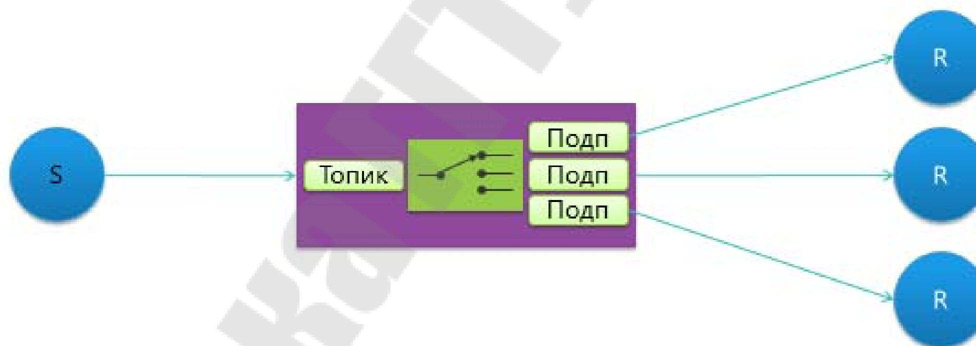


Рис. 7.5. Маршрутизатор на базе контента

Сообщения в очередях и топиках уничтожаются, если у них истекает срок *TTL* – этот механизм помогает защититься от передачи и получения устаревшей информации. Например, если с очень коротким промежутком передаются сообщения с актуальными только некоторый небольшой промежуток времени данными, решение, не удаляющее устаревшие сообщения, может привести к непрогнозируемому поведению системы.

Библиотека ГГТУ им.П.О.Сухого

7.1. Сценарии использования Windows Azure Service Bus

Первый распространенный сценарий – это *Web*-сервис, например, это *Web*-сервис, обрабатывающий бизнес-логику работы ритейлера.

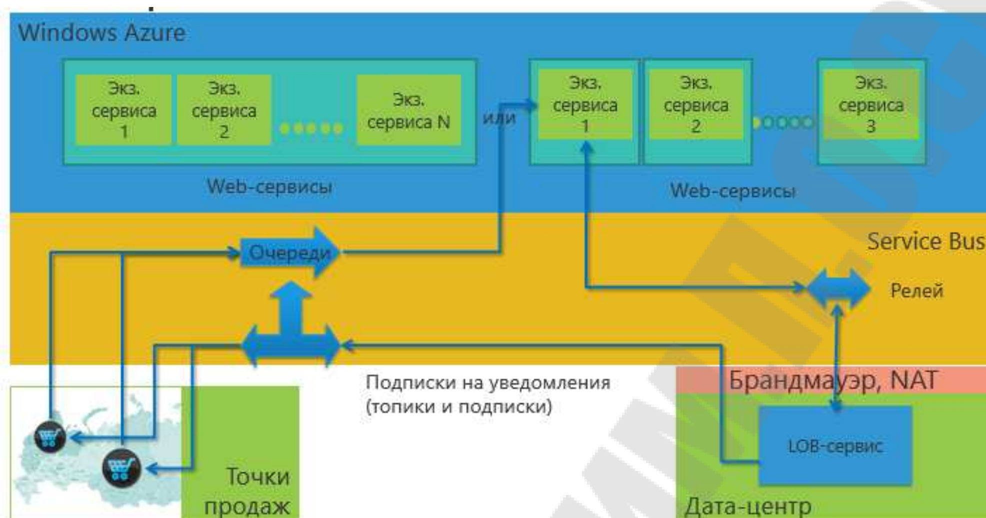


Рис. 7.6. Windows Azure Service Bus:
сценарий – управление очередьми заказов

Для решения интеграционной задачи объединения и доставки сообщений по неограниченно большому количеству мобильных клиентов, устройств и точек продаж будут использованы все компоненты *Windows Azure Service Bus*: очереди – для управления очередями заказов. Каждый из агентов, точек продаж, устройств и клиентов отправляет сообщение в очередь о формировании заказа, далее сообщение обрабатывается сервисом, и эти сервисы, обрабатывая заказы, могут обращаться к корпоративному приложению, находящемуся за брандмауэром в корпоративном центре обработки данных, управляющем данными, которые нельзя выставлять на внешний мир. Для решения этой задачи используются безопасные рилеи *Windows Azure Service Bus*. Для уведомления клиентов о различных действиях системы – например, скидках и так далее – используются топики и подписки.

Второй сценарий – специализированный сервис, провайдер вычислительных мощностей для обработки данных по запросу (рис. 7.7). Пользователи отправляют данные для задачи в облачное хранилище, формируя таким образом очередь задач и сообщение для контроллера (контроллером может выступать *Cloud Service*), принимающего сообще-ние и решающего, что надо создать новую задачу для обработки

Библиотека ГГТУ им. П. О. Сухого

данных. Используя сообщения с низкой латентностью внутри инфраструктуры *Windows Azure*, он обращается к системе и выделяет некоторые мощности. Для мониторинга состояния задач используются подписки, и каждый экземпляр обработчика регулярно сообщает о статусе обработки задачи, клиенты могут подписаться на эти подписки. На эти подписки подписан также контроллер, на основании данных откуда он может принимать решения. Решение работает следующим образом: пользователь отправляет задачи через клиентское приложение; задачи обрабатываются в *HPC*-стиле (распределенном и параллельно обрабатывающемся несколькими обработчиками) на *Windows Azure*; пользователи могут следить за прогрессом и получать уведомления.

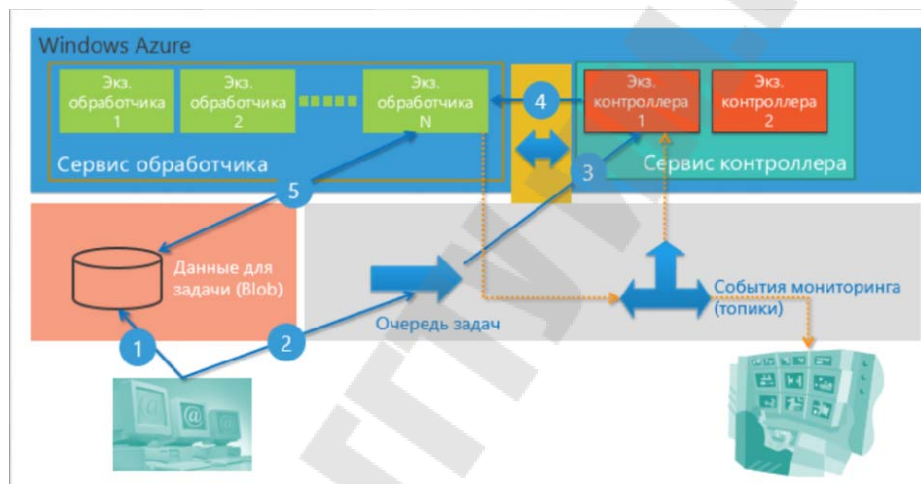


Рис. 7.7. Специализированный сервис, провайдер вычислительных мощностей для обработки данных по запросу

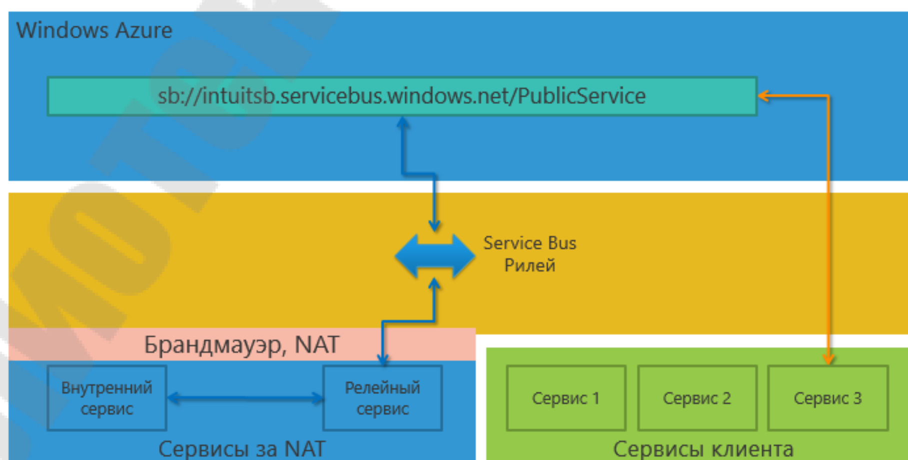


Рис. 7.8. Предоставление доступа к внутреннему корпоративному сервису внешним клиентам

Библиотека ГГТУ им. П.О.Сухого

Третий сценарий – это предоставление доступа к внутреннему корпоративному сервису внешним клиентам, вендорам, устройствам (рис. 7.8). Например, есть данные о сотрудниках, которые необходимо передать в государственное учреждение. Для этого может быть создан рилейный сервис, промежуточный сервис, который будет оперировать с внешним миром. Рилейный сервис, обращаясь к рилею *Windows Azure Service Bus*, создает новый сервис, который находится по этому *URL* (заглушке) и будет использоваться внешними клиентами для обращения к данным за *NAT*. Теперь клиенты могут обращаться к корпоративному сервису или данным по урлу, с помощью которого эти запросы будут безопасно транслироваться в корпоративную закрытую сеть.

7.2. Windows Azure Notification Hubs

Notification Hubs – это сервис, который помогает разработчику, решая проблему управления *Push*-уведомлениями. Вместо реализации собственных механизмов управления разработчик может воспользоваться *Notification Hubs* и получить инфраструктуру, обеспечивающую поддержку следующих сценариев:

- *Мультиплатформенность*. Сервис *Notification Hubs* объединяет в себе функциональность, необходимую для рассылки уведомлений на популярные платформы (*Windows* 8, *Windows Phone* 8, *iOS*, *Android*), и автоматизирует работу по их формированию и рассылке.
- *Рассылка уведомлений на основе правил*. Каждый объект, подписанный на *Notification Hub*, может использовать в своей работе один или более тегов-маркеров, позволяющих хабу определить, куда необходимо рассылать уведомления и кто является заинтересованным в этих уведомлениях, а кто их получать не должен.
- *Масштабирование*. *Notification Hubs* способны реализовать систему, рассылающую уведомления на миллионы подключенных устройств без необходимости внесения изменений в архитектуру системы.

Таким образом, можно отметить, что *Notification Hubs* являются аналогичным, но более мощным механизмом, по сравнению с *Windows Azure Mobile Services Push Notifications*. Безопасность проводимых с *Notification Hub* операций обеспечивается сервисом *Windows Azure Access Control Service*, который позволяет регламентировать доступ к операциям на основе трех основных прав: *Listen* (прослушивание), *Send* (отправка) и *Manage* (управление).

Библиотека ГГТУ им. П.О.Сухого

Что касается регистраций устройств, то необходимо учитывать, что регистрации имеют время жизни, которое может быть установлено максимум в 90 дней, после чего регистрации должны быть обновлены.

Транзакции в *Windows Azure Service Bus*

Windows Azure Service Bus предоставляет возможность сущностям *Windows Azure Service Bus* участвовать в транзакциях, что позволяет разработчикам выполнять несколько операций в пределах одной транзакций, и быть уверенными в том, что все действия будут выполнены либо, если возникнет ошибка, ни одно из этих действий не будет выполнено. В контексте выполнения в транзакции не поддерживается операция *Abandon*, необходимая для отказа от сообщений от обработки. Все остальные операции могут выполняться в пределах транзакции. Реализацией транзакционного механизма занимается *Windows Azure SQL Databases*, которая лежит в основе хранилища для *Windows Azure Service Bus*.

7.3. Определение дубликатов сообщений

Встроенный механизм автоматического определения дубликатов сообщений позволяет избавиться от проблемы хранения одинаковых сообщений, которые могут привести к непрогнозируемому результату. Дубликаты сообщений могут возникать в тех ситуациях, когда разработчик в ответ на ошибку принятия сервером сообщения (например, по тайм-ауту) реализует логику повторной отправки сообщений. Однако может возникнуть ситуация, когда сообщение дошло до сервера, но тем не менее была выдана какая-то ошибка. Тогда возникают дубликаты сообщений.

Автоматическое определение дубликатов сообщений по умолчанию выключено, так как его использование может привести к серьезным нагрузкам: этот механизм использует в своей основе идентификаторы сообщения и заданный разработчиком временной промежуток хранения этих идентификаторов. Чем больше временной промежуток, тем больше дополнительных данных хранится. В системах, оперирующих миллионами сообщений, накладываемая нагрузка может быть существенной:

```
namespaceManager.CreateQueue(
 new QueueDescription(queueName)
 {
```


Библиотека ГГТУ им. П.О.Сухого

```
 RequiresDuplicateDetection = true,
DuplicateDetectionHistoryTimeWindow = TimeSpan.FromHours(1)
 });
```

Windows Azure Service Bus – это масштабируемый облачный сервис сообщений, доступный по требованию и являющийся совершенным средством интеграции корпоративных автоматизированных систем и сервисов. К компонентам *Windows Azure Service Bus* относятся рилеи, очереди, топики и подписки. *Windows Azure Service Bus* может помочь решить множество различных интеграционных задач, например:

– реализацию промышленных архитектурных паттернов;
– интеграцию компонент инфраструктуры и облака (гибридный сценарий);
– интеграцию своих и сторонних компонент;
– реализацию сценариев распределенных систем любой сложности.

7.4. Шаблон проектирования каналов и фильтров

С использованием *Windows Azure Service Bus* можно реализовать шаблон проектирования каналов и фильтров (*Pipes and Filters pattern*).

Данный шаблон позволяет разбить задачу, которая выполняет сложную обработку, на ряд отдельных элементов для повторного использования при необходимости. Так можно повысить производительность, масштабируемость и возможность многократного использования, позволяя развертывать и масштабировать элементы задачи, которые выполняют обработку, независимо друг от друга.

Контекст и проблема

Приложение должно выполнять различные задачи определенной сложности с информацией, которую оно обрабатывает. Для осуществления такой обработки в виде неделимого модуля применяется простой, но гибкий способ реализации такого приложения. Однако этот метод, вероятно, сократит возможности для рефакторинга кода, его оптимизации или повторного использования, если части одной и той же обработки потребуются в другом месте приложения.

Приложение получает и обрабатывает данные из двух источников. Данные из каждого источника обрабатываются отдельным модулем, который выполняет ряд задач для преобразования этих данных, прежде чем передавать результат в бизнес-логику приложения.

Библиотека БГТУ им. П.О. Сухого

Функции некоторых задач, выполняемых неделимыми модулями, очень похожи, но модули разрабатываются отдельно. Код, который реализует задачи, тесно связан в модуле. Он незначительно или вообще не учитывает задачи масштабируемости и многократного использования.

Тем не менее задачи обработки, выполняемые каждым модулем, или требования к развертыванию для каждой задачи, могут изменяться в соответствии с бизнес-требованиями. Некоторые задачи могут быть ресурсоемкими и более эффективными на мощном оборудовании. В то время как для других задач такие дорогостоящие ресурсы могут не требоваться. Кроме того, в будущем может предполагаться дополнительная обработка или может измениться порядок выполнения задач по обработке. Требуется такое решение, которое устранит эти проблемы и увеличит возможность многократного использования кода.

Решение

Разбейте процесс обработки, требуемый для каждого потока, на ряд отдельных компонентов (или фильтров), каждый из которых выполняет определенную задачу. За счет стандартизации формата данных, которые получает и отправляет каждый компонент, эти фильтры можно объединить в конвейер. Так предотвращается дублирование кода и упрощается удаление, замена или интеграция дополнительных компонентов при изменении требований к обработке.

Время, необходимое для обработки одного запроса, зависит от скорости самого медленного фильтра в конвейере. Один или несколько фильтров могут быть сдерживающим фактором, особенно если в потоке из определенного источника данных содержится много запросов. Ключевое преимущество конвейерной структуры заключается в том, что в ней можно параллельно выполнять экземпляры медленных фильтров, позволяя распределить нагрузку и повысить пропускную способность в системе.

Составляющие конвейер фильтры могут выполняться на разных компьютерах, что позволяет масштабировать их независимо друг от друга и использовать возможности эластичности, которые предоставляются в большинстве облачных сред. Ресурсоемкие фильтры могут выполняться на оборудовании с высоким уровнем производительности, а другие, менее ресурсоемкие, могут размещаться на более дешевом стандартном оборудовании. Фильтры не должны находиться в одном центре обработки данных или географическом расположении.

Библиотека ГГТУ им. П.О. Сухого

Это позволит каждому элементу в конвейере работать в среде, близкой к необходимым ресурсам.

Если входные и выходные данные фильтра структурированы в виде потока, обработка для каждого фильтра может выполняться в параллельном режиме. Первый фильтр в конвейере может начать свою работу и вернуть результаты, которые непосредственно передаются следующему фильтру в последовательности, прежде чем первый фильтр завершит свою работу.

Еще одно преимущество, предоставляемое этой моделью, – устойчивость. Если произойдет сбой фильтра или компьютер, на котором он выполняется, станет недоступен, конвейер позволит перепланировать операции, выполняемые фильтром и передать их другому экземпляру компонента. Сбой одного фильтра не обязательно приводит к сбою всего конвейера.

Альтернативный подход к реализации распределенных транзакций – использовать шаблон каналов и фильтров в сочетании с шаблоном компенсирующих транзакций. Распределенные транзакции можно разделить на отдельные компенсируемые задачи, каждую из которых можно реализовать с помощью фильтра, также реализуемого с помощью шаблона компенсирующих транзакций. Фильтры в конвейере можно реализовать в виде отдельно размещенных задач, выполняемых в непосредственной близости к данным, которыми они управляют.

Проблемы и рекомендации

При выборе схемы реализации этого шаблона следует учитывать следующие особенности:

• *Сложность.* Повышенная гибкость, которую обеспечивает этот шаблон, может также вызвать сложности, особенно если фильтры в конвейере распределяются между разными серверами.

• *Надежность.* Используйте инфраструктуру, которая гарантирует сохранность данных, передаваемых между фильтрами в конвейере.

• *Идемпотентность.* Если после получения сообщения происходит сбой фильтра в конвейере и операция переносится на другой экземпляр фильтра, часть операции может быть уже выполнена. Если эта операция обновляет некоторые аспекты глобального состояния (например, сведения, хранящиеся в базе данных), это обновление можно повторить. Такая же проблема может произойти после передачи результатов фильтра следующему фильтру в конвейере, но до того, как появится сообщение об успешном завершении работы фильтра. В

Библиотека ГГТУ им. П.О. Сухого

этих случаях другой экземпляр фильтра может повторить эти операции, что приведет к тому, что одни и те же результаты будут переданы дважды. В результате следующие фильтры в конвейере могут дважды обработать одни и те же данные. Поэтому фильтры в конвейере должны разрабатываться идемпотентными. (Дополнительные сведения смотри в описании шаблонов идемпотентности в блоге Джонатана Оливера (*Jonathan Oliver.*))

• *Повторяющиеся сообщения.* Если сбой фильтра в конвейере происходит после публикации сообщения для следующего этапа конвейера, может запуститься другой экземпляр фильтра и опубликовать копию этого сообщения в конвейере. Это может привести к передаче следующему фильтру двух экземпляров одного сообщения. Чтобы избежать этого, конвейер должен обнаруживать и удалять дублирующие сообщения.

Если реализуется конвейер с использованием очередей сообщений (например, очередей служебной шины *Microsoft Azure*), в инфраструктуре очередей сообщений может быть предусмотрено автоматическое обнаружение и удаление дубликатов сообщений.

• *Контекст и состояние.* В конвейере каждый фильтр по сути выполняется изолированно. Способ его вызова не должен вызывать предположения. Это означает, что каждому фильтру нужно предоставить достаточный контекст для выполнения операций. Этот контекст может содержать много информации о состоянии.

Рассмотрим варианты использования этого шаблона:

– процесс обработки, требуемый приложением, можно легко разделить на ряд независимых этапов;

– у этапов обработки, которые выполняются приложением, разные требования к масштабируемости;

– требуется определенная гибкость, чтобы изменять порядок шагов обработки, выполняемых приложением, или добавлять и удалять шаги;

– система будет работать эффективнее, если распределить шаги обработки между несколькими серверами;

– чтобы свести к минимуму последствия сбоя на шаге при обработке данных, требуется надежное решение.

Этот шаблон может оказаться неэффективным в следующих случаях:

– шаги обработки, выполняемые приложением, не являются независимыми или должны выполняться вместе в рамках одной транзакции.

Библиотека ГГТУ им. П.О.Сухого

– объем контекста или сведений о состоянии, необходимых для выполнения шага, делают такой подход неэффективным. Вместо этого можно сохранить информацию о состоянии в базе данных.

Пример

Чтобы предоставить инфраструктуру, необходимую для реализации конвейера, можно использовать последовательность очередей сообщений. В начальную очередь поступают необработанные сообщения. Компонент, реализованный в виде задачи фильтра, ожидает передачи сообщения в очереди, выполняет свою операцию и затем передает преобразованное сообщение в следующую очередь в последовательности. Другая задача фильтра может ожидать передачи сообщения в этой очереди, обрабатывать их и отправлять результаты в другую очередь. И так до тех пор пока в последнем сообщении в очереди не появятся полностью преобразованные данные.

Можно использовать очереди служебной шины, чтобы создать надежный и масштабируемый механизм организации очереди. В примере класса *ServiceBusPipeFilter* на языке *C#* ниже показано, как можно реализовать фильтр, который получает входящие сообщения из очереди, обрабатывает их и передает результаты в другую очередь.

```
public class ServiceBusPipeFilter
{
 ...
 private readonly string inQueuePath;
 private readonly string outQueuePath;
 ...
 private QueueClient inQueue;
 private QueueClient outQueue;
 ...
 public ServiceBusPipeFilter(..., string inQueuePath, string outQueuePath = null)
 {
 ...
 this.inQueuePath = inQueuePath;
 this.outQueuePath = outQueuePath;
 }
 public void Start()
 {
 ...
```


Библиотека ГГТУ им. П.О. Сухого

```
this.outQueue = QueueClient.CreateFromConnectionString(...);
...
this.inQueue = QueueClient.CreateFromConnectionString(...);
}
public void OnPipeFilterMessageAsync(
Func<BrokeredMessage, Task<BrokeredMessage>> asyncFilter
Task, ...)
{
...
this.inQueue.OnMessageAsync(
async (msg) =>
{
...
var outMessage = await asyncFilterTask(msg);
if (outQueue != null)
{
await outQueue.SendAsync(outMessage);
}
},
options);
}
public async Task Close(TimeSpan timespan)
{
this.pauseProcessingEvent.Reset();
Thread.Sleep(timespan);
this.inQueue.Close();
...
}
...
}
```

Метод *Start* в классе *ServiceBusPipeFilter* позволяет подключиться к паре очередей ввода и вывода, а метод *Close* – отключиться от очереди ввода. Метод *OnPipeFilterMessageAsync* выполняет фактическую обработку сообщений, а параметр *asyncFilterTask* для этого метода указывает, какую именно обработку следует выполнить. Метод *OnPipeFilterMessageAsync* ожидает поступления сообщений в очередь ввода, выполняет код, определяемый параметром *asyncFilterTask* для каждого сообщения по мере их поступления, и передает результаты в очередь вывода. Сами очереди определяются с помощью конструктора.

Библиотека ГГТУ им.П.О.Сухого

В примере решения фильтры реализуются в наборе рабочих ролей. Каждую рабочую роль можно масштабировать автономно в зависимости от сложности обработки коммерческих данных или ресурсов, необходимых для обработки. Кроме того, можно параллельно выполнять несколько экземпляров каждой рабочей роли, чтобы повысить пропускную способность.

В следующем коде показана рабочая роль *Azure* с именем *PipeFilterARoleEntry*, определенная в проекте *PipeFilterA* в примере решения.

```
public class PipeFilterARoleEntry : RoleEntryPoint
{
 ...
 private ServiceBusPipeFilter pipeFilterA;
 public override bool OnStart()
 {
 ...
 this.pipeFilterA = new ServiceBusPipeFilter(
 ...,
 Constants.QueueAPath,
 Constants.QueueBPath);
 this.pipeFilterA.Start();
 ...
 }
 public override void Run()
 {
 this.pipeFilterA.OnPipeFilterMessageAsync(async (msg) =>
 {
 var newMsg = msg.Clone();
 await Task.Delay(500); // DOING WORK
 Trace.TraceInformation("Filter A processed message:{0} at {1}",
 msg.MessageId, DateTime.UtcNow);
 newMsg.Properties.Add(Constants.FilterAMessageKey, "Complete");
 return newMsg;
 });
 ...
 }
 ...
}
```


Библиотека ГГТУ им. П.О. Сухого

Эта роль содержит объект *ServiceBusPipeFilter*. Метод *OnStart* в роли позволяет подключиться к очередям для получения входных сообщений и передачи выходных сообщений (имена очередей определены в классе *Constants*). Метод *Run* вызывает метод *OnPipeFilterMessagesAsync* для обработки каждого полученного сообщения. (В этом примере обработка имитируется ожиданием в течение короткого периода времени.) В завершение обработки создается новое сообщение, содержащее результаты (в этом случае во входящее сообщение добавлено пользовательское свойство), и это сообщение передается в очередь вывода.

Пример кода содержит еще одну рабочую роль с именем *PipeFilterBRoleEntry* в проекте *PipeFilterB*. Эта роль аналогична роли *PipeFilterARoleEntry* за исключением того, что она выполняет другие операции обработки в методе *Run*. В примере решения эти две роли объединяются для создания конвейера, где очередь вывода для роли *PipeFilterARoleEntry* является очередью ввода для роли *PipeFilter BRoleEntry*.

В примере решения также предоставлены две дополнительные роли с именами *InitialSenderRoleEntry* (в проекте *InitialSender*) и *FinalReceiverRoleEntry* (в проекте *FinalReceiver*). Роль *InitialSenderRoleEntry* предоставляет начальное сообщение в конвейере. Метод *OnStart* позволяет подключиться к одной очереди, а метод *Run* передает метод в эту очередь. Эта очередь является очередью ввода, которая используется ролью *PipeFilterARoleEntry*, поэтому при передаче сообщения в нее это сообщение принимается и обрабатывается ролью *PipeFilterARoleEntry*. Затем обработанное сообщение передается через роль *PipeFilterBRoleEntry*.

Очередь ввода для роли *FinalReceiveRoleEntry* является очередью вывода для роли *PipeFilterBRoleEntry*. Метод *Run* в роли *FinalReceiveRoleEntry*, приведенный ниже, получает сообщение и выполняет некоторые задачи окончательной обработки. Затем он записывает значения пользовательских свойств, добавленных фильтрами в конвейере, чтобы выполнить трассировку выходных данных.

```
public class FinalReceiverRoleEntry : RoleEntryPoint
{
 ...
 private ServiceBusPipeFilter queueFinal;
 public override bool OnStart()
 {
```

```
...
this.queueFinal = new ServiceBusPipeFilter(...,Constants.Queue
FinalPath);
this.queueFinal.Start();
...
}
public override void Run()
{
this.queueFinal.OnPipeFilterMessageAsync(
async (msg) =>
{
await Task.Delay(500); // DOING WORK
Trace.TraceInformation(
"Pipeline Message Complete - FilterA:{0} FilterB:{1}",
msg.Properties[Constants.FilterAMessageKey],
msg.Properties[Constants.FilterBMessageKey]);
return null;
});
...
} ...}
```

Библиотека ГГТУ им. П.О.Сухого

Глава 8. Apache Kafka

8.1. Общие сведения об Apache Kafka в Azure HDInsight

Apache Kafka – это распределенная платформа потоковой передачи с открытым исходным кодом, которую можно использовать для создания конвейеров и приложений потоковой передачи данных в режиме реального времени. *Kafka* также предоставляет функцию брокера сообщений, подобную очереди сообщений, с помощью которой можно выполнять публикацию и подписываться на именованные потоки данных.

Ниже приведены характеристики *Kafka* в *HDInsight*:

1. Это управляемая служба, которая упрощает процесс настройки. Она создает конфигурацию, проверенную и поддерживаемую корпорацией Майкрософт.

2. Корпорация Майкрософт предоставляет соглашение об уровне обслуживания (*SLA*), гарантирующее 99,9 % времени непрерывной работы *Kafka*.

3. В качестве резервного хранилища для *Kafka* используются управляемые диски *Azure*. Управляемые диски могут обеспечить до 16 ТБ хранилища для каждого брокера *Kafka*.

4. Служба *Kafka* была разработана для одномерной стойки. *Azure* разделяет стойку на два измерения – домены обновления (*UD*) и домены сбоя (*FD*). Корпорация Майкрософт предоставляет инструменты, которые могут выполнять перераспределение секций и реплик *Kafka* в доменах обновления и доменах сбоя.

5. *HDInsight* позволяет изменить количество рабочих узлов (в которых размещается брокер *Kafka*) после создания кластера. Масштабирование можно выполнить на портале *Azure*, в *Azure PowerShell* и других интерфейсах управления *Azure*. Для *Kafka* выполните повторную балансировку реплик секций после масштабирования. Балансировка секций позволяет *Kafka* пользоваться преимуществами нового количества рабочих узлов.

6. Журналы *Azure Monitor* можно использовать для мониторинга *Kafka* в *HDInsight*. В журналах *Azure Monitor* представлена информация об уровне виртуальной машины (например, метрики диска и сетевой карты, а также метрики *JMX* от *Kafka*).

Библиотека ГГТУ им. П.О. Сухого

8.2. Архитектура Apache Kafka в HDInsight

На рис. 8.1 показана стандартная конфигурация *Kafka*, в которой предусмотрены группы потребителей, секционирование и репликация, чтобы обеспечить параллельное считывание событий и отказоустойчивость.

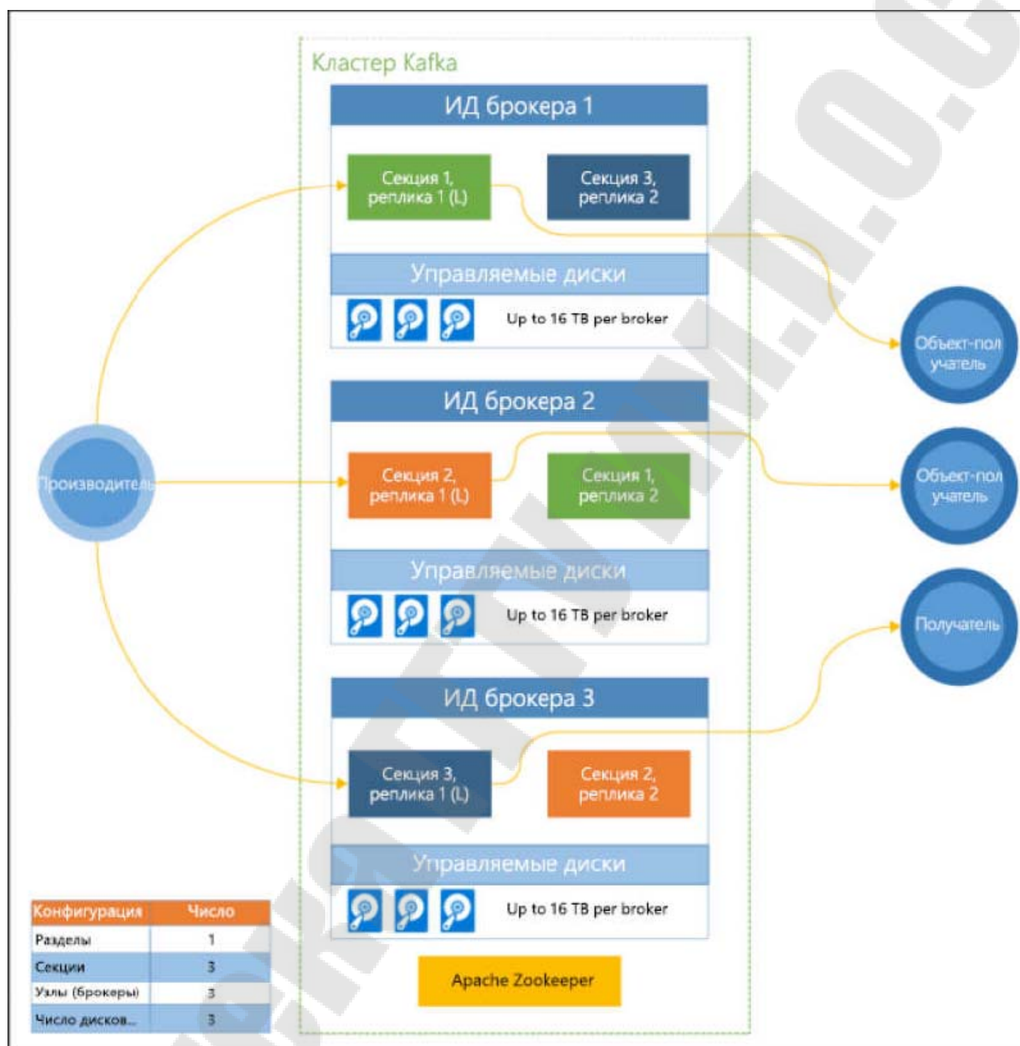


Рис. 8.1. Стандартная конфигурация *Kafka*

Apache ZooKeeper управляет состоянием кластера *Kafka*. Служба *Zookeeper* предназначена для параллельных надежных транзакций с низкой задержкой.

В *Kafka* записи (данные) хранятся в разделах. Записи создаются производителями, а используются потребителями. Производители отправляют записи в брокеры *Kafka*. Каждый рабочий узел в кластере *HDInsight* – это брокер *Kafka*.

Библиотека ГГТУ им.П.О.Сухого

Разделы позволяют распределить записи между брокерами. При считывании записей можно использовать один потребитель на секцию, чтобы обеспечить параллельную обработку данных.

Репликация используется для дублирования секций между узлами, чтобы обеспечить защиту от сбоев узлов (брокеров). Секция, обозначенная на схеме буквой (*L*), является ведущей (*leader*) в определенном разделе. Трафик производителя направляется в ведущую секцию каждого узла в зависимости от состояния, которым управляет *ZooKeeper*.

В табл. 8.1 приведены распространенные задачи и шаблоны, которые могут быть выполнены с помощью *Kafka* в *HDInsight*.

Таблица 8.1

Задачи и шаблоны *Kafka* в *HDInsight*

Использование	Описание
Репликация данных *Apache Kafka*	*Kafka* предоставляет программу *MirrorMaker*, которая производит репликацию данных между кластерами *Kafka*
Модель обмена сообщениями по схеме «публикация – подписка»	*Kafka* предоставляет *API* производителя для публикации записей в разделе *Kafka*. При подписке на раздел используется *API* пользователя
Потоковая обработка	Kafka часто используется с Apache Storm или Spark для потоковой обработки в режиме реального времени. В Kafka 0.10.0.0 (HDInsight 3.5 и 3.6) представлен API для потоковой передачи, позволяющий создавать соответствующие решения без использования Storm или Spark
Горизонтальное масштабирование	В *Kafka* потоки разделяются между узлами в кластере *HDInsight*. Процессы пользователя можно связать с отдельными секциями для обеспечения балансировки нагрузки при использовании записей
Упорядоченная доставка	В каждой секции записи сохраняются в потоке в том порядке, в котором они были получены. Достаточно связать один процесс пользователя с секцией – и записи будут обрабатываться в определенном порядке
Обмен сообщениями	Благодаря поддержке модели обмена сообщениями по схеме «публикация – подписка» *Kafka* часто используется как брокер сообщений

Библиотека ГГТУ им.П.О.Сухого

Окончание табл. 8.1

Использование	Описание
Отслеживание действий	*Kafka* предоставляет возможность ведения журнала записей в определенном порядке, поэтому *Kafka* можно использовать для отслеживания и повторного создания действий. Например, действий пользователя на web-сайте или в приложении
Агрегирование	С помощью потоковой обработки можно объединить информацию из разных потоков для ее централизации в качестве оперативных данных
Преобразование	С помощью потоковой обработки можно объединять и дополнять данные из нескольких входных разделов в один или несколько выходных разделов

Конфигурация влияет на общую производительность *Kafka,* которая зависит от следующих факторов:

– параметры аппаратного оборудования улов кластера (hardware) – память, центральный процессор, жесткие диски;

– пропускная способности сети;

– количество производителей и потребителей;

– конфигурация производителей – совокупность настроек (параметров) producer'ов, которые определяют поведение отправителей сообщений.

8.3. Создание кластера Apache Kafka в Azure HDInsight с помощью портала Azure

Apache Kafka – это распределенная платформа потоковой передачи с открытым кодом. Она часто используется как брокер сообщений, предоставляя такие же функциональные возможности, как и очередь сообщений типа «публикация – подписка».

API Apache Kafka доступен только ресурсам, размещенным в той же виртуальной сети. В этом разделе вы напрямую обращаетесь к кластеру по протоколу *SSH*. Чтобы подключить к *Apache Kafka* другие службы, сети или виртуальные машины, необходимо сначала создать виртуальную сеть, а затем создать в ней эти ресурсы.

Если у вас еще нет подписки *Azure*, создайте бесплатную учетную запись, прежде чем начинать работу.

Чтобы создать кластер *Apache Kafka* в *HDInsight*, сделайте следующее:

Библиотека ГГТУ www.gstu.by

1. Войдите на портал *Azure*.

2. В меню сверху выберите + *Create a resource* (+ Создать ресурс).

3. Выберите элементы *Analytics –> Azure HDInsight*, чтобы перейти на страницу Создание кластера *HDInsight*.

4. На вкладке «Основные сведения» (рис. 8.2) укажите информацию, изложенную в табл. 8.2.

Создание кластера HDInsight

Основные сведения | Хранилище | Безопасность и сеть | Конфигурация и цены | Отзыв и создание

Создайте управляемый кластер HDInsight. Вы можете выбрать Spark, Kafka, Hadoop, Storm и многое другое. Подробнее

Сведения о проекте

Выберите подписку для управления развернутыми ресурсами и затратами. Используйте группы ресурсов, такие как папки, для организации всех ресурсов и управления ими.

* Подписка — Azure

* Группа ресурсов — Выбрать существующую...

Создать

Сведения о кластере

Присвойте кластеру имя, выберите расположение, выберите тип и версию кластера. Подробнее

* Имя кластера — Введите имя нового кластера

* Расположение — Восточная часть США

* Тип кластера — Выберите тип кластера

* Версия

Учетные данные для кластера

Введите новые учетные данные, которые будут использоваться для администрирования кластера и доступа к нему.

* Имя пользователя для входа в кластер — Администратор

* Пароль для входа в кластер — ••••••••••

* Подтвердите пароль для входа в кластер

* Имя пользователя для Secure Shell (SSH) — sshuser

Использовать логин и пароль кластера при подключении через SSH

Отзыв и создание | « Назад | Далее: Хранилище »

Рис. 8.2. Вкладка «Основные сведения»

Библиотека БГТУ им. П.О.Сухого

Таблица 8.2

Основные сведения для создания кластера

Свойство	Описание
Подписка	В раскрывающемся списке выберите подписку *Azure*, которая используется для кластера
Группа ресурсов	Создайте новую группу ресурсов или выберите имеющуюся группу ресурсов. Группа ресурсов – это контейнер компоннтов *Azure*. В данном случае группа ресурсов содержит кластер *HDInsight* и зависимую учетную запись хранения *Azure*
Имя кластера	Введите глобально уникальное имя. Имя может содержать до 59 знаков, включая буквы, цифры и дефисы. Первый и последний знаки в имени не могут быть дефисами
Регион	В раскрывающемся списке выберите регион, в котором создается кластер. Выберите ближайший к себе регион для повышения производительности
Тип кластера	Щелкните «Выберите тип кластера», чтобы открыть список. В списке выберите *Kafka* в качестве типа кластера
Версия	Для типа кластера будет указана версия по умолчанию. Если хотите указать другую версию, выберите ее из раскрывающегося списка
Имя для входа и пароль для кластера	Имя для входа по умолчанию – *admin*. Длина пароля должна составлять не менее 10 символов. Пароль должен содержать по меньшей мере одну цифру, одну прописную и одну строчную буквы, а также один специальный символ (кроме ' " `). Ни в коем случае не вводите распространенные пароли, например *Pass@word*1
Имя пользователя для *Secure Shell* (*SSH*)	Имя пользователя по умолчанию – *sshuser*. Можно указать другое имя пользователя *SSH*
Использование пароля для входа в кластер для *SSH*	Установите этот флажок, чтобы использовать одинаковый пароль для пользователя *SSH* и имени для входа для кластера

В каждом регионе *Azure* (расположении) предоставляются домены сбоя. Домен сбоя – это логическое объединение базового оборудования в центре обработки данных *Azure*. Все домены сбоя используют общий источник питания и сетевой коммутатор. Виртуальные машины и управляемые диски, на которых реализуются узлы в кластере *HDInsight*, распределяются по этим доменам сбоя. Такая ар-

Библиотека ГГТУ им. П.О.Сухого

хитектура ограничивает потенциальное влияние сбоев физического оборудования.

Для обеспечения высокой доступности данных выберите регион (расположение), содержащий три домена сбоя.

Нажмите кнопку «Далее»: Вкладка «Хранилище» >> для перехода к настройкам хранилища.

5. На вкладке «Хранилище» (рис. 8.3) укажите значения приведенные в табл. 8.3.

Создание кластера HDInsight

Основные сведения | Хранилище | Безопасность и сеть | Конфигурация и цены | Теги | Отзыв и создание

Выберите или создайте учетные записи хранения, которые будут использоваться для журналов кластера, входных и выходных данных задания. При необходимости настройте доступ кластера к этим учетным записям.

Основное хранилище

Выберите или создайте учетную запись хранения, которая будет расположением по умолчанию для журналов кластера и других выходных данных.

Тип основного хранилища * — Служба хранилища Azure

Метод выбора * — Выбрать в списке / Использовать ключ доступа

Основная учетная запись хранения * — (Новая) myazurestorageacct

Создать

Контейнер * — mycontainer

Data Lake Storage 1-го поколения

Предоставьте данные для подключения кластера к Azure Data Lake Storage 1-го поколения. После этого кластер сможет обращаться к любым учетным записям Azure Data Lake Storage 1-го поколения, к которым есть доступ у выбранного субъекта-службы.

Доступ к Data Lake Storage 1-го поколения — Настройка параметров доступа

Дополнительная служба хранилища Azure

Свяжите дополнительные учетные записи службы хранилища Azure с кластером.

Добавить службу хранилища Azure

Параметры хранилища метаданных

Чтобы сохранить метаданные Hive и Oozie за пределами этого кластера, выберите базу данных SQL для этого кластера.

База данных SQL для Hive — Выберите базу данных (введите название для фильтрации).

База данных SQL для Oozie — Выберите базу данных (введите название для фильтрации).

База данных SQL для Ambari — Выберите базу данных (введите название для фильтрации).

Просмотреть и создать | « Назад | Далее: Безопасность и сеть »

Рис. 8.3. Вкладка «Хранилище»

Библиотека БГТУ им. П.О. Сухого

Метаданные могут сохранятся в базу данных для различных сервисов, как, например, для Oozie. Oozie – планировщик потоков задач. Изначально спроектирован для объединения отдельных MapReduce работ в единый конвейер и запуска их по расписанию. Дополнительно может выполнять Hive, Java и консольные действия, но в контексте Spark, Impala и др., этот список выглядит довольно бесполезным. Очень сложный и практически не поддается отладке.

Таблица 8.3

Описание вкладки «Хранилище»

Свойство	Описание
Тип первичного хранилища	Используйте значение службы хранилища *Azure* по умолчанию
Метод выбора	Используйте значение «Выбрать в списке по умолчанию»
Основная учетная запись хранения	Используйте раскрывающийся список, чтобы выбрать имеющуюся учетную запись хранения, или создайте новую, щелкнув «Создать». При создании учетной записи имя должно содержать от 3 до 24 символов, включая цифры и строчные буквы
Контейнер	Используйте значение, предоставленное автоматически

Выберите вкладку «Безопасность и сетевые подключения» (рис. 8.4).

6. Значения параметров безопасности оставьте по умолчанию.

Если вы хотите подключить к виртуальной сети кластер, выберите виртуальную сеть из раскрывающегося списка «Виртуальная сеть». Виртуальная сеть может использоваться для репликации разделов между двумя кластерами *HDInsight*. При этом оба кластера находятся в разных виртуальных сетях в центрах обработки данных. Зеркальное отображение работает с помощью средства MirrorMaker (входит в состав *Apache Kafka*), чтобы использовать записи из разделов основного кластера, а затем создать локальную копию на дополнительном кластере. *MirrorMaker* использует одного (или нескольких) потребителя, считывающего данные из основного кластера, и производитель, записывающий данные в локальный (дополнительный) кластер.

Библиотека ГГТУ им. П.О. Сухого

Создание кластера HDInsight

Основные сведения   Хранилище   Безопасность и сеть   Конфигурация и цены   Теги   Отзыв и создание

Настройте параметры безопасности и сети для кластера.

Корпоративный пакет безопасности

Подключите этот кластер к доменным службам Azure Active Directory (AAD-DS) для более полного контроля доступа к кластеру. Подробнее.

☐ Включение Корпоративного пакета безопасности ⓘ (Повышает стоимость на 0,01 долл. США за ядро в час)

TLS

Выберите минимальную поддерживаемую версию TLS для кластера. Подробнее

Минимальная версия TLS ⓘ

Виртуальная сеть

Подключите кластер к виртуальной сети. Подробнее

Виртуальная сеть ⓘ (По умолчанию) Виртуальная сеть, управляемая системой

⚠ Рекомендуется выбрать виртуальную сеть. Создание кластера Kafka без виртуальной сети может сделать кластер непригодным к использованию, так как только клиенты, доступные из виртуальной сети, могут взаимодействовать с отдельными узлами кластера.

Параметры шифрования диска

Настройка шифрования дисков для кластера

☐ Предоставьте свой ключ из Key Vault. Подробнее

Прокси-сервер REST для Kafka

Добавьте прокси-сервер REST для Kafka в кластер.

☐ Включить прокси-сервер REST для Kafka (предварительная версия) Подробнее

Удостоверение

Выберите удостоверение службы, назначаемое пользователем, которое будет представлять ваш кластер для Корпоративного пакета безопасности или шифрования дисков. Подробнее

Управляемое удостоверение, назн... ⓘ — Я не хочу предоставлять управляемое удостоверение службы.

Просмотреть и создать   « Назад   Далее: Конфигурация и цены »

Рис. 8.4. Вкладка «Безопасность и сетевые подключения»

Перейдите на вкладку «Конфигурация и цены» (рис. 8.5).

7. Для обеспечения доступности кластера *Apache Kafka* в *HDInsight* число узлов для рабочего узла должно быть не меньше 3. Значение по умолчанию – 4.

Библиотека ГГТУ им. П.О.Сухого

Число стандартных дисков для каждой записи рабочего узла определяет степень масштабируемости *Apache Kafka* в *HDInsight*. Кластер *Apache Kafka* в *HDInsight* использует локальный диск виртуальных машин в кластере для хранения данных. *Apache Kafka* обрабатывает большое количество операций ввода-вывода, поэтому используются Управляемые диски *Azure*, чтобы обеспечить высокую пропускную способность и предоставить дополнительное хранилище для каждого узла. Управляемый диск может быть двух типов: Стандартный (*HDD*) или Премиум (*SSD*). Тип диска зависит от размера виртуальной машины, используемой рабочими узлами (брокерами *Apache Kafka*). Диски категории «Премиум» автоматически используются для виртуальных машин серий *DS* и *GS*. Для всех остальных виртуальных машин используются стандартные управляемые диски.

Рис. 8.5. Вкладка «Конфигурация и цены»

Библиотека БГТУ им. П.О. Сухого

Перейдите на вкладку *Review + create* (Просмотр и создание), показанную на рис. 8.6.

8. Проверьте конфигурацию кластера. Измените неправильные параметры. Наконец, выберите «Создать», чтобы создать кластер.

Создание кластера HDInsight

Перейти к классическому интерфейсу создания

Проверка прошла успешно.

Основные сведения | Хранилище | Безопасность и сеть | Конфигурация и цены | Просмотр и создание

Kafka 1.1.0 (HDI 3.6)

<цена> долл. США — расчетная стоимость в час
Эта оценка стоимости не учитывает скидки по подписке и (или) затраты на хранение, сеть и передачу данных.

Основные сведения

Подписка	Azure
Группа ресурсов	(новая) myResourceGroup
Расположение	Восточная часть США
Имя кластера	(новая) MyKafka2019
Тип кластера	Kafka 1.1.0 (HDI 3.6)
Имя пользователя для входа в кластер	admin
Имя пользователя для Secure Shell (SSH)	sshuser
Использовать логин и пароль кластера при подключении через SSH	Включено

Хранилище

Тип первичного хранилища	Azure Storage
Основная учетная запись хранения	(новое) mykafka2019hdistorage
Контейнер	mykafka2019-2019-09-27t23-16-22-445z
Дополнительная служба хранилища Azure	Нет
Доступ к Data Lake Storage 1-го поколения	Отключено

Конфигурация кластера

Головной	Узлы: 2, D3 v2 (4 Cores, 14 GB RAM)
Zookeeper	Узлы: 3, A4 v2 (4 Cores, 8 GB RAM)
Диски	Узлы: 8, S30
Рабочая роль	Узлы: 4, D3 v2 (4 Cores, 14 GB RAM)

Создание | « Назад | Далее | Скачать шаблон для автоматизации

Рис. 8.6. Вкладка «Просмотр и создание»

Операция создания кластера может занять до 20 минут.

Подключение к кластеру

1. С помощью команды ***ssh command*** подключитесь к кластеру. Измените приведенную ниже команду, заменив ***CLUSTERNAME*** именем своего кластера, а затем введите команду:

Библиотека ГГТУ им. П.О. Сухого

ssh sshuser@CLUSTERNAME-ssh.azurehdinsight.net

2. При появлении запроса введите пароль пользователя SSH.

После подключения отобразятся сведения, аналогичные приведенному ниже тексту:

Authorized uses only. All activity may be monitored and reported.

Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.13.0-1011-azure x86_64)

Documentation: https://help.ubuntu.com

Management: https://landscape.canonical.com

Support: https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest: https://www.ubuntu.com/business/services/cloud

83 packages can be updated.

37 updates are security updates.

Welcome to Apache Kafka on HDInsight.

Last login: Thu Mar 29 13:25:27 2018 from 108.252.109.241

8.4. Получение сведений об узлах Apache Zookeeper и брокера

Для работы с Kafka необходимы сведения об узле Apache Zookeeper и узле брокера. Эти узлы используются API Apache Kafka и многими другими служебными программами, поставляемыми с платформой Kafka.

В этом разделе для получения сведений об узле используется REST API Apache Ambari в кластере:

1. Установите *jq* – обработчик командной строки JSON. Эта служебная программа используется для анализа документов JSON. С ее помощью также удобно анализировать сведения об узлах.

2. Извлеките имя кластера с правильным регистром. Фактический регистр имени кластера может отличаться от ожидаемого, в зависимости от способа создания кластера. Эта команда получит фактический регистр для хранения в переменной.

3. Введите следующую команду:

export clusterName=$(curl -u admin:$password -sS -G http:// headnodehost:8080/api/v1/clusters | jq -r '.items[].Clusters.cluster_name')

4. Чтобы задать переменную среды со сведениями об узле Zookeeper, выполните следующую команду. Эта команда извлекает

Библиотека ГГТУ им. П.О.Сухого

все узлы Zookeeper, а затем возвращает только первые две записи. Причина этого состоит в том, что требуется обеспечить избыточность на случай, если узел станет недоступным:

*export HOSTS=$(curl -sS -u admin:$password -G https://$clusterName.azurehdinsight.net/api/v1/clusters/$clusterName/services/ZOOKEEPER/component s/ZOOKEEPER_SERVER | jq -r '["\(.host_-components[].
HostRoles.host_name):2181"] | join(",")' | cut -d',' -f1,2);*

5. Чтобы убедиться, что переменная среды задана верно, выполните следующую команду:

echo $HOSTS

6. Эта команда возвращает сведения аналогичные следующим:

*zk0kafka.eahjefxxp1netdbyklgqj5y1ud.ex.internal.cloudapp.net:2181
zk2kafka.eahjefxxp1netdbyklgqj5y1ud.ex.internal.cloudapp.net:2181*

7. Чтобы задать переменную среды со сведениями об узле брокера Apache Kafka, выполните следующую команду:

export KAFKABROKERS=$(curl -sS -u admin:$password -G

https://$clusterName.azurehdinsight.net/api/v1/clusters/$clusterName/services/KAFKA/components/KA FKA_BROKER | jq -r '["\(.host_components[].HostRoles.host_name):9092"] | join(",")' | cut -d',' f1,2);

8. Чтобы убедиться, что переменная среды задана верно, выполните следующую команду:

echo $KAFKABROKERS

9. Эта команда возвращает сведения, аналогичные следующим:

*wn1kafka.eahjefxxp1netdbyklgqj5y1ud.cx.internal.cloudapp.net:9092,
wn0kafka.eahjefxxp1netdbyklgqj5y1ud.cx.internal.cloudapp.net:9092*

8.5. Создание кластера Apache Kafka в Azure HDInsight с помощью шаблонов Resource Manager

Рассмотрим, как с помощью шаблона *Azure Resource Manager* создать кластер *Apache Kafka* в *Azure HDInsight*.

Шаблон *Resource Manager* является файлом нотации объектов *JavaScript* (*JSON*), определяющих инфраструктуру и конфигурацию вашего проекта. Шаблон использует декларативный синтаксис, который позволяет указать объект, который вы собираетесь развернуть. При этом для развертывания объекта не нужно писать последовательность команд.

Библиотека ГГТУ им. П.О. Сухого

API Kafka доступен только ресурсам, размещенным в той же виртуальной сети. В этом разделе вы напрямую обращаетесь к кластеру по протоколу *SSH*. Чтобы подключить к *Kafka* другие службы, сети или виртуальные машины, необходимо сначала создать виртуальную сеть, а затем создать в ней эти ресурсы.

Шаблон, используемый в этом разделе, взят из шаблонов быстрого запуска *Azure*:

```
{
"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
"clusterName": {
"type": "string",
"metadata": {
"description": "The name of the Kafka cluster to create. This must be a unique name."
}
},
"clusterLoginUserName": {
"type": "string",
"metadata": {
"description": "These credentials can be used to submit jobs to the cluster and to log into cluster dashboards."
}
},
"clusterLoginPassword": {
"type": "securestring",
"metadata": {
"description": "The password must be at least 10 characters in length and must contain at least one digit, one non-alphanumeric character, and one upper or lower case letter."
}
},
"sshUserName": {
"type": "string",
"metadata": {
```

Библиотека ГГТУ им. П.О. Сухого

```
"description": "These credentials can be used to remotely access the cluster."
}
},
"sshPassword": {
"type": "securestring",
"metadata": {
"description": "The password must be at least 10 characters in length and must contain at least one digit, one non-alphanumeric character, and one upper or lower case letter."
}
},
"location": {
"type": "string",
"defaultValue": "[resourceGroup().location]",
"metadata": {
"description": "Location for all resources." }
}
},
"variables": {
"defaultStorageAccount": {
"name": "[uniqueString(resourceGroup().id)]",
"type": "Standard_LRS"
}
},
"resources": [
{
"type": "Microsoft.Storage/storageAccounts",
"name": "[variables('defaultStorageAccount').name]",
"location": "[parameters('location')]",
"apiVersion": "2017-10-01",
"sku": {
"name": "[variables('defaultStorageAccount').type]"
},
"kind": "Storage",
"properties": {}
},
{
"name": "[parameters('clusterName')]",
```

Библиотека ГГТУ им. П.О. Сухого

```
"type": "Microsoft.HDInsight/clusters",
"location": "[parameters('location')]",
"apiVersion": "2015-03-01-preview",
"dependsOn": [
"[con-
cat('Microsoft.Storage/storageAccounts/',variables('defaultStorageAccount
').name)]"
],
"tags": {},
"properties": {
"clusterVersion": "3.6",
"osType": "Linux",
"clusterDefinition": {
"kind": "kafka",
"configurations": {
"gateway": {
"restAuthCredential.isEnabled": true,
"restAuthCredential.username":                    "[parame-
ters('clusterLoginUserName')]",
"restAuthCredential.password":                    "[parame-
ters('clusterLoginPassword')]"
}
}
},
"storageProfile": {
"storageaccounts": [
{
"name":                                                  "[re-
place(replace(concat(reference(concat('Microsoft.Storage/storageAccount
s/',
variables('defaultStorageAccount').name),                 '2017-10-
01').primaryEndpoints.blob),'https:',"),'/',")]",
"isDefault": true,
"container": "[parameters('clusterName')]",
"key": "[listKeys(resourceId('Microsoft.Storage/storageAccounts',
variables('defaultStorageAccount').name),                 '2017-10-
01').keys[0].value]"
}
]
```


Библиотека ГГТУ им. П.О.Сухого

```
},
"computeProfile": {
"roles": [
{
"name": "headnode",
"targetInstanceCount": 2,
"hardwareProfile": {
"vmSize": "Standard_D3_v2"
},
"osProfile": {
"linuxOperatingSystemProfile": {
"username": "[parameters('sshUserName')]",
"password": "[parameters('sshPassword')]" }
}
},
{
"name": "workernode",
"targetInstanceCount": 4,
"hardwareProfile": {
"vmSize": "Standard_D3_v2"
},
"dataDisksGroups": [
{
"disksPerNode": 2
}
],
"osProfile": {
"linuxOperatingSystemProfile": {
"username": "[parameters('sshUserName')]",
"password": "[parameters('sshPassword')]" }
}
},
{
"name": "zookeepernode",
"targetInstanceCount": 3,
"hardwareProfile": {
"vmSize": "Standard_A3"
},
"osProfile": {
```

Библиотека ГГТУ им. П.О.Сухого

```
"linuxOperatingSystemProfile": {
"username": "[parameters('sshUserName')]",
"password": "[parameters('sshPassword')]" }
}
}
]
}
}
}
],
"outputs": {
"cluster": {
"type": "object",
"value": "[reference(resourceId('Microsoft.HDInsight/clusters',parameters('clusterName')))]"
}
}
}
```

В шаблоне определено два ресурса *Azure*:

– с помощью *Microsoft.Storage/storageAccounts* создается учетная запись хранения *Azure*;

– с помощью *Microsoft.HDInsight/cluster* создается кластер *HDInsight*.

Развертывание шаблона:

– нажмите кнопку «Развертывание» в *Azure* ниже, чтобы войти в *Azure* и открыть шаблон *Resource Manager*;

– введите или выберите следующие значения (рис. 8.6) из табл. 8.3;

– ознакомьтесь с УСЛОВИЯМИ ИСПОЛЬЗОВАНИЯ. Затем установите флажок «Я принимаю указанные выше условия» и нажмите кнопку «Приобрести». Вы получите уведомление о выполнении развертывания. Процесс создания кластера занимает около 20 минут.

Подключение к кластеру

Чтобы подключиться к первичному головному узлу кластера Kafka, выполните приведенную ниже команду. Замените sshuser именем пользователя SSH. Замените mykafka именем кластера Kafka:

```
ssh sshuser@mykafka-ssh.azurehdinsight.net
```


Библиотека ГГТУ им.П.О.Сухого

Таблица 8.4

Описание шаблон Resource Manager

Свойство	Описание
Подписка	В раскрывающемся списке выберите подписку *Azure*, которая используется для кластера
Группа ресурсов	В раскрывающемся списке выберите существующую группу ресурсов, а затем «Создать новую»
Расположение	В качестве значения будет автоматически указано расположение, используемое для группы ресурсов
Имя кластера,	Введите глобально уникальное имя. Для этого шаблона вы можете использовать только строчные буквы и цифры
Имя пользователя для входа в кластер	Укажите имя пользователя, по умолчанию – *admin*
Пароль для входа в кластер	Введите пароль. Длина пароля должна составлять не менее 10 символов. Пароль должен содержать по меньшей мере одну цифру, одну прописную и одну строчную буквы, а также один специальный символ (кроме ' " `)
Имя пользователя *SSH*	Укажите имя пользователя, по умолчанию – *sshuser*
Пароль *SSH*	Укажите пароль

После создания кластера вы получите уведомление «Развертывание прошло успешно» со ссылкой «Перейти к ресурсу». На странице группы ресурсов будет указан новый кластер *HDInsight* и хранилище по умолчанию, связанное с кластером. У каждого кластера есть зависимость учетной записи службы хранилища *Azure* или учетной записи *Azure Data Lake Storage*. Она называется учетной записью хранения по умолчанию. Кластер *HDInsight* должен находиться в том же регионе *Azure*, что и его учетная запись хранения, используемая по умолчанию. Удаление кластеров не приведет к удалению учетной записи хранения.

Библиотека ГГТУ им.П.О.Сухого

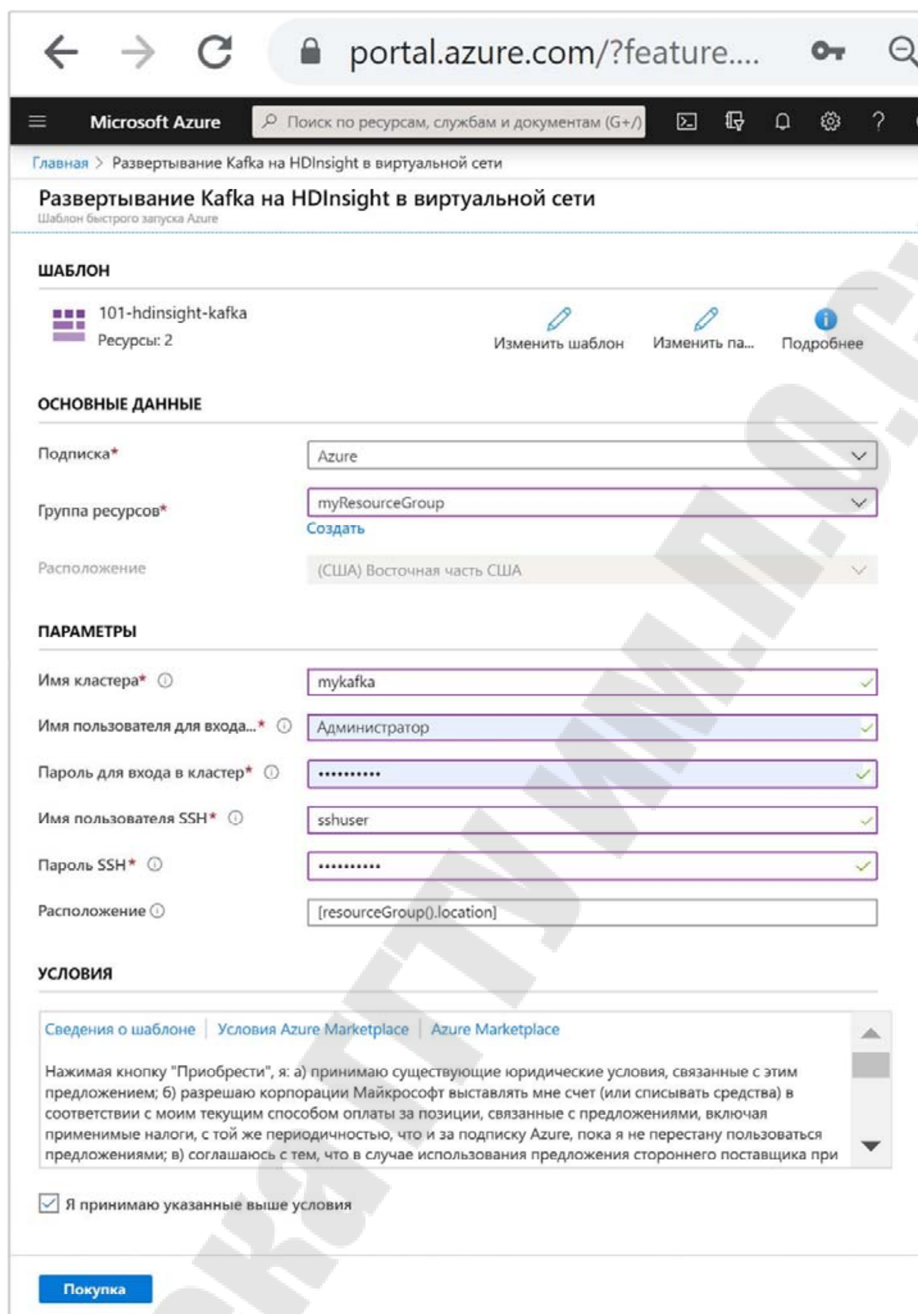


Рис. 8.7. Развертывание *Kafka*

Очистка ресурсов

После завершения работы с этим кратким руководством кластер можно удалить. В случае с *HDInsight* ваши данные хранятся в службе хранилища *Azure*, что позволяет безопасно удалить неиспользуемый кластер. Плата за кластеры *HDInsight* взимается, даже когда они не используются. Так как затраты на кластер во много раз превышают затраты на хранилище, экономически целесообразно удалять неиспользуемые кластеры.

Библиотека ГГТУ им.П.О.Сухого

На портале *Azure* перейдите в свой кластер и выберите «Удалить».

Кроме того, можно выбрать имя группы ресурсов, чтобы открыть страницу группы ресурсов, а затем щелкнуть «Удалить группу ресурсов». Вместе с группой ресурсов вы также удалите кластер *HDInsight* и учетную запись хранения по умолчанию.

Давайте создадим приложение для публикации и использования сообщений с помощью клиента *Java*. Клиент производителя *Kafka* состоит из следующих *API*.

8.6. KafkaProducer API

Давайте разберемся с наиболее важным набором *API* производителей *Kafka* в этом разделе. Центральная часть *API KafkaProducer* – класс *KafkaProducer*. Класс *KafkaProducer* предоставляет возможность подключить брокер *Kafka* в своем конструкторе с помощью следующих методов.

Класс *KafkaProducer* предоставляет метод *send* для асинхронной отправки сообщений в тему. Подпись *send* () выглядит следующим образом:

producer.*send*(*new ProducerRecord<byte[],byte[]>*(*topic*,
partition, *key*1, *value*1) , *callback*);

ProducerRecord – производитель управляет буфером записей, ожидающих отправки.

Обратный вызов – предоставленный пользователем обратный вызов для выполнения, когда запись была подтверждена сервером (ноль означает отсутствие обратного вызова).

Класс *KafkaProducer* предоставляет метод сброса, обеспечивающий фактическое завершение всех ранее отправленных сообщений. Синтаксис метода очистки следующий:

public void flush();

Класс *KafkaProducer* предоставляет метод *partitionFor*, который помогает в получении метаданных раздела для данной темы. Это может быть использовано для пользовательского разбиения:

public Map metrics();

Возвращает карту внутренних метрик, поддерживаемых производителем.

public void close () – класс *KafkaProducer* предоставляет блоки методов *close*, пока все ранее отправленные запросы не будут выполнены.

public void close () – класс *KafkaProducer* предоставляет блоки методов *close*, пока все ранее отправленные запросы не будут выполнены.

Библиотека ГГТУ им. П.О.Сухого

***API* продюсера**

Центральной частью *API* производителя является класс *Producer*. Класс *Producer* предоставляет возможность подключить брокер *Kafka* в своем конструкторе следующими методами.

Класс продюсера

Класс продюсера предоставляет метод *send* для отправки сообщений в одну или несколько тем с использованием следующих подписей:

public void send(*KeyedMessaget<k*,*v> message*) - *sends the data to a single topic*, *par-titioned by key using either sync or async producer*.

public void send(*List<KeyedMessage<k*,*v>>messages*) - *sends data to multiple topics*.

Properties prop = *new Properties*();

prop.*put*(*producer*.*type*,"*async*")

ProducerConfig config = *new ProducerConfig*(*prop*);

Существует два типа производителей – *Sync* и *Async*.

Та же конфигурация *API* применима и к производителю *Sync*. Разница между ними заключается в том, что производитель синхронизации отправляет сообщения напрямую, но отправляет сообщения в фоновом режиме. Асинхронный производитель предпочтителен, когда вы хотите более высокую пропускную способность. В предыдущих выпусках, таких как 0.8, у асинхронного производителя нет обратного вызова для *send*() для регистрации обработчиков ошибок. Это доступно только в текущей версии 0.9.

public void close ()

Класс *Producer* предоставляет метод *close* для закрытия соединений пула производителей со всеми брокерами *Kafka*.

ProducerRecord API

ProducerRecord – это пара «ключ – значение», отправляемая в кластер *Kafka*. Конструктор класса *ProducerRecord* для создания записи с парами разделов, ключей и значений с использованием следующей подписи:

public ProducerRecord (*string topic*, *int partition*, *k key*, *v value*);

Тема – определенное пользователем название темы, которое будет добавлено в запись.

Раздел – количество разделов.

Ключ – ключ, который будет включен в запись.

Значение – запись содержимого.

Библиотека ГГТУ им. П.О. Сухого

Тема – определенное пользователем название темы, которое будет добавлено в запись.

Раздел – количество разделов.

Ключ – ключ, который будет включен в запись.

public ProducerRecord (*string topic*, *k key*, *v value*);

Конструктор класса *ProducerRecord* используется для создания записи с ключом, парами значений и без разделения:

Тема – Создать тему для назначения записи.

Ключ – ключ для записи.

Значение – запись содержимого.

Тема – Создать тему для назначения записи.

Ключ – ключ для записи.

Значение – запись содержимого.

public ProducerRecord (*string topic*, *v value*)

Класс *ProducerRecord* создает запись без раздела и ключа.

Тема – создать тему.

Значение – запись содержимого.

Тема – создать тему.

Значение – запись содержимого.

Приложение *SimpleProducer*

Перед созданием приложения сначала запустите *ZooKeeper* и *Kafka broker*, затем создайте свою собственную тему в *Kafka broker*, используя команду *create topic*. После этого создайте класс *Java* с именем *SimpleProducer.java* и введите следующую кодировку:

SimpleProducer.java:

```
//import util.properties packages
import java.util.Properties;
//import simple producer packages
import org.apache.kafka.clients.producer.Producer;
//import KafkaProducer packages
import org.apache.kafka.clients.producer.KafkaProducer;
//import ProducerRecord packages
import org.apache.kafka.clients.producer.ProducerRecord;
//Create java class named "SimpleProducer"
public class SimpleProducer {
 public static void main(String[] args) throws Exception{
 // Check arguments length value
 if(args.length == 0){
```


Библиотека ГГТУ им. П.О.Сухого

```
System.out.println("Enter topic name");
return;
}
//Assign topicName to string variable
String topicName = args[0].toString();
// create instance for properties to access producer configs
Properties props = new Properties();
//Assign localhost id
props.put("bootstrap.servers", "localhost:9092");
//Set acknowledgements for producer requests.
props.put("acks", "all");
//If the request fails, the producer can automatically retry,
props.put("retries", 0);
//Specify buffer size in config
props.put("batch.size", 16384);
//Reduce the no of requests less than 0
props.put("linger.ms", 1);

//The buffer.memory controls the total amount of memory available
to the producer for buffering.
props.put("buffer.memory", 33554432);
props.put("key.serializer",
"org.apache.kafka.common.serializa-tion.StringSerializer");
props.put("value.serializer",
"org.apache.kafka.common.serializa-tion.StringSerializer");
Producer<String, String> producer = new KafkaProducer
<String, String>(props);
for(int i = 0; i < 10; i++)
producer.send(new ProducerRecord<String, String>(topicName,
Integer.toString(i), Integer.toString(i)));
System.out.println("Message sent successfully");
producer.close();
}
}
```

Компиляция – приложение может быть скомпилировано с помощью следующей команды:

javac -cp "/path/to/kafka/kafka_2.11-0.9.0.0/lib/" *.java*

Выполнение – приложение может быть выполнено с помощью следующей команды:

Библиотека ГГТУ им. П.О.Сухого

*java -cp "/path/to/kafka/kafka*_2.11-0.9.0.0/*lib*/*":. *SimpleProducer* *<topic-name>*

Вывод команд:

Message sent successfully

To check the above output open new terminal and type Consumer CLI command to receive messages.

>> bin/kafka-console-consumer.sh --zookeeper localhost:2181 *–topic* *<topic-name> –from-beginning*

1
2
3
4
5
6
7
8
9
10

Простой потребительский пример

На данный момент мы создали производителя для отправки сообщений в кластер *Kafka*. Теперь давайте создадим потребителя для потребления сообщений из кластера *Kafka*. *API KafkaConsumer* используется для приема сообщений из кластера *Kafka*. Конструктор класса *KafkaConsumer* определен ниже:

public KafkaConsumer(*java.util.Map<java.lang.String,java.lang.Object>configs*)

configs – вернуть карту пользовательских конфигов.

ConsumerRecord API

API ConsumerRecord используется для получения записей из кластера *Kafka*. Этот *API* состоит из имени темы, номера раздела, из которого принимается запись, и смещения, которое указывает на запись в разделе *Kafka*. Класс *ConsumerRecord* используется для создания записи потребителя с определенным именем темы, количеством разделов и парами «ключ – значение». Имеет следующую подпись:

public ConsumerRecord(*string topic*,*int partition*, *long offset*,*K key*, *V value*).

Тема – Название темы для записи потребителя, полученной из кластера *Kafka*.

Раздел – Раздел по теме.

Библиотека ГГТУ им.П.О.Сухого

Ключ – Ключ записи, если ключа не существует, возвращается ноль.

Значение – запись содержимого.

Тема – Название темы для записи потребителя, полученной из кластера *Kafka*.

Раздел – Раздел по теме.

Ключ – ключ записи, если ключа не существует, возвращается ноль.

Значение – запись содержимого.

ConsumerRecords API

ConsumerRecords API действует как контейнер для *ConsumerRecord*. Этот *API* используется для хранения списка *ConsumerRecord* для каждого раздела для определенной темы. Его конструктор определен ниже:

public ConsumerRecords(*java.util.Map<TopicPartition,java.util.List <Consumer-Record>K,V>>> records*)

TopicPartition – возвращает карту разделов для определенной темы.

Записи – возврат списка *ConsumerRecord*.

TopicPartition – возвращает карту разделов для определенной темы.

Записи – возврат списка *ConsumerRecord*.

Приложение *SimpleConsumer*

Шаги создания приложения производителя остаются неизменными. Сначала запустите брокера *ZooKeeper* и *Kafka*. Затем создайте приложение *SimpleConsumer* с помощью класса *Java* с именем *SimpleConsumer.java* и введите следующий код:

```
import java.util.Properties;
import java.util.Arrays;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;

public class SimpleConsumer {
 public static void main(String[] args) throws Exception {
 if(args.length == 0){
 System.out.println("Enter topic name");
 return;
```


Библиотека БГТУ им. П.О. Сухого

```
}
//Kafka consumer configuration settings
String topicName = args[0].toString();
Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("session.timeout.ms", "30000");
props.put("key.deserializer",
"org.apache.kafka.common.serializa-tion.StringDeserializer");
props.put("value.deserializer",
"org.apache.kafka.common.serializa-tion.StringDeserializer");
KafkaConsumer<String, String> consumer = new KafkaConsumer
<String, String>(props);
//Kafka Consumer subscribes list of topics here.
consumer.subscribe(Arrays.asList(topicName))
//print the topic name
System.out.println("Subscribed to topic " + topicName);
int i = 0;
while (true) {
ConsumerRecords<String, String> records = con-sumer.poll(100);
for (ConsumerRecord<String, String> record : records
// print the offset,key and value for the consumer records.
System.out.printf("offset = %d, key = %s, value = %s\n",
record.offset(), record.key(), record.value());
}
}
}
```

Компиляция – приложение может быть скомпилировано с помощью следующей команды:

javac -cp “*/path/to/kafka/kafka*_2.11-0.9.0.0/*lib*/*” **.java*

Выполнение – приложение может быть выполнено с помощью следующей команды:

java -cp “*/path/to/kafka/kafka*_2.11-0.9.0.0/*lib*/*”:. *SimpleConsumer* *<topic-name>*

Вход – откройте *CLI* производителя и отправьте несколько сообщений в тему. Вы можете указать *smple input* как «*Hello Consumer*».

Библиотека ГГТУ им. П.О.Сухого

Глава 9. Apache Storm

9.1. Общие сведения об Apache Kafka в Azure HDInsight

Apache Storm – это распределенная отказоустойчивая вычислительная система с открытым исходным кодом. Ее можно использовать для обработки потоков данных в реальном времени с помощью *Apache Hadoop*.

Решения *Storm* могут также обеспечить гарантированную обработку данных и возможность воспроизвести те данные, которые не прошли удачную обработку в первый раз.

Использование *Storm* в *HDInsight* обеспечивает следующие преимущества:

1. Соглашение об уровне обслуживания (*SL A*) 99 % в отношении бесперебойной работы *Storm*. Для кластеров *Storm* в *HDInsight* действует полная и постоянная поддержка. Кроме того, для кластеров *Storm* в *HDInsight* заявлена гарантированная доступность в течение 99,9 % времени. Это означает, что корпорация Майкрософт гарантирует возможность подключения к кластеру *Storm* извне в течение как минимум 99,9 % времени.

2. Простая настройка с помощью скриптов во время или после создания кластера *Storm*. Создание решений на нескольких языках. Вы можете написать компоненты *Storm* на удобном для вас языке, например *Java*, *C#* и *Python*.

3. Интеграция *Visual Studio* с *HDInsight* для разработки, администрирования и отслеживания топологий *C#*.

4. Поддержка *Java*-интерфейса *Trident*. Вы можете создавать топологии *Storm*, поддерживающие обработку сообщений в рамках подхода «только один раз», сохраняемость «транзакционных» хранилищ данных, а также набор распространенных операций *Stream Analytics*.

5. Динамическое масштабирование. Вы можете добавить или удалить рабочие узлы, не влияя на выполняющиеся топологии *Storm*. Чтобы использовать новые узлы, добавленные с помощью операций масштабирования, деактивируйте и повторно активируйте выполняющиеся топологии.

6. Создание конвейеров потоковой передачи с помощью нескольких служб *Azure*. *Storm* в *HDInsight* интегрируется с другими службами *Azure*, например, Центрами событий, Базой данных *SQL*, службой хранилища *Azure* и *Azure Data Lake Storage*.

Библиотека ГГТУ им. П. О. Сухого

Использование *Apache Storm*

Storm работает с топологиями, а не заданиями *Apache Hadoop MapReduce*, которые вы могли использовать ранее. Топологии *Storm* состоят из нескольких компонентов, которые расположены в виде направленного ациклического графа (*DAG*). Каждый компонент использует один или несколько потоков данных и может генерировать дополнительные потоки.

Надежность

Система *Apache Storm* гарантирует, что каждое входящее сообщение будет полностью обработано даже в том случае, если анализ данных распределен между сотнями узлов.

Узел *Nimbus* предоставляет функции, аналогичные *Apache Hadoop JobTracker*. *Nimbus* назначает задачи другим узлам в кластере с помощью *Apache ZooKeeper*. Узлы *Zookeeper* обеспечивают координацию кластера и способствуют взаимодействию между узлом *Nimbus* и процессом контролера на рабочих узлах. Если один узел обработки выходит из строя, узел *Nimbus* узнает об этом и назначает задачу и связанные данные другому узлу.

По умолчанию для кластеров *Apache Storm* используется только один узел *Nimbus*, но кластер *Storm* в *HDInsight* имеет два. При сбое основного узла кластер *Storm* переключится на дополнительный на время восстановления основного узла.

9.2. Создание топологии

Топология Apache Storm на платформе Java состоит из компонентов, которые необходимо создать или указать как зависимость:

- «Воронки» – чтение данных из внешних источников и отправка потоков данных в топологию.
- «Винты» – обработка потоков, созданных с помощью порождений или других «винтов», и эмиссия одного или нескольких потоков.
- Топология – определяет взаимное расположение воронок и сит и предоставляет точку входа для топологии.

Создание «воронки»

Чтобы снизить требования к настройке внешних источников данных, следующая воронка просто выдает случайные предложения. Это измененная версия *spout*, которая поставляется с примерами для работы с наборами начальных версий. Хотя в этой топологии исполь-

Библиотека ГГТУ им. П.О.Сухого

зуется один *spout*, другие могут иметь несколько потоков данных из разных источников в топологии.

Введите следующую команду, чтобы создать и открыть новый файл:

notepad src\main\java\com\microsoft\example\RandomSentenceSpout.java

Затем скопируйте и вставьте приведенный ниже код *Java* в новый файл. Затем закройте файл.

RandomSentenceSpout.java

```
package com.microsoft.example;
import org.apache.storm.spout.SpoutOutputCollector;
import org.apache.storm.task.TopologyContext;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseRichSpout;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Values;
import org.apache.storm.utils.Utils;
import java.util.Map; import java.util.Random;
//This spout randomly emits sentences public class RandomSentenceSpout extends BaseRichSpout {
//Collector used to emit output
SpoutOutputCollector _collector;
//Used to generate a random number
Random _rand;
//Open is called when an instance of the class is created
@Override public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
//Set the instance collector to the one passed in
_collector = collector;
//For randomness
_rand = new Random();
}
//Emit data to the stream
@Override public void nextTuple() {
//Sleep for a bit
Utils.sleep(100);
//The sentences that are randomly emitted
String[] sentences = new String[]{ "the cow jumped over the moon",
```


Библиотека ГГТУ им. П.О. Сухого

```
"an apple a day keeps the doctor away",
        "four score and seven years ago", "snow white and the seven dwarfs", "i am at two with nature" };
        //Randomly pick a sentence
        String sentence = sentences[_rand.nextInt(sentences.length)];
        //Emit the sentence
        _collector.emit(new Values(sentence));
        }
        //Ack is not implemented since this is a basic example
        @Override public void ack(Object id) { }
        //Fail is not implemented since this is a basic example
        @Override public void fail(Object id) { }
        //Declare the output fields. In this case, an sentence
        @Override
        public void declareOutputFields(OutputFieldsDeclarer declarer) { declarer.declare(new Fields("sentence")); }
        }
```

Создание «сит»

Сита выполняют обработку данных. Элементы *bolt* могут выполнять любые операции, например, вычисление, сохранение, а также взаимодействие с внешними компонентами. Эта топология включает два сита:

1) *SplitSentence* – разделяет предложения, отправленные *RandomSentenceSpout*, на отдельные слова;

2) *WordCount* – подсчитывает частоту употребления каждого слова.

Введите следующую команду, чтобы создать и открыть новый файл:

notepad src\main\java\com\microsoft\example\SplitSentence.java

Затем скопируйте и вставьте приведенный ниже код *Java* в новый файл. Затем закройте файл.

```
SplitSentence.java
package com.microsoft.example;
import java.text.BreakIterator;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
```


Библиотека ГГТУ им. П.О. Сухого

```
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
//There are a variety of bolt types. In this case, use BaseBasicBolt
public class SplitSentence extends BaseBasicBolt {
//Execute is called to process tuples
@Override public void execute(Tuple tuple, BasicOutputCollector collector) {
//Get the sentence content from the tuple
String sentence = tuple.getString(0);
//An iterator to get each word
BreakIterator boundary=BreakIterator.getWordInstance();
//Give the iterator the sentence boundary.setText(sentence);
//Find the beginning first word int start=boundary.first();
//Iterate over each word and emit it to the output stream
for (int end=boundary.next();
end != BreakIterator.DONE;
start=end, end=boundary.next())
{
//get the word
String word=sentence.substring(start,end);
//If a word is whitespace characters, replace it with empty word=word.replaceAll("\\s+","");
//if it's an actual word, emit it
if (!word.equals("")) { collector.emit(new Values(word)); }
}
}
//Declare that emitted tuples contain a word field
@Override public void declareOutputFields (OutputFieldsDeclarer declarer)
{ declarer.declare(new Fields("word")); }
}
```

Введите следующую команду, чтобы создать и открыть новый файл:

notepad src\main\java\com\microsoft\example\WordCount.java

Затем скопируйте и вставьте приведенный ниже код *Java* в новый файл. Затем закройте файл.

```
WordCount.java
package com.microsoft.example;
```


Библиотека ГГТУ им. П.О. Сухого

```
import java.util.HashMap;
import java.util.Map;
import java.util.Iterator;
import org.apache.storm.Constants;
import org.apache.storm.topology.BasicOutputCollector;
import org.apache.storm.topology.OutputFieldsDeclarer;
import org.apache.storm.topology.base.BaseBasicBolt;
import org.apache.storm.tuple.Fields;
import org.apache.storm.tuple.Tuple;
import org.apache.storm.tuple.Values;
import org.apache.storm.Config;
// For logging import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.LogManager;
//There are a variety of bolt types. In this case, use BaseBasicBolt
public class WordCount extends BaseBasicBolt {
//Create logger for this class
private static final Logger logger = LogManager.getLogger-
(WordCount.class);
//For holding words and counts
Map<String, Integer> counts = new HashMap<String, Integer>();
//How often to emit a count of words private Integer emitFrequency;
// Default constructor
public WordCount() {
emitFrequency=5; // Default to 60 seconds
}
// Constructor that sets emit frequency
public WordCount(Integer frequency) {
emitFrequency=frequency;
}
//Configure frequency of tick tuples for this bolt
//This delivers a 'tick' tuple on a specific interval,
//which is used to trigger certain actions
@Override
public Map<String, Object> getComponentConfiguration() {
Config conf = new Config();
conf.put(Config.TOPOLOGY_TICK_TUPLE_FREQ_SECS, emitFre-
quency); return conf;
}
//execute is called to process tuples
```

Библиотека ГГТУ им. П.О. Сухого

```
@Override public void execute(Tuple tuple, BasicOutputCollector
collector) {
//If it's a tick tuple, emit all words and counts
if(tuple.getSourceComponent().equals(Constants.SYSTEM_COMPO
NENT_ID)
&& tu-
ple.getSourceStreamId().equals(Constants.SYSTEM_TICK_STREAM_ID))
{
for(String word : counts.keySet()) {
Integer count = counts.get(word);
collector.emit(new Values(word, count));
logger.info("Emitting a count of " + count + " for word " + word); }
} else {
//Get the word contents from the tuple
String word = tuple.getString(0); //Have we counted any already? In-
teger count = counts.get(word);
if (count == null)
count = 0;
//Increment the count and store it
count++;
counts.put(word, count);
}
}
//Declare that this emits a tuple containing two fields; word and
count
@Override
public void declareOutputFields(OutputFieldsDeclarer declarer) {
declarer.declare(new Fields("word", "count")); }
}
```

9.3. Определение топологии

Топология связывает и объединяет все вместе с графиком. Она также предоставляет указания по обеспечению параллелизма, которые *Storm* использует при создании экземпляров компонентов в кластере.

На рис. 9.1 приведена базовая диаграмма компонентов этой топологии.

Библиотека ГГТУ им. П.О.Сухого

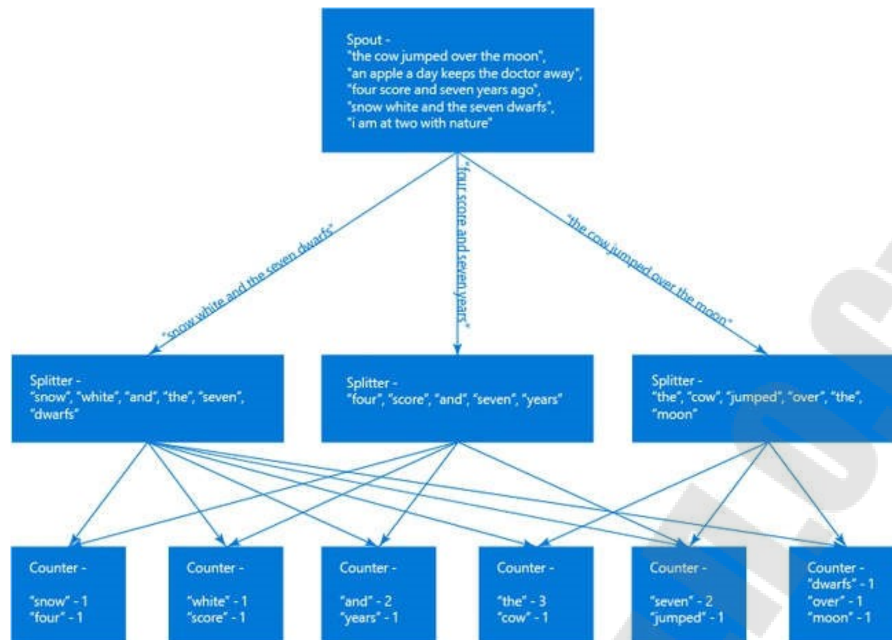


Рис. 9.1. Базовая диаграмма компонентов топологии

Для того чтобы реализовать топологию, введите следующую команду, чтобы создать и открыть новый файл *WordCountTopology.java*:

notepad src\main\java\com\microsoft\example\WordCountTopology.java

Затем скопируйте и вставьте приведенный ниже код Java в новый файл. Затем закройте файл.

WordCountTopology.java

```
package com.microsoft.example;
import org.apache.storm.Config;
import org.apache.storm.LocalCluster;
import org.apache.storm.StormSubmitter;
import org.apache.storm.topology.TopologyBuilder;
import org.apache.storm.tuple.Fields;
import com.microsoft.example.RandomSentenceSpout;
public class WordCountTopology {
//Entry point for the topology public static void main(String[] args) throws Exception {
//Used to build the topology
TopologyBuilder builder = new TopologyBuilder();
//Add the spout, with a name of 'spout' //and parallelism hint of 5 executors builder.setSpout("spout", new RandomSentenceSpout(), 5);
```

Библиотека ГГТУ им. П.О. Сухого

```
//Add the SplitSentence bolt, with a name of 'split'
//and parallelism hint of 8 executors
//shufflegrouping subscribes to the spout, and equally distributes
//tuples (sentences) across instances of the SplitSentence bolt
builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
//Add the counter, with a name of 'count'
//and parallelism hint of 12 executors
//fieldsgrouping subscribes to the split bolt, and
//ensures that the same word is sent to the same instance (
//group by field 'word')
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("sp-
lit", new Fields("word"));
//new configuration
Config conf = new Config();
//Set to false to disable debug information when
// running in production on a cluster conf.setDebug(false);
//If there are arguments, we are running on a cluster
if (args != null && args.length > 0) {
//parallelism hint to set the number of workers
conf.setNumWorkers(3);
//submit the topology
StormSubmitter.submitTopology(args[0], conf, builder.createTopo-
logy());
}
//Otherwise, we are running locally else {
//Cap the maximum number of executors that can be spawned
//for a component to 3
conf.setMaxTaskParallelism(3);
//LocalCluster is used to run locally
LocalCluster cluster = new LocalCluster();
//submit the topology cluster.submitTopology("word-count", conf,
builder.createTopology()); //sleep
Thread.sleep(10000); //shut down the cluster cluster.shutdown();
}
}


}
```


Библиотека ГГТУ им. П.О. Сухого

Настройка журнала

Для записи сведений в журнал используется *Apache log*4*j* 2. Если ведение журнала не настроено, топология выдает диагностическую информацию. Чтобы управлять ведением журнала, создайте файл с именем в каталоге, введя следующую команду:

notepad resources\log4j2.xml

Затем скопируйте и вставьте приведенный ниже *XML*-текст в новый файл. Далее закройте файл.

log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?> <Configuration>
<Appenders>
<Console name="STDOUT" target="SYSTEM_OUT">
<PatternLayout pattern="%d{HH:mm:ss} [%t] %-5level %logger{36} - %msg%n"/>
</Console>
</Appenders>
<Loggers>
<Logger name="com.microsoft.example" level="trace" additivity="false">
<AppenderRef ref="STDOUT"/>
</Logger>
<Root level="error">
<Appender-Ref ref="STDOUT"/>
</Root>
</Loggers>
</Configuration>
```

com.microsoft.example

Это позволит настроить новое средство ведения журнала для класса, который включает компоненты в этом примере топологии. Для этого средства ведения журнала задается уровень трассировки. Таким образом будут протоколироваться все сведения о журнале, генерируемые компонентами данной топологии:

<Root level="error">

В разделе для корневого уровня ведения журнала (все, что не находится в файле *com.microsoft.example*) настраивается регистрация только сведений об ошибках.

Библиотека ГГТУ им. П.О. Сухого

Глава 10. Параллельное программирование на основе MPI

10.1. MPI: основные понятия и определения

В вычислительных системах с распределенной памятью процессоры работают независимо друг от друга. Для организации параллельных вычислений в таких условиях необходимо иметь возможность распределять вычислительную нагрузку и организовать информационное взаимодействие (передачу данных) между процессорами.

Решение всех перечисленных вопросов и обеспечивает интерфейс передачи данных (*message passing interface – MPI*).

В общем плане для распределения вычислений между процессорами необходимо проанализировать алгоритм решения задачи, выделить информационно независимые фрагменты вычислений, провести их программную реализацию и затем разместить полученные части программы на разных процессорах. В рамках *MPI* принят более простой подход – для решения поставленной задачи разрабатывается одна программа и эта единственная программа запускается одновременно на выполнение на всех имеющихся процессорах. При этом для того чтобы избежать идентичности вычислений на разных процессорах, можно, во-первых, подставлять разные данные для программы на разных процессорах, а во-вторых, использовать имеющиеся в *MPI* средства для идентификации процессора, на котором выполняется программа (тем самым предоставляется возможность организовать различия в вычислениях в зависимости от используемого программой процессора).

Подобный способ организации параллельных вычислений получил наименование модели «одна программа – множество процессов» (*single program multiple processes or SPMP*).

Для организации информационного взаимодействия между процессорами в самом минимальном варианте достаточно операций приема и передачи данных (при этом, конечно, должна существовать техническая возможность коммуникации между процессорами – каналы или линии связи). В *MPI* существует целое множество операций передачи данных. Они обеспечивают разные способы пересылки данных, реализуют практически все коммуникационные операции. Именно данные возможности являются наиболее сильной стороной *MPI* (об этом, в частности, свидетельствует и само название *MPI*).

Библиотека ГГТУ им.П.О.Сухого

Следует отметить, что попытки создания программных средств передачи данных между процессорами начали предприниматься практически сразу с появлением локальных компьютерных сетей. Однако подобные средства часто были неполными и, самое главное, являлись несовместимыми. Таким образом, одна из самых серьезных проблем в программировании – переносимость программ при переводе программного обеспечения на другие компьютерные системы – проявлялась при разработке параллельных программ в максимальной степени. Как результат, уже с 90-х гг. стали предприниматься усилия по стандартизации средств организации передачи сообщений в многопроцессорных вычислительных системах. Началом работ, непосредственно приведших к появлению *MPI*, послужило проведение рабочего совещания по стандартам для передачи сообщений в среде *распределенной памяти*. По итогам совещания была образована рабочая группа, позднее преобразованная в международное сообщество *MPI Forum*, результатом деятельности которого явилось создание и принятие в 1994 г. стандарта *интерфейса передачи сообщений* (*message passing interface – MPI*) версии 1.0. В последующие годы стандарт *MPI* последовательно развивался. В 1997 г. был принят стандарт *MPI* версии 2.0.

Итак, теперь можно пояснить, что означает понятие *MPI*. Во-первых, *MPI* – это стандарт, которому должны удовлетворять средства организации передачи сообщений. Во-вторых, *MPI* – это программные средства, которые обеспечивают возможность передачи сообщений и при этом соответствуют всем требованиям стандарта *MPI*. Так, по стандарту, эти программные средства должны быть организованы в виде библиотек программных функций (*библиотеки MPI*) и должны быть доступны для наиболее широко используемых алгоритмических языков *C* и *Fortran*. Подобную «двойственность» *MPI* следует учитывать при использовании терминологии. Как правило, аббревиатура *MPI* применяется при упоминании стандарта, а сочетание «библиотека *MPI*» указывает на ту или иную программную реализацию стандарта. Однако достаточно часто для краткости обозначение *MPI* используется и для библиотек *MPI*, и тем самым для правильной интерпретации термина следует учитывать контекст.

Вопросы, связанные с разработкой *параллельных программ* с применением *MPI*, достаточно широко рассмотрены в литературе – краткий обзор полезных материалов содержится в конце данной главы. Здесь же, еще не приступая к изучению *MPI*, приведем ряд его важных положительных особенностей:

Библиотека ГГТУ им. П.О.Сухого

MPI позволяет в значительной степени снизить остроту проблемы переносимости *параллельных программ* между разными компьютерными системами – *параллельная программа*, разработанная на алгоритмическом языке *C* или *Fortran* с использованием библиотеки *MPI*, как правило, будет работать на разных вычислительных платформах;

MPI содействует повышению эффективности параллельных вычислений, поскольку в настоящее время практически для каждого типа вычислительных систем существуют реализации библиотек *MPI*, в максимальной степени учитывающие возможности компьютерного оборудования;

MPI уменьшает в определенном плане сложность разработки *параллельных программ*, так как, с одной стороны, основные операции передачи данных предусматриваются стандартом *MPI*, а с другой стороны, уже имеется большое количество библиотек параллельных методов, созданных с использованием *MPI*.

Рассмотрим ряд понятий и определений, являющихся основополагающими для стандарта *MPI*.

Понятие параллельной программы

Под *параллельной программой* в рамках *MPI* понимается множество одновременно выполняемых *процессов*. *Процессы* могут выполняться на разных процессорах, но на одном процессоре могут располагаться и несколько *процессов* (в этом случае их исполнение осуществляется в режиме разделения времени). В предельном случае для выполнения *параллельной программы* может использоваться один процессор – как правило, такой способ применяется для начальной проверки правильности *параллельной программы*.

Каждый *процесс параллельной программы* порождается на основе копии одного и того же программного кода (*модель SPMP*). Данный программный код, представленный в виде исполняемой программы, должен быть доступен в момент запуска *параллельной программы* на всех используемых процессорах. Исходный программный код для исполняемой программы разрабатывается на алгоритмических языках *C* или *Fortran* с применением той или иной реализации библиотеки *MPI*.

Количество *процессов* и число используемых процессоров определяется в момент запуска *параллельной программы* средствами среды исполнения *MPI*-программ и в ходе вычислений не может меняться без применения специальных, но редко задействуемых средств ди-

намического порождения *процессов* и управления ими, появившихся в стандарте *MPI* версии 2.0. Все *процессы* программы последовательно перенумерованы от 0 до *p*-1, где *p* есть общее количество *процессов*. Номер *процесса* именуется рангом *процесса*.

Операции передачи данных

Основу *MPI* составляют операции передачи сообщений. Среди предусмотренных в составе *MPI* функций различаются *парные* (*point-to-point*) операции между двумя *процессами* и *коллективные* (*collective*) коммуникационные действия для одновременного взаимодействия нескольких *процессов*.

Для выполнения парных операций могут использоваться разные *режимы передачи*, среди которых синхронный, блокирующий и др. Как отмечалось выше, в стандарт *MPI* включено большинство основных коллективных операций передачи данных.

Понятие коммуникаторов

Процессы параллельной программы объединяются в *группы*. Другим важным понятием *MPI*, описывающим набор *процессов*, является понятие коммуникатора. Под коммуникатором в *MPI* понимается специально создаваемый служебный объект, который объединяет в своем составе группу *процессов* и ряд дополнительных параметров (контекст), используемых при выполнении операций передачи данных.

Парные операции передачи данных выполняются только для *процессов*, принадлежащих одному и тому же коммуникатору. Коллективные операции применяются одновременно для всех *процессов* одного коммуникатора. Как результат, указание используемого коммуникатора является обязательным для операций передачи данных в *MPI*.

В ходе вычислений могут создаваться новые и удаляться существующие группы *процессов* и коммуникаторы. Один и тот же *процесс* может принадлежать разным группам и коммуникаторам. Все имеющиеся в *параллельной программе процессы* входят в состав конструируемого по умолчанию коммуникатора с идентификатором *MPI_COMM_WORLD*.

В версии 2.0 стандарта появилась возможность создавать глобальные коммуникаторы (*intercommunicator*), объединяющие в одну структуру пару групп при необходимости выполнения коллективных операций между *процессами* из разных групп.

Подробное рассмотрение возможностей *MPI* для работы с группами и коммуникаторами будет выполнено в параграфе 5.6.

Библиотека ГГТУ им. П.О.Сухого

Типы данных

При выполнении операций передачи сообщений для указания передаваемых или получаемых данных в функциях *MPI* необходимо указывать *тип пересылаемых данных*. *MPI* содержит большой набор *базовых типов данных*, во многом совпадающих с типами данных в алгоритмических языках *C* и *Fortran*. Кроме того, в *MPI* имеются возможности создания новых *производных типов* данных для более точного и краткого описания содержимого пересылаемых сообщений.

Виртуальные топологии

Как отмечалось выше, парные операции передачи данных могут быть выполнены между любыми *процессами* одного и того же коммуникатора, а в коллективной операции принимают участие все *процессы* коммуникатора. Логическая топология линий связи между *процессами* имеет структуру полного графа (независимо от наличия реальных физических каналов связи между процессорами).

Вместе с этим для изложения и последующего анализа ряда параллельных алгоритмов целесообразно логическое представление имеющейся коммуникационной сети в виде тех или иных топологий.

В *MPI* имеется возможность представления множества *процессов* в виде решетки произвольной размерности. При этом граничные *процессы* решеток могут быть объявлены соседними и тем самым на основе решеток могут быть определены структуры типа тор.

Кроме того, в *MPI* имеются средства и для формирования логических (виртуальных) топологий любого требуемого типа.

Наконец, последний ряд замечаний перед началом рассмотрения *MPI*:

– описание функций и все приводимые примеры программ будут представлены на алгоритмическом языке *C*;

– основное изложение возможностей *MPI* будет ориентировано на стандарт версии 1.2 (так называемый *MPI*-1).

Приступая к изучению *MPI*, можно отметить, что, с одной стороны, *MPI* достаточно сложен – в стандарте *MPI* предусматривается наличие более чем 120 функций. С другой стороны, структура *MPI* является тщательно продуманной – разработка *параллельных программ* может быть начата уже после рассмотрения всего лишь шести функций *MPI*. Все дополнительные возможности *MPI* могут осваиваться по мере роста сложности разрабатываемых алгоритмов и программ. Именное в таком стиле – от простого к сложному – и будет далее представлен весь учебный материал по *MPI*.

Библиотека ГГТУ им. П.О. Сухого

10.2. Введение в разработку параллельных программ с использованием MPI

Основы MPI

Приведем минимально необходимый набор функций MPI, достаточный для разработки сравнительно простых параллельных программ.

Инициализация и завершение *MPI*-программ

Первой вызываемой функцией *MPI* должна быть функция:

int MPI_Init(*int* **argc*, *char* ****argv*),

где

argc – указатель на количество параметров командной строки;

argv – параметры командной строки, применяемые для инициализации среды выполнения *MPI*-программы. Параметрами функции являются количество аргументов в командной строке и адрес указателя на массив символов текста самой командной строки.

Последней вызываемой функцией MPI обязательно должна являться функция:

int MPI_Finalize(*void*).

Как результат, можно отметить, что структура *параллельной программы*, разработанная с использованием *MPI*, должна иметь следующий вид:

```
#include "mpi.h"
int main(int argc, char *argv[]) {
 <программный код без использования функций MPI>
 MPI_Init(&argc, &argv);
 <программный код с использованием функций MPI>
 MPI_Finalize();
 <программный код без использования функций MPI>
 return 0;
}
```

Следует отметить:

– файл *mpi.h* содержит определения именованных констант, прототипов функций и типов данных библиотеки *MPI*;

– функции *MPI_Init* и *MPI_Finalize* являются обязательными и должны быть выполнены (и только один раз) каждым *процессом параллельной программы*;

Библиотека ГГТУ им. П.О. Сухого

перед вызовом *MPI_Init* может быть использована функция *MPI_Initialized* для определения того, был ли ранее выполнен вызов *MPI_Init*, а после вызова *MPI_Finalize* – *MPI_Finalized* аналогичного предназначения.

Рассмотренные примеры функций дают представление синтаксиса именования функций в *MPI*. Имени функции предшествует префикс *MPI*, далее следует одно или несколько слов названия, первое слово в имени функции начинается с заглавного символа, слова разделяются знаком подчеркивания. Названия функций *MPI*, как правило, поясняют назначение выполняемых функцией действий.

Определение количества и ранга процессов

Определение *количества процессов* в выполняемой *параллельной программе* осуществляется при помощи функции:

int MPI_Comm_size(*MPI_Comm comm*, *int *size*),

где

comm – коммуникатор, размер которого определяется;

size – определяемое количество *процессов* в коммуникаторе.

Для определения ранга *процесса* используется функция:

int MPI_Comm_rank(*MPI_Comm comm*, *int *rank*),

где

comm – коммуникатор, в котором определяется ранг *процесса*;

rank – ранг *процесса* в коммуникаторе.

Как правило, вызов функций *MPI_Comm_size* и *MPI_Comm_rank* выполняется сразу после *MPI_Init* для получения общего количества *процессов* и ранга текущего *процесса*:

```
#include "mpi.h"
int main(int argc, char *argv[]) {
 int ProcNum, ProcRank;
 <программный код без использования функций MPI>
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
 MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
 <программный код с использованием функций MPI>
 MPI_Finalize();
 <программный код без использования функций MPI>
 return 0;
}
```

Библиотека ГГТУ им. П.О.Сухого

Следует отметить:

– коммуникатор *MPI_COMM_WORLD*, как отмечалось выше, создается по умолчанию и представляет все *процессы* выполняемой *параллельной программы*;

ранг, получаемый при помощи функции *MPI_Comm_rank*, является рангом *процесса*, выполнившего вызов этой функции, т. е. переменная *ProcRank* примет различные значения у разных *процессов*.

Передача сообщений

Для *передачи сообщения процесс*-отправитель должен выполнить функцию:

int MPI_Send(*void *buf*, *int count*, *MPI_Datatype type*, *int dest*, *int tag*, *MPI_Comm comm*),

где

buf – адрес буфера памяти, в котором располагаются данные отправляемого сообщения;

count – количество элементов данных в сообщении;

type – тип элементов данных пересылаемого сообщения;

dest – ранг *процесса*, которому отправляется сообщение;

tag – значение-тег, используемое для идентификации сообщения;

comm – коммуникатор, в рамках которого выполняется передача данных.

Для указания типа пересылаемых данных в *MPI* имеется ряд базовых типов.

Следует отметить:

– отправляемое сообщение определяется через указание блока памяти (буфера), в котором это сообщение располагается. Используемая для указания буфера триада (*buf*, *count*, *type*) входит в состав параметров практически всех функций передачи данных;

– процессы, между которыми выполняется передача данных, в обязательном порядке должны принадлежать коммуникатору, указываемому в функции *MPI_Send*;

– параметр *tag* используется только при необходимости различения передаваемых сообщений, в противном случае в качестве значения параметра может быть использовано произвольное положительное целое число (см. также описание функции *MPI_Recv*).

Сразу же после завершения функции *MPI_Send процесс*-отправитель может начать повторно использовать буфер памяти, в котором располагалось отправляемое сообщение. Также следует понимать, что в момент завершения функции *MPI_Send* состояние само-

Библиотека ГГТУ им. П.О.Сухого

го пересылаемого сообщения может быть совершенно различным: сообщение может располагаться в *процессе*-отправителе, может находиться в состоянии передачи, может храниться в *процессе*-получателе или же может быть принято *процессом*-получателем при помощи функции *MPI_Recv*. Тем самым завершение функции *MPI_Send* означает лишь, что операция передачи начала выполняться и пересылка сообщения рано или поздно будет выполнена.

Пример использования функции будет представлен после описания функции *MPI_Recv*.

Прием сообщений

Для приема сообщения *процесс*-получатель должен выполнить функцию:

int MPI_Recv(*void *buf*, *int count*, *MPI_Datatype type*, *int source*,
int tag, *MPI_Comm comm*, *MPI_Status *status*),

где

buf, *count*, *type* – буфер памяти для приема сообщения, назначение каждого отдельного параметра соответствует описанию в *MPI_Send*;

source – ранг процесса, от которого должен быть выполнен прием сообщения;

tag – тег сообщения, которое должно быть принято для процесса;

comm – коммуникатор, в рамках которого выполняется передача данных;

status – указатель на структуру данных с информацией о результате выполнения операции приема данных.

Следует отметить:

– буфер памяти должен быть достаточным для приема сообщения. При нехватке памяти часть сообщения будет потеряна и в коде завершения функции будет зафиксирована ошибка переполнения; с другой стороны, принимаемое сообщение может быть и короче, чем размер приемного буфера, в таком случае изменятся только участки буфера, затронутые принятым сообщением;

– типы элементов передаваемого и принимаемого сообщения должны совпадать;

– при необходимости приема сообщения от любого процесса-отправителя для параметра *source* может быть указано значение *MPI_ANY_SOURCE* (в отличие от функции передачи *MPI_Send*, которая отсылает сообщение строго определенному адресату);

Библиотека ГГТУ им.П.О.Сухого

– при необходимости приема сообщения с любым тегом для параметра *tag* может быть указано значение *MPI_ANY_TAG* (опять-таки, при использовании функции *MPI_Send* должно быть указано конкретное значение тега);

– в отличие от параметров «процесс-получатель» и «тег», параметр «коммуникатор» не имеет значения, означающего «любой коммуникатор»;

– параметр *status* позволяет определить ряд характеристик принятого сообщения:

status.MPI_SOURCE – ранг процесса-отправителя принятого сообщения;

status.MPI_TAG – тег принятого сообщения.

Приведенные значения *MPI_ANY_SOURCE* и *MPI_ANY_TAG* иногда называют джокерами.

Значение переменной *status* позволяет определить количество элементов данных в принятом сообщении при помощи функции:

int MPI_Get_count(*MPI_Status *status*, *MPI_Datatype type*,
*int *count*),

где

status – статус операции *MPI_Recv*;

type – тип принятых данных;

count – количество элементов данных в сообщении.

Вызов функции *MPI_Recv* не обязан быть согласованным со временем вызова соответствующей функции передачи сообщения *MPI_Send* – прием сообщения может быть инициирован до момента, в момент или после момента начала отправки сообщения.

В завершение функции *MPI_Recv* в заданном буфере памяти будет располагаться принятое сообщение. Принципиальный момент здесь состоит в том, что функция *MPI_Recv* является *блокирующей* для *процесса*-получателя, т. е. его выполнение приостанавливается до завершения работы функции. Таким образом, если по каким-то причинам ожидаемое для приема сообщение будет отсутствовать, выполнение *параллельной программы* будет блокировано.

Первая параллельная программа с использованием *MPI*

Рассмотренный набор функций оказывается достаточным для разработки параллельных программ. Приводимая ниже программа является стандартным начальным примером для алгоритмического языка *C*:

Библиотека ГГТУ им.П.О.Сухого

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[]){
int ProcNum, ProcRank, RecvRank;
MPI_Status Status;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ){
// Действия, выполняемые только процессом с рангом 0
printf("\n Hello from process %3d", ProcRank);
for (int i = 1; i < ProcNum; i++ ) {
MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
printf("\n Hello from process %3d", RecvRank);
}
}
else // Сообщение, отправляемое всеми процессами,
// кроме процесса с рангом 0
MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
MPI_Finalize();
return 0;
}
```

Как следует из текста программы, каждый *процесс* определяет свой ранг, после чего действия в программе разделяются. Все *процессы*, кроме *процесса* с рангом 0, передают значение своего ранга нулевому *процессу*. *Процесс* с рангом 0 сначала печатает значение своего ранга, а далее последовательно принимает сообщения с рангами *процессов* и также печатает их значения. При этом важно отметить, что порядок приема сообщений заранее не определен и зависит от условий выполнения *параллельной программы* (более того, этот порядок может изменяться от запуска к запуску). Так, возможный вариант результатов печати *процесса* 0 может состоять в следующем (для *параллельной программы* из четырех *процессов*):

Hello from process 0
Hello from process 2
Hello from process 1
Hello from process 3

Библиотека ГГТУ им. П.О.Сухого

Такой «плавающий» вид получаемых результатов существенным образом усложняет разработку, тестирование и отладку *параллельных программ*, так как в этом случае исчезает один из основных принципов программирования – повторяемость выполняемых вычислительных экспериментов. Как правило, если это не приводит к потере эффективности, следует обеспечивать однозначность расчетов и при использовании параллельных вычислений. Для рассматриваемого простого примера можно восстановить постоянство получаемых результатов при помощи задания ранга *процесса*-отправителя в операции приема сообщения:

MPI_Recv(&*RecvRank*, 1, *MPI_INT*, *i*, *MPI_ANY_TAG*, *MPI_COMM_WORLD*, &*Status*).

Указание ранга *процесса*-отправителя регламентирует порядок приема сообщений, и, как результат, строки печати будут появляться строго в порядке возрастания рангов *процессов* (повторим, что такая регламентация в отдельных ситуациях может приводить к замедлению выполняемых параллельных вычислений).

Следует отметить еще один важный момент: разрабатываемая с применением *MPI* программа как в данном частном варианте, так и в самом общем случае используется для порождения всех *процессов параллельной программы*, а значит, должна определять вычисления, выполняемые всеми этими *процессами*. Можно отметить, что *MPI*-программа является некоторой «*макропрограммой*», различные части которой используются разными *процессами*. Так, в приведенном примере программы выделенные рамкой участки программного кода не выполняются одновременно ни одним из *процессов*. Первый выделенный участок с функцией приема *MPI_Recv* исполняется только *процессом* с рангом 0, второй участок с функцией передачи *MPI_Send* задействуется всеми *процессами*, за исключением нулевого *процесса*.

Для разделения фрагментов кода между *процессами* обычно используется подход, примененный в только что рассмотренной программе, – при помощи функции *MPI_Comm_rank* определяется ранг *процесса*, а затем в соответствии с рангом выделяются необходимые для *процесса* участки программного кода. Наличие в одной и той же программе фрагментов кода разных *процессов* также значительно усложняет понимание и в целом разработку *MPI*-программы. Как результат, можно рекомендовать при увеличении объема разрабатываемых программ выносить программный код разных *процессов* в отдельные программные модули (функции). Общая схема *MPI*-программы в этом случае будет иметь вид:

Библиотека ГГТУ им. П.О.Сухого

MPI_Comm_rank(*MPI_COMM_WORLD*, &*ProcRank*);
if (*ProcRank* == 0) *DoProcess*0();
else if (*ProcRank* == 1) *DoProcess*1();
else if (*ProcRank* == 2) *DoProcess*2();

Во многих случаях, как и в рассмотренном примере, выполняемые действия являются отличающимися только для *процесса* с рангом 0. В этом случае общая схема *MPI*-программы принимает более простой вид:

MPI_Comm_rank(*MPI_COMM_WORLD*, &*ProcRank*);
if (*ProcRank* == 0) *DoManagerProcess*();
else DoWorkerProcesses();

В завершение обсуждения примера поясним примененный в *MPI* подход для контроля правильности выполнения функций. Все функции *MPI* (кроме *MPI_Wtime* и *MPI_Wtick*) возвращают в качестве своего значения *код завершения*. При успешном выполнении функции возвращаемый код равен *MPI_SUCCESS*. Другие значения кода завершения свидетельствуют об обнаружении тех или иных ошибочных ситуаций в ходе выполнения функций. Для выяснения типа обнаруженной ошибки используются предопределенные именованные константы, среди которых:

MPI_ERR_BUFFER – неправильный указатель на буфер;

MPI_ERR_TRUNCATE – сообщение превышает размер приемного буфера;

MPI_ERR_COMM – неправильный коммуникатор;

MPI_ERR_RANK – неправильный ранг процесса и др.

Полный список констант для проверки кода завершения содержится в файле *mpi.h*. Однако, по умолчанию, возникновение любой ошибки во время выполнения функции *MPI* приводит к немедленному завершению *параллельной программы*. Для того чтобы иметь возможность проанализировать возвращаемый код завершения, необходимо воспользоваться предоставляемыми *MPI* функциями по созданию обработчиков ошибок и управлению ими, которые не рассматриваются в рамках данной главы.

Определение времени выполнение *MPI*-программы

Практически сразу после разработки первых *параллельных программ* возникает необходимость определения времени выполнения вычислений для оценки достигаемого ускорения *процессов* решения задач за счет использования параллелизма. Используемые обычно средства для измерения времени работы программ зависят, как пра-

Библиотека ГГТУ им. П.О.Сухого

вило, от аппаратной платформы, операционной системы, алгоритмического языка и т. п. Стандарт *MPI* включает определение специальных функций для измерения времени, применение которых позволяет устранить зависимость от среды выполнения *параллельных программ*.

Получение *текущего момента времени* обеспечивается при помощи функции:

double MPI_Wtime(*void*),

результат ее вызова есть количество секунд, прошедшее от некоторого определенного момента времени в прошлом. Этот момент времени в прошлом, от которого происходит отсчет секунд, может зависеть от среды реализации библиотеки *MPI*, и тем самым для ухода от такой зависимости функцию *MPI_Wtime* следует использовать только для определения длительности выполнения тех или иных фрагментов кода *параллельных программ*. Возможная схема применения функции *MPI_Wtime* может состоять в следующем:

*double t*1, *t*2, *dt*;
*t*1 = *MPI_Wtime*();
s
*t*2 = *MPI_Wtime*();
dt = *t*2 – *t*1;

Точность измерения времени также может зависеть от среды выполнения *параллельной программы*. Для определения текущего значения точности может быть использована функция:

double MPI_Wtick(*void*),

позволяющая определить время в секундах между двумя последовательными показателями времени аппаратного таймера примененной компьютерной системы.

Начальное знакомство с коллективными операциями передачи данных

Функции *MPI_Send* и *MPI_Recv* обеспечивают возможность выполнения *парных операций* передачи данных между двумя *процессами параллельной программы*. Для выполнения коммуникационных *коллективных операций*, в которых принимают участие все *процессы* коммуникатора, в *MPI* предусмотрен специальный набор функций.

Для демонстрации применения рассматриваемых функций *MPI* будет использоваться учебная *задача суммирования* элементов вектора *x*.

Разработка параллельного алгоритма для решения данной задачи не вызывает затруднений: необходимо разделить данные на равные блоки, передать эти блоки *процессам*, выполнить в *процессах* суммиро-

Библиотека ГГТУ им. П.О.Сухого

вание полученных данных, собрать значения вычисленных частных сумм на одном из *процессов* и сложить значения частичных сумм для получения общего результата решаемой задачи. При последующей разработке демонстрационных программ данный рассмотренный алгоритм будет несколько упрощен: *процессам* программы будет передаваться весь суммируемый вектор, а не отдельные блоки этого вектора.

Передача данных от одного процесса всем процессам программы

Первая задача при выполнении рассмотренного параллельного алгоритма суммирования состоит в необходимости передачи значений вектора *x* всем *процессам параллельной программы*. Конечно, для решения этой задачи можно воспользоваться рассмотренными ранее функциями парных операций передачи данных:

```
MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
for (int i = 1; i < ProcNum; i++)
 MPI_Send(&x, n, MPI_DOUBLE, i, 0, MPI_COMM_WORLD);
```

Однако такое решение будет крайне неэффективным, поскольку повторение операций передачи приводит к суммированию затрат (латентностей) на подготовку передаваемых сообщений (рис. 10.1). Данная операция может быть выполнена за $log_2 p$ итераций передачи данных.

Достижение эффективного выполнения операции передачи данных от одного *процесса* всем *процессам* программы (*широковещательная рассылка данных*) может быть обеспечено при помощи функции *MPI*:

```
int MPI_Bcast(void *buf, int count, MPI_Datatype type, int root,
 MPI_Comm comm),
```

где

buf, *count*, *type* – буфер памяти с отправляемым сообщением (для процесса с рангом 0) и для приема сообщений (для всех остальных процессов);

root – ранг процесса, выполняющего рассылку данных;

comm – коммуникатор, в рамках которого выполняется передача данных.

Функция *MPI_Bcast* осуществляет рассылку данных из буфера *buf*, содержащего *count* элементов типа *type*, с *процесса*, имеющего номер *root*, всем *процессам*, входящим в коммуникатор *comm*.

Следует отметить:

– функция *MPI_Bcast* определяет коллективную операцию и тем самым при выполнении необходимых рассылок данных вызов функ-

Библиотека ГГТУ им. П. О. Сухого

ции *MPI_Bcast* должен быть осуществлен всеми процессами указываемого коммуникатора (см. далее пример программы);

– указываемый в функции *MPI_Bcast* буфер памяти имеет различное назначение у разных процессов: для процесса с рангом *root*, которым осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение, а для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных;

– все коллективные операции «несовместимы» с парными операциями. Так, принять широковещательное сообщение, отосланное с помощью MPI_Bcast, функцией MPI_Recv нельзя, для этого можно задействовать только MPI_Bcast.

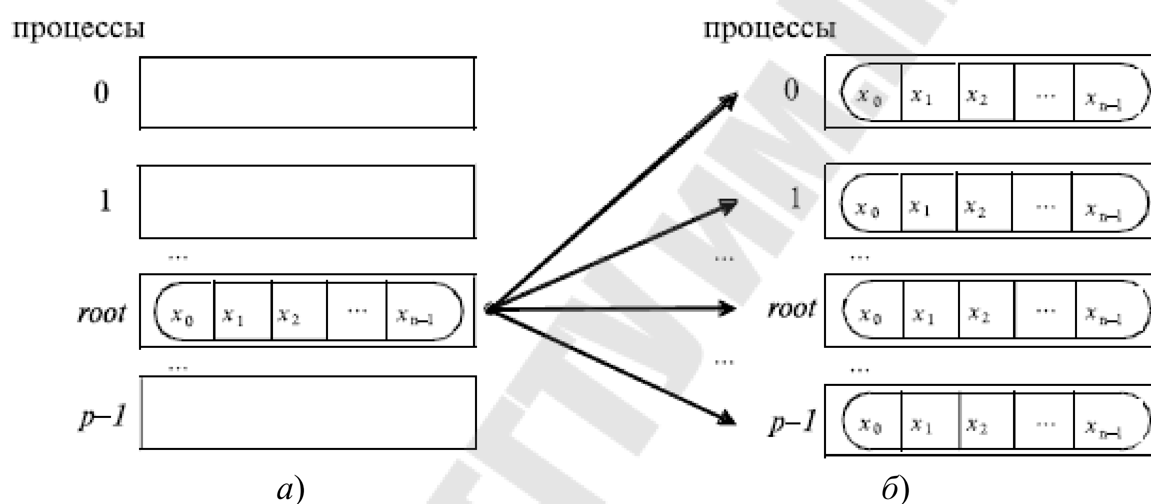


Рис. 10.1. Общая схема операции передачи данных от одного процесса всем процессам: *а* – до начала операции; *б* – после завершения операции

Приведем программу для решения учебной задачи суммирования элементов вектора с использованием рассмотренной функции:

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
int main(int argc, char* argv[]){
 double x[100], TotalSum, ProcSum = 0.0;
 int ProcRank, ProcNum, N=100, k, i1, i2;
 MPI_Status Status;
 // Инициализация
```


Библиотека ГГТУ им. П.О.Сухого

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
MPI_Comm_rank(MPI_COMM_WORLD,&ProcRank);
// Подготовка данных
if ( ProcRank == 0 ) DataInitialization(x,N);
// Рассылка данных на все процессы
MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// Вычисление частичной суммы на каждом из процессов
// на каждом процессе суммируются элементы вектора x от i1 до i2
k = N / ProcNum;
i1 = k * ProcRank;
i2 = k * ( ProcRank + 1 );
if ( ProcRank == ProcNum-1 ) i2 = N;
for ( int i = i1; i < i2; i++ )
ProcSum = ProcSum + x[i];
// Сборка частичных сумм на процессе с рангом 0
if ( ProcRank == 0 ) {
TotalSum = ProcSum;
for ( int i=1; i < ProcNum; i++ ) {
MPI_Recv(&ProcSum,1,MPI_DOUBLE,MPI_ANY_SOURCE,0,
MPI_COMM_WORLD, &Status);
TotalSum = TotalSum + ProcSum;
}
}
else // Все процессы отсылают свои частичные суммы
MPI_Send(&ProcSum, 1, MPI_DOUBLE, 0, 0,
MPI_COMM_WORLD);
// Вывод результата
if ( ProcRank == 0 )
printf("\nTotal Sum = %10.2f",TotalSum);
MPI_Finalize();
return 0;
}
```

В приведенной программе функция DataInitialization осуществляет подготовку начальных данных. Необходимые данные могут быть введены с клавиатуры, прочитаны из файла или сгенерированы при помощи датчика случайных чисел – подготовка этой функции предоставляется как задание для самостоятельной разработки.

Библиотека ГГТУ им.П.О.Сухого

Передача данных от всех процессов одному процессу. Операция редукции

В рассмотренной выше программе суммирования числовых значений имеющаяся процедура сбора и последующего суммирования данных является примером часто выполняемой коллективной операции передачи данных от всех процессов одному процессу. В этой операции над собираемыми значениями осуществляется та или иная обработка данных (для подчеркивания последнего момента данная операция еще именуется операцией редукции данных (рис. 10.20)). Как было изложено выше, реализация операции редукции при помощи обычных парных операций передачи данных является неэффективной и достаточно трудоемкой.

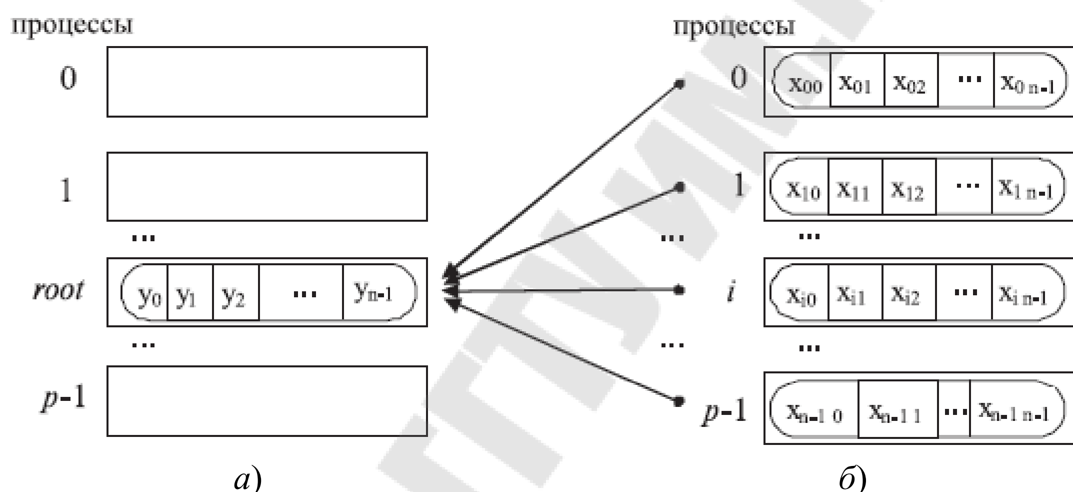


Рис. 10.2. Общая схема операции сбора и обработки на одном процессе данных от всех процессов: *а* – после завершения операции; *б* – до начала операции

Для наилучшего выполнения действий, связанных с редукцией данных, в *MPI* предусмотрена функция:

int MPI_Reduce(*void *sendbuf*, *void *recvbuf*, *int count*,
MPI_Datatype type, *MPI_Op op*, *int root*, *MPI_Comm comm*),

где

– *sendbuf* – буфер памяти с отправляемым сообщением;

– *recvbuf* – буфер памяти для результирующего сообщения (только для процесса с рангом *root*);

– *count* – количество элементов в сообщениях;

– *type* – тип элементов сообщений;

– *op* – операция, которая должна быть выполнена над данными;

Библиотека ГГТУ им. П.О. Сухого

– *root* – ранг процесса, на котором должен быть получен результат;

– *comm* – коммуникатор, в рамках которого выполняется операция.

В качестве операций редукции данных могут быть использованы предопределенные в *MPI* операции.

Помимо данного стандартного набора операций могут быть определены и новые дополнительные операции непосредственно самим пользователем библиотеки *MPI*.

Элементы получаемого сообщения на *процессе root* представляют собой результаты обработки соответствующих элементов передаваемых *процессами* сообщений.

Следует отметить:

– функция *MPI_Reduce* определяет коллективную операцию и тем самым вызов функции должен быть выполнен всеми процессами указываемого коммуникатора. При этом все вызовы функции должны содержать одинаковые значения параметров *count*, *type*, *op*, *root*, *comm*;

– передача сообщений должна быть выполнена всеми процессами, результат операции будет получен только процессом с рангом *root*;

– выполнение операции редукции осуществляется над отдельными элементами передаваемых сообщений. Так, если сообщения содержат по два элемента данных и выполняется операция суммирования *MPI_SUM*, то результат также будет состоять из двух значений, первое из которых будет содержать сумму первых элементов всех отправленных сообщений, а второе значение будет равно сумме вторых элементов сообщений соответственно (рис. 10.3);

– не все сочетания типа данных *type* и операции *op* возможны.

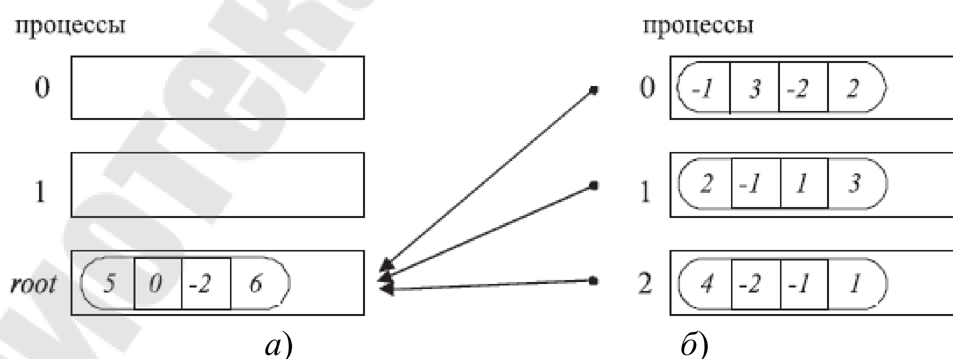


Рис. 10.3. Пример выполнения операции редукции при суммировании пересылаемых данных для трех процессов (в каждом сообщении четыре элемента, сообщения собираются на процессе с рангом 2): *а* – после завершения операции; *б* – до начала операции

Библиотека ГГТУ им. П.О.Сухого

Применим полученные знания для переработки ранее рассмотренной программы суммирования: как можно увидеть, весь программный код («Сборка частичных сумм на процессе с рангом 0»), может быть теперь заменен на вызов одной лишь функции *MPI_Reduce*:

// Сборка частичных сумм на процессе с рангом 0

MPI_Reduce(&*ProcSum*, &*TotalSum*, 1, *MPI_DOUBLE*, *MPI_SUM*, 0, *MPI_COMM_WORLD*);

Синхронизация вычислений

В ряде ситуаций независимо выполняемые в *процессах* вычисления необходимо синхронизировать. Так, для измерения времени начала работы *параллельной программы* необходимо, чтобы для всех *процессов* одновременно были завершены все подготовительные действия, перед окончанием работы программы все *процессы* должны завершить свои вычисления и т. п.

Синхронизация *процессов*, т. е. одновременное достижение *процессами* тех или иных точек *процесса* вычислений, обеспечивается при помощи функции *MPI*:

int MPI_Barrier(*MPI_Comm comm*),

где

comm – коммуникатор, в рамках которого выполняется операция.

Функция *MPI_Barrier* определяет коллективную операцию и тем самым при использовании она должна вызываться всеми *процессами* используемого коммуникатора. При вызове функции *MPI_Barrier* выполнение *процесса* блокируется, продолжение вычислений *процесса* произойдет только после вызова функции *MPI_Barrier* всеми *процессами* коммуникатора.

Аварийное завершение параллельной программы

Для корректного завершения *параллельной программы* в случае непредвиденных ситуаций необходимо использовать функцию:

int MPI_Abort(*MPI_Comm comm*, *int errorcode*),

где

comm – коммуникатор, процессы которого необходимо аварийно остановить;

errorcode – код возврата из параллельной программы.

Эта функция корректно прерывает выполнение *параллельной программы*, оповещая об этом событии среду *MPI*, в отличие от функций стандартной библиотеки алгоритмического языка *C*, таких как *abort* или *terminate*. Обычное ее использование заключается в следующем:

MPI_Abort(*MPI_COMM_WORLD*, *MPI_ERR_OTHER*);

Библиотека ГГТУ им. П.О. Сухого

10.3. Операции передачи данных между двумя процессами

Продолжим изучение функций *MPI* для выполнения операций передачи данных между *процессами параллельной программы*.

Режимы передачи данных

Рассмотренная ранее функция *MPI_Send* обеспечивает так называемый стандартный (*standard*) режим отправки сообщений, при котором:

– на время выполнения функции процесс-отправитель сообщения блокируется;

– после завершения функции буфер может быть использован повторно;

– состояние отправленного сообщения может быть различным – сообщение может располагаться на процессе-отправителе, может находиться в состоянии передачи, может храниться на процессе-получателе или может быть принято процессом-получателем при помощи функции *MPI_Recv*.

Кроме стандартного режима в *MPI* предусматриваются следующие дополнительные режимы *передачи сообщений*:

– синхронный (*synchronous*) режим состоит в том, что завершение функции отправки сообщения происходит только при получении от процесса-получателя подтверждения о начале приема отправленного сообщения. Отправленное сообщение или полностью принято процессом-получателем, или находится в состоянии приема;

– буферизованный (*buffered*) режим предполагает использование дополнительных системных или задаваемых пользователем буферов для копирования в них отправляемых сообщений. Функция отправки сообщения завершается сразу после копирования сообщения в системный буфер;

– режим передачи по готовности (*ready*) может быть использован только в том случае, если операция приема сообщения уже инициирована. Буфер сообщения после завершения функции отправки сообщения может быть повторно использован.

Для именования функций отправки сообщения для разных режимов выполнения в *MPI* применяется название функции *MPI_Send*, к которому как префикс добавляется начальный символ названия соответствующего режима работы, т. е.:

MPI_Ssend – функция отправки сообщения в синхронном режиме;

MPI_Bsend – функция отправки сообщения в буферизованном режиме;

Библиотека ГГТУ им. П.О. Сухого

MPI_Rsend – функция отправки сообщения в режиме по готовности.

Список параметров всех перечисленных функций совпадает с составом параметров функции *MPI_Send*.

Для применения буферизованного *режима передачи* может быть создан и передан *MPI* буфер памяти. Используемая для этого функция имеет вид:

int MPI_Buffer_attach(*void *buf*, *int size*),

где

buf – адрес буфера памяти;

size – размер буфера.

После завершения работы с буфером он должен быть отключен от *MPI* при помощи функции:

int MPI_Buffer_detach(*void *buf*, *int *size*),

где

buf – адрес буфера памяти;

size – возвращаемый размер буфера.

По практическому использованию режимов можно привести следующие рекомендации:

– стандартный режим обычно реализуется как буферизированный или синхронный, в зависимости от размера передаваемого сообщения, и зачастую является наиболее оптимизированным по производительности;

– режим передачи по готовности формально является наиболее быстрым, но используется достаточно редко, так как обычно сложно гарантировать готовность операции приема;

– буферизованный режим также выполняется достаточно быстро, но может приводить к большим расходам ресурсов (памяти);

– в целом может быть рекомендован для передачи коротких сообщений;

– синхронный режим является наиболее медленным, так как требует подтверждения приема, однако не нуждается в дополнительной памяти для хранения сообщения. Этот режим может быть рекомендован для передачи длинных сообщений.

В заключение отметим, что для функции приема *MPI_Recv* не существует различных режимов работы.

Организация неблокирующих обменов данными между процессами

Все рассмотренные выше функции отправки и приема сообщений являются *блокирующими*, т. е. приостанавливающими выполне-

Библиотека ГГТУ им. П.О. Сухого

ние *процессов* до момента завершения работы вызванных функций. В то же время при выполнении параллельных вычислений часть сообщений может быть отправлена и принята заранее, до момента реальной потребности в пересылаемых данных. В таких ситуациях желательно иметь возможность выполнить функцию обмена данными без блокировки *процессов* для совмещения *процессов передачи сообщений* и вычислений. Такой неблокирующий способ выполнения обменов является, конечно, более сложным для использования, но при правильном применении может в значительной степени уменьшить потери эффективности параллельных вычислений из-за медленных (по сравнению с быстродействием процессоров) коммуникационных операций.

MPI обеспечивает возможность неблокированного выполнения операций передачи данных между двумя *процессами*. Наименование неблокирующих аналогов образуется из названий соответствующих функций путем добавления префикса *I* (*Immediate*). Список параметров неблокирующих функций содержит обычный набор параметров исходных функций и один дополнительный параметр *request* с типом *MPI_Request* (в функции *MPI_Irecv* отсутствует также параметр *status*):

int MPI_Isend(*void *buf*, *int count*, *MPI_Datatype type*, *int dest*,
int tag, *MPI_Comm comm*, *MPI_Request *request*),
int MPI_Issend(*void *buf*, *int count*, *MPI_Datatype type*, *int dest*,
int tag, *MPI_Comm comm*, *MPI_Request *request*),
int MPI_Ibsend(*void *buf*, *int count*, *MPI_Datatype type*, *int dest*,
int tag, *MPI_Comm comm*, *MPI_Request *request*),
int MPI_Irsend(*void *buf*, *int count*, *MPI_Datatype type*, *int dest*,
int tag, *MPI_Comm comm*, *MPI_Request *request*),
int MPI_Irecv(*void *buf*, *int count*, *MPI_Datatype type*, *int source*,
int tag, *MPI_Comm comm*, *MPI_Request *request*).

Вызов неблокирующей функции приводит к инициации запрошенной операции передачи, после чего выполнение функции завершается и *процесс* может продолжить свои действия. Перед своим завершением неблокирующая функция определяет переменную *request*, которая далее может использоваться для проверки завершения инициированной операции обмена.

Проверка состояния выполняемой неблокирующей операции передачи данных производится при помощи функции:

int MPI_Test(*MPI_Request *request*, *int *flag*, *MPI_status *status*),

Библиотека БГУИР

где

request – дескриптор операции, определенный при вызове неблокирующей функции;

flag – результат проверки (*true*, если операция завершена);

status – результат выполнения операции обмена (только для завершенной операции).

Операция проверки является неблокирующей, т. е. *процесс* может проверить состояние неблокирующей операции обмена и продолжить далее свои вычисления, если по результатам проверки окажется, что операция все еще не завершена. Возможная схема совмещения вычислений и выполнения неблокирующей операции обмена может состоять в следующем:

```
MPI_Isend(buf, count, type, dest, tag, comm, &request);
s
do {
 s
 MPI_Test(&request, &flag, &status);
} while (!flag);
```

Если при выполнении неблокирующей операции окажется, что продолжение вычислений невозможно без получения передаваемых данных, то может быть использована блокирующая операция ожидания завершения операции:

int MPI_Wait(*MPI_Request* **request*, *MPI_status* **status*),

где

request – дескриптор операции, определенный при вызове неблокирующей функции;

status – результат выполнения операции обмена (только для завершенной операции).

Кроме рассмотренных *MPI* содержит ряд дополнительных функций проверки и ожидания неблокирующих операций обмена:

MPI_Testall – проверка завершения всех перечисленных операций обмена;

MPI_Waitall – ожидание завершения всех операций обмена;

MPI_Testany – проверка завершения хотя бы одной из перечисленных операций обмена;

MPI_Waitany – ожидание завершения любой из перечисленных операций обмена;

MPI_Testsome – проверка завершения каждой из перечисленных операций обмена;

Библиотека ГГТУ им. П.О.Сухого

MPI_Waitsome – ожидание завершения хотя бы одной из перечисленных операций обмена и оценка состояния по всем операциям.

Приведение простого примера использования неблокирующих функций достаточно затруднительно. Хорошей возможностью для освоения рассмотренных функций могут служить, например, параллельные алгоритмы матричного умножения.

Одновременное выполнение передачи и приема

Одной из часто выполняемых форм информационного взаимодействия в *параллельных программах* является обмен данными между *процессами*, когда для продолжения вычислений *процессам* необходимо отправить данные одним *процессам* и в то же время получить сообщения от других. Простейший вариант этой ситуации состоит, например, в обмене данными между двумя *процессами*. Реализация таких обменов при помощи обычных парных операций передачи данных может быть неэффективна, кроме того, такая реализация должна гарантировать отсутствие тупиковых ситуаций, которые могут возникать, например, когда два *процесса* начинают передавать сообщения друг другу с использованием блокирующих функций передачи данных.

Достижение эффективного и гарантированного одновременного выполнения операций передачи и приема данных может быть обеспечено при помощи функции *MPI*:

int MPI_Sendrecv(*void *sbuf*,*int scount*,*MPI_Datatype stype*,
int dest, *int stag*, *void *rbuf*,*int rcount*,*MPI_Datatype rtype*,
int source,*int rtag*, *MPI_Comm comm*, *MPI_Status *status*),

где

sbuf, *scount*, *stype*, *dest*, *stag* – параметры передаваемого сообщения;

rbuf, *rcount*, *rtype*, *source*, *rtag* – параметры принимаемого сообщения;

comm – коммуникатор, в рамках которого выполняется передача данных;

status – структура данных с информацией о результате выполнения операции.

Как следует из описания, функция *MPI_Sendrecv* передает сообщение, описываемое параметрами (*sbuf*, *scount*, *stype*, *dest*, *stag*), *процессу* с рангом *dest* и принимает сообщение в буфер, определяемый параметрами (*rbuf*, *rcount*, *rtype*, *source*, *rtag*), от *процесса* с рангом *source*.

В функции *MPI_Sendrecv* для передачи и приема сообщений применяются разные буферы. В том случае если отсылаемое сообще-

Библиотека ГГТУ им.П.О.Сухого

ние больше не нужно на *процессе*-отправителе, в *MPI* имеется возможность использования единого буфера:

int MPI_Sendrecv_replace(*void *buf*, *int count*, *MPI_Datatype type*, *int dest*, *int stag*, *int source*, *int rtag*, *MPI_Comm comm*, *MPI_Status* status*),

где

buf, *count*, *type* – параметры передаваемого сообщения;

dest – ранг процесса, которому отправляется сообщение;

stag – тег для идентификации отправляемого сообщения;

source – ранг процесса, от которого выполняется прием сообщения;

rtag – тег для идентификации принимаемого сообщения;

comm – коммуникатор, в рамках которого выполняется передача данных;

status – структура данных с информацией о результате выполнения операции.

Коллективные операции передачи данных

Как было изложено выше, под *коллективными операциями* в *MPI* понимаются операции над данными, в которых принимают участие все *процессы* используемого коммуникатора. Часть коллективных операций была представлена выше – это операции передачи от одного *процесса* всем *процессам* коммуникатора (*широковещательная рассылка*) и операции обработки данных, полученных на одном *процессе* от всех *процессов* (*редукция данных*).

Рассмотрим далее оставшиеся базовые коллективные операции передачи данных.

Обобщенная передача данных от одного процесса всем процессам

Обобщенная операция передачи данных от одного *процесса* всем *процессам* (*распределение данных*) отличается от широковещательной рассылки тем, что *процесс* передает *процессам* различающиеся данные (рис. 10.4). Выполнение данной операции может быть обеспечено при помощи функции:

int MPI_Scatter(*void *sbuf*, *int scount*, *MPI_Datatype stype*, *void *rbuf*, *int rcount*, *MPI_Datatype rtype*, *int root*, *MPI_Comm comm*),

где

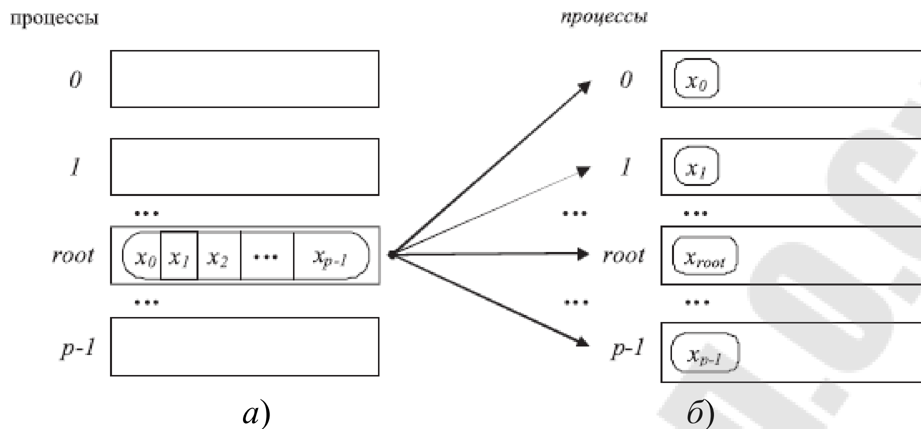
sbuf, *scount*, *stype* – параметры передаваемого сообщения (scount определяет количество элементов, передаваемых на каждый процесс);

rbuf, *rcount*, *rtype* – параметры сообщения, принимаемого в процессах;

root – ранг процесса, выполняющего рассылку данных;

Библиотека ГГТУ им. П.О.Сухого

comm – коммуникатор, в рамках которого выполняется передача данных.



а) *б*)

Рис. 10.4. Общая схема операции обобщенной передачи данных от одного процесса всем процессам: *а* – до начала операции; *б* – после завершения операции

При вызове этой функции процесс с рангом root произведет передачу данных всем другим процессам в коммуникаторе. Каждому процессу будет отправлено scount элементов. Процесс с рангом 0 получит блок данных из sbuf элементов с индексами от 0 до scount-1, процессу с рангом 1 будет отправлен блок из sbuf элементов с индексами от scount до 2*scount-1 и т. д. Тем самым общий размер отправляемого сообщения должен быть равен scount * *p* элементов, где *p* есть количество процессов в коммуникаторе comm.

Следует отметить, что поскольку функция MPI_Scatter определяет коллективную операцию, вызов этой функции при выполнении рассылки данных должен быть обеспечен на каждом процессе коммуникатора.

Отметим также, что функция *MPI_Scatter* передает всем *процессам* сообщения одинакового размера. Выполнение более общего варианта операции распределения данных, когда размеры сообщений для *процессов* могут быть различны, обеспечивается при помощи функции *MPI_Scatterv*.

Пример использования функции *MPI_Scatter* рассматривается при разработке *параллельных программ* умножения матрицы на вектор.

Обобщенная передача данных от всех процессов одному процессу

Операция обобщенной передачи данных от всех *процессов* одному *процессу* (сбор данных) является двойственной к процедуре

Библиотека ГГТУ им. П.О.Сухого

распределения данных (рис. 10.5). Для выполнения этой операции в *MPI* предназначена функция:

int MPI_Gather(*void *sbuf*, *int scount*, *MPI_Datatype stype*,
*void *rbuf*, *int rcount*, *MPI_Datatype rtype*,
int root, *MPI_Comm comm*),

где

- *sbuf*, *scount*, *stype* – параметры передаваемого сообщения;
- *rbuf*, *rcount*, *rtype* – параметры принимаемого сообщения;
- *root* – ранг процесса, выполняющего сбор данных;
- *comm* – коммуникатор, в рамках которого выполняется передача данных.

При выполнении функции *MPI_Gather* каждый *процесс* в коммуникаторе передает данные из буфера *sbuf* на *процесс* с рангом *root*. *Процесс* с рангом *root* собирает все получаемые данные в буфере *rbuf* (размещение данных в буфере осуществляется в соответствии с рангами *процессов* – отправителей сообщений). Для того чтобы разместить все поступающие данные, размер буфера *rbuf* должен быть равен *scount* * *p* элементов, где *p* есть количество *процессов* в коммуникаторе *comm*.

Функция *MPI_Gather* также определяет коллективную операцию, и ее вызов при выполнении сбора данных должен быть обеспечен в каждом *процессе* коммуникатора.

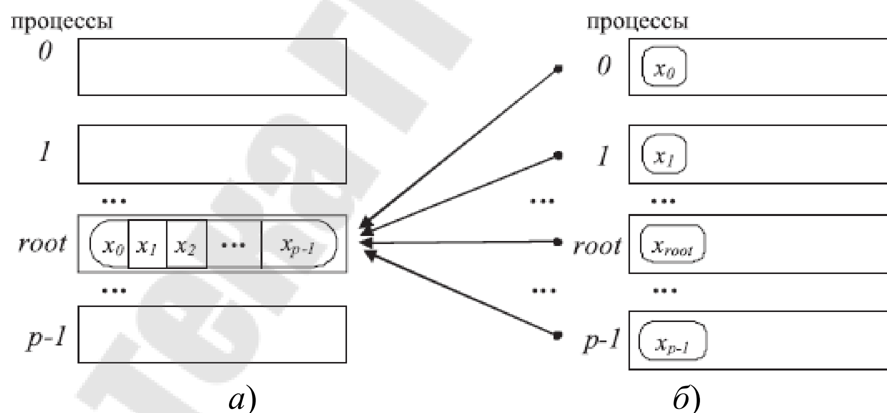


Рис. 10.5. Общая схема операции обобщенной передачи данных от всех процессов одному процессу:
а – до начала операции; *б* – после завершения операции

Следует отметить, что при использовании функции MPI_Gather сборка данных осуществляется только на одном процессе. Для получения всех собираемых данных на каждом из процессов коммуникатора необходимо применять функцию сбора и рассылки:

int MPI_Allgather(*void *sbuf, int scount, MPI_Datatype stype,*
*void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm*),

где

sbuf, *scount*, *stype* – параметры передаваемого сообщения;

rbuf, *rcount*, *rtype* – параметры принимаемого сообщения;

comm – коммуникатор, в рамках которого выполняется передача данных.

Выполнение общего варианта операции сбора данных, когда размеры передаваемых процессами сообщений могут быть различны, обеспечивается при помощи функций MPI_Gatherv и MPI_Allgatherv.

Пример использования функции MPI_Gather рассматривается при разработке параллельных программ умножения матрицы на вектор.

Общая передача данных от всех процессов всем процессам

Передача данных от всех процессов всем процессам является наиболее общей операцией передачи данных (рис. 10.6). Выполнение данной операции может быть обеспечено при помощи функции:

int MPI_Alltoall(*void *sbuf*,*int scount*,*MPI_Datatype stype*,
*void *rbuf*,*int rcount*,*MPI_Datatype rtype*,*MPI_Comm comm*),

где

sbuf, *scount*, *stype* – параметры передаваемых сообщений;

rbuf, *rcount*, *rtype* – параметры принимаемых сообщений;

comm – коммуникатор, в рамках которого выполняется передача данных.

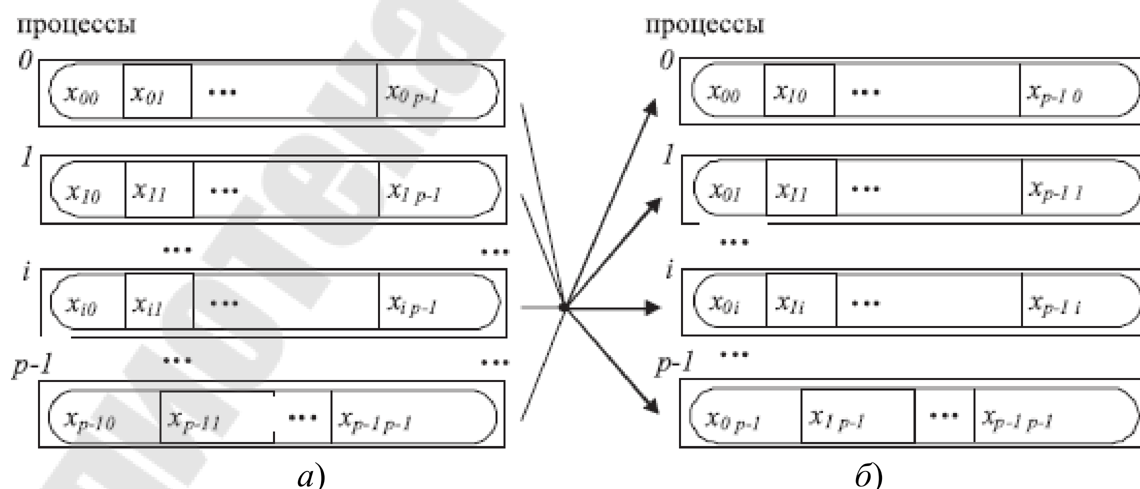


Рис. 10.6. Общая схема операции передачи данных от всех процессов всем процессам: *а* – до начала операции; *б* – после завершения операции

Библиотека ГГТУ им. П.О.Сухого

При выполнении функции *_Alltoall MPI* каждый *процесс* в коммуникаторе передает данные из *scount* элементов каждому *процессу* (общий размер отправляемых сообщений в *процессах* должен быть равен *scount * p* элементов, где *p* есть количество *процессов* в коммуникаторе *comm*) и принимает сообщения от каждого *процесса*.

Вызов функции *MPI_Alltoall* при выполнении операции общего обмена данными должен быть выполнен в каждом *процессе* коммуникатора.

Вариант операции общего обмена данных, когда размеры передаваемых *процессами* сообщений могут быть различны, обеспечивается при помощи функций *MPI_Alltoallv*.

Пример использования функции *MPI_Alltoall* рассматривается при разработке *параллельных программ* умножения матрицы на вектор как задание для самостоятельного выполнения.

Дополнительные операции редукции данных

Функция *MPI_Reduce* обеспечивает получение результатов редукции данных только на одном *процессе*. Для получения результатов редукции данных на каждом из *процессов* коммуникатора необходимо использовать функцию редукции и рассылки:

*int MPI_Allreduce(void *sendbuf, void *recvbuf, int count,*
MPI_Datatype type, MPI_Op op, MPI_Comm comm),

где

- *sendbuf* – буфер памяти с отправляемым сообщением;
- *recvbuf* – буфер памяти для результирующего сообщения;
- *count* – количество элементов в сообщениях;
- *type* – тип элементов сообщений;
- *op* – операция, которая должна быть выполнена над данными;
- *comm* – коммуникатор, в рамках которого выполняется операция.

Функция *MPI_Allreduce* выполняет рассылку между *процессами* всех результатов операции редукции. Возможность управления распределением этих данных между *процессами* предоставляется функцией *MPI_Reduce_scatter*.

И еще один вариант операции сбора и обработки данных, при котором обеспечивается получение всех частичных результатов редуцирования, может быть реализован при помощи функции:

*int MPI_Scan(void *sendbuf, void *recvbuf, int count,*
MPI_Datatype type, MPI_Op op, MPI_Comm comm),

Библиотека ГГТУ им.П.О.Сухого

где

- *sendbuf* – буфер памяти с отправляемым сообщением;
- *recvbuf* – буфер памяти для результирующего сообщения;
- *count* – количество элементов в сообщениях;
- *type* – тип элементов сообщений;
- *op* – операция, которая должна быть выполнена над данными;
- *comm* – коммуникатор, в рамках которого выполняется операция.

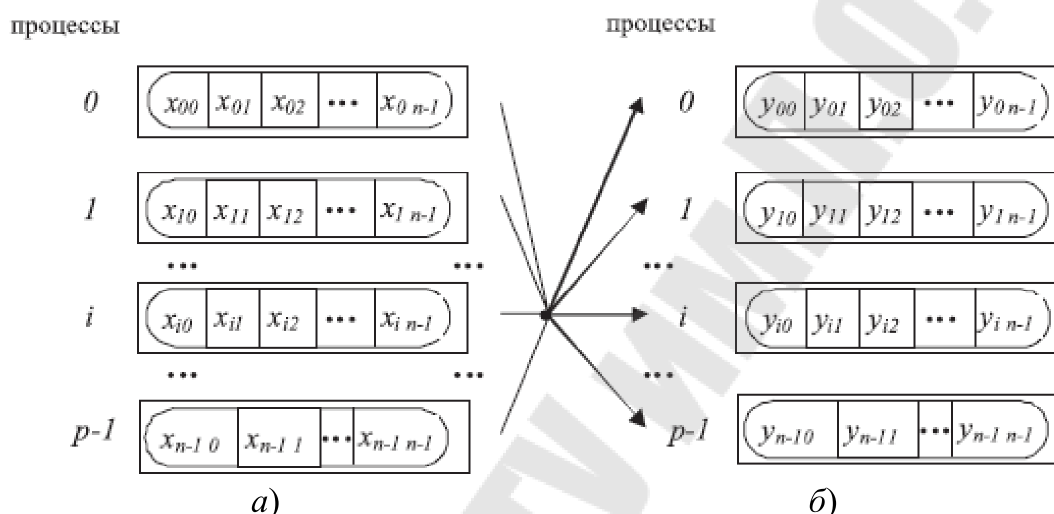


Рис. 10.7. Общая схема операции редукции с получением частичных результатов обработки данных:
а – до начала операции; *б* – после завершения операции

Общая схема выполнения функции *MPI_Scan* показана на рис. 10.7. Элементы получаемых сообщений представляют собой результаты обработки соответствующих элементов передаваемых *процессами* сообщений, при этом для получения результатов на *процессе* с рангом i, $0 \le i < n$, используются данные от *процессов*, ранг которых меньше или равен i.

10.4. Производные типы данных в MPI

Во всех рассмотренных выше примерах использования функций передачи данных предполагалось, что сообщения представляют собой некоторый непрерывный вектор элементов предусмотренного в *MPI* типа. Понятно, что в общем случае необходимые к пересылке данные могут рядом не располагаться и состоять из значений разных типов. Конечно, и в этих ситуациях разрозненные данные могут быть пере-

Библиотека ГГТУ имени П.О. Сухого

даны с использованием нескольких сообщений, но такой способ решения неэффективен в силу накопления латентностей множества выполняемых операций передачи данных. Другой возможный подход состоит в предварительной упаковке передаваемых данных в формат того или иного непрерывного вектора, однако и здесь появляются лишние операции копирования данных, да и понятность таких операций передачи далека от желаемой.

Для обеспечения больших возможностей при определении состава передаваемых сообщений в *MPI* предусмотрен механизм так называемых производных типов данных. Далее будут даны основные понятия используемого подхода, приведены возможные способы конструирования производных типов данных и рассмотрены функции упаковки и распаковки данных.

Понятие производного типа данных

В самом общем виде под *производным типом данных* в *MPI* можно понимать описание набора значений предусмотренного в *MPI* типа, причем в общем случае описываемые значения не обязательно непрерывно располагаются в памяти. Задание типа в *MPI* принято осуществлять при помощи *карты типа* (*type map*) в виде последовательности описаний, входящих в тип значений; каждое отдельное значение описывается указанием типа и смещения адреса месторасположения от некоторого базового адреса, т. е.

$TypeMap = \{(type_0, disp_0),...,(type_{n-1}, disp_{n-1})\}$.

Часть карты типа с указанием только типов значений именуется в *MPI сигнатурой типа*:

$TypeSignature = \{type_0,...,type_{n-1}\}$.

Сигнатура типа описывает, какие базовые типы данных образуют некоторый *производный тип данных MPI*, и тем самым управляет интерпретацией элементов данных при передаче или получении сообщений. Смещения карты типа определяют, где находятся значения данных.

Поясним рассмотренные понятия на следующем примере. Пусть в сообщение должны входить значения переменных:

double a; /* адрес 24 */
double b; /* адрес 40 */
int n; /* адрес 48 */

Тогда производный тип для описания таких данных должен иметь карту типа следующего вида:

{(*MPI_DOUBLE*, 0),
(*MPI_DOUBLE*, 16),

Библиотека ГГТУ им. П.О.Сухого

(*MPI_INT*, 24)}

Согласно определению, *нижняя граница* есть смещение для первого байта значений рассматриваемого типа данных. Соответственно, *верхняя граница* представляет собой смещение для байта, располагающегося вслед за последним элементом рассматриваемого типа данных. При этом величина смещения для верхней границы может быть округлена вверх с учетом требований выравнивания адресов. Так, одно из требований, которые налагают некоторые реализации языков *C* и *Fortran*, состоит в том, чтобы адрес элемента был кратен длине этого элемента в байтах. Например, если тип *int* занимает четыре байта, то адрес элемента типа *int* должен нацело делиться на четыре. Именно это требование и отражается в определении верхней границы типа данных *MPI*. Поясним данный момент на рассмотренном выше примере набора переменных *a*, *b* и *n*, для которого нижняя граница равна 0, а верхняя принимает значение 32 (величина округления 6 или 4 в зависимости от размера типа *int*). Здесь следует отметить, что требуемое выравнивание определяется по типу первого элемента данных в карте типа.

Следует также указать на различие понятий «протяженность» и «размер типа». *Протяженность* – это размер памяти в байтах, который нужно отводить для одного элемента производного типа. *Размер типа данных* – это число байтов, которые занимают данные (разность между адресами последнего и первого байтов данных). Различие в значениях протяженности и размера опять же в величине округления для выравнивания адресов. Так, в рассматриваемом примере размер типа равен 28, а протяженность – 32 (предполагается, что тип *int* занимает четыре байта).

Для получения значения протяженности типа в *MPI* предусмотрена функция:

int MPI_Type_extent(*MPI_Datatype type*, *MPI_Aint *extent*),

где

type – тип данных, протяженность которого отыскивается;

extent – протяженность типа.

Размер типа можно найти, используя функцию:

int MPI_Type_size(*MPI_Datatype type*, *MPI_Aint *size*),

где

type – тип данных, размер которого отыскивается;

size – размер типа.

Определение нижней и верхней границ типа может быть выпол-

Библиотека ГГТУ им. П.О.Сухого

нено при помощи функций:

int MPI_Type_lb(*MPI_Datatype type*, *MPI_Aint *disp*) и

int MPI_Type_ub(*MPI_Datatype type*, *MPI_Aint *disp*),

где

type – тип данных, нижняя граница которого отыскивается;

disp – нижняя/верхняя граница типа.

Важной и необходимой при конструировании производных типов является функция получения адреса переменной:

int MPI_Address(*void *location*, *MPI_Aint *address*),

где

location – адрес памяти;

address – адрес памяти в переносимом *MPI*-формате.

(Следует отметить, что данная функция является переносимым вариантом средств получения адресов в алгоритмических языках *C* и *Fortran.*)

Способы конструирования производных типов данных

Для снижения сложности в *MPI* предусмотрено несколько различных способов конструирования производных типов:

– непрерывный способ позволяет определить непрерывный набор элементов существующего типа как новый производный тип;

– векторный способ обеспечивает создание нового производного типа как набора элементов существующего типа, между элементами которого имеются регулярные промежутки по памяти. При этом размер промежутков задается в числе элементов исходного типа, в то время как в варианте *H*-векторного способа этот размер указывается в байтах;

– индексный способ отличается от векторного метода тем, что промежутки между элементами исходного типа могут иметь нерегулярный характер (имеется и *H*-индексный способ, отличающийся способом задания промежутков);

– структурный способ обеспечивает самое общее описание производного типа через явное указание карты создаваемого типа данных.

Далее перечисленные способы конструирования производных типов данных будут рассмотрены более подробно.

Непрерывный способ конструирования

При *непрерывном способе* конструирования производного типа данных в *MPI* используется функция:

int MPI_Type_contiguous(*int count*, *MPI_Data_type oldtype*,
*MPI_Datatype *newtype*),

Библиотека ГГТУ им. П.О.Сухого

где

count – количество элементов исходного типа;

oldtype – исходный тип данных;

newtype – новый определяемый тип данных.

Как следует из описания, новый тип *newtype* создается как *count* элементов исходного типа *oldtype*. Например, если исходный тип данных имеет карту типа

{(*MPI_INT*, 0), (*MPI_DOUBLE*, 8)},

то вызов функции *MPI_Type_contiguous* с параметрами

MPI_Type_contiguous(2, *oldtype*, &*newtype*);

приведет к созданию типа данных с картой типа

{(*MPI_INT*, 0), (*MPI_DOUBLE*, 8), (*MPI_INT*, 16), (*MPI_DOUBLE*, 24)}.

В определенном плане наличие непрерывного способа конструирования является избыточным, поскольку использование аргумента *count* в процедурах *MPI* равносильно использованию непрерывного типа данных такого же размера.

Векторный способ конструирования

При *векторном способе* конструирования производного типа данных в *MPI* применяются функции:

int MPI_Type_vector(*int count*, *int blocklen*, *int stride*,
MPI_Data_type oldtype, *MPI_Datatype* **newtype*) и

int MPI_Type_hvector(*int count*, *int blocklen*, *MPI_Aint stride*,
MPI_Data_type oldtype, *MPI_Datatype* **newtype*),

где

count – количество блоков;

blocklen – размер каждого блока;

stride – количество элементов, расположенных между двумя соседними блоками;

oldtype – исходный тип данных;

newtype – новый определяемый тип данных.

Отличие способа конструирования, определяемого функцией *MPI_Type_hvector*, состоит лишь в том, что параметр *stride* для определения интервала между блоками задается в байтах, а не в элементах исходного типа данных.

Как следует из описания, при векторном способе новый производный тип создается как набор блоков из элементов исходного типа, при этом между блоками могут иметься регулярные промежутки по памяти. Приведем несколько примеров использования данного способа конструирования типов:

Библиотека ГГТУ им.П.О.Сухого

– конструирование типа для выделения половины (только четных или только нечетных) строк матрицы размером $n \times n$:

MPI_Type_vector(*n* / 2, *n*, 2 * *n*, &*StripRowType*, &*ElemType*),

– конструирование типа для выделения столбца матрицы размером $n \times n$:

MPI_Type_vector(*n*, 1, *n*, &*ColumnType*, &*ElemType*),

– конструирование типа для выделения главной диагонали матрицы размером $n \times n$:

MPI_Type_vector(*n*, 1, *n* + 1, &*DiagonalType*, &*ElemType*).

С учетом характера приводимых примеров можно упомянуть имеющуюся в *MPI* возможность создания производных типов для описания подмассивов многомерных массивов при помощи функции (данная функция предусматривается стандартом *MPI*-2):

int MPI_Type_create_subarray(*int ndims*, *int *sizes*, *int *subsizes*, *int *starts*, *int order*, *MPI_Data_type oldtype*, *MPI_Datatype *newtype*),

где

ndims – размерность массива;

sizes – количество элементов в каждой размерности исходного массива;

subsizes – количество элементов в каждой размерности определяемого подмассива;

starts – индексы начальных элементов в каждой размерности определяемого подмассива;

order – параметр для указания необходимости переупорядочения;

oldtype – тип данных элементов исходного массива;

newtype – новый тип данных для описания подмассива.

Индексный способ конструирования

При *индексном способе* конструирования производного типа данных в *MPI* используются функции:

int MPI_Type_indexed(*int count*, *int blocklens[]*, *int indices[]*, *MPI_Data_type oldtype*, *MPI_Datatype *newtype*) и

int MPI_Type_hindexed(*int count*, *int blocklens[]*, *MPI_Aint indices[]*, *MPI_Data_type oldtype*, *MPI_Datatype *newtype*),

где

count – количество блоков;

blocklens – количество элементов в каждом блоке;

Библиотека ГГТУ им. П.О.Сухого

indices – смещение каждого блока от начала типа;
oldtype – исходный тип данных;
newtype – новый определяемый тип данных.

Как следует из описания, при индексном способе новый производный тип создается как набор блоков разного размера из элементов исходного типа, при этом между блоками могут иметься разные промежутки по памяти. Для пояснения данного способа можно привести пример конструирования типа для описания верхней треугольной матрицы размером $n \times n$:

```
// Конструирование типа для описания верхней треугольной матрицы
for ( i = 0, i < n; i++ ) {
blocklens[i] = n - i;
indices [i] = i * n + i;
}
MPI_Type_indexed(n, blocklens, indices, &UTMatrixType, &Elem Type).
```

Способ конструирования, определяемый функцией *MPI_Type_ hindexed*, отличается тем, что элементы *indices* для определения интервалов между блоками задаются в байтах, а не в элементах исходного типа данных.

Следует отметить, что существует еще одна дополнительная функция *MPI_Type_create_indexed_block* индексного способа конструирования для определения типов с блоками одинакового размера (данная функция предусматривается стандартом *MPI*-2).

Структурный способ конструирования

Как отмечалось выше, *структурный способ* является самым общим методом конструирования производного типа данных при явном задании соответствующей карты типа. Использование такого способа производится при помощи функции:

```
int MPI_Type_struct(int count, int blocklens[], MPI_Aint indices[],
 MPI_Data_type oldtypes[], MPI_Datatype *newtype),
```

где

count – количество блоков;
blocklens – количество элементов в каждом блоке;
indices – смещение каждого блока от начала типа (в байтах);
oldtypes – исходные типы данных в каждом блоке в отдельности;
newtype – новый определяемый тип данных.

Библиотека ГГТУ им. П.О.Сухого

Как следует из описания, структурный способ дополнительно к индексному методу позволяет указывать типы элементов для каждого блока в отдельности.

Объявление производных типов и их удаление

Приведенные выше функции конструирования позволяют определить *производный тип данных*. Дополнительно перед использованием созданный тип *должен быть объявлен* при помощи функции:

int MPI_Type_commit(*MPI_Datatype* **type*),

где

type – объявляемый тип данных.

При завершении использования производный тип *должен быть аннулирован* при помощи функции:

int MPI_Type_free(*MPI_Datatype* **type*),

где

type – аннулируемый тип данных.

Формирование сообщений при помощи упаковки и распаковки данных

Наряду с рассмотренными выше методами конструирования производных типов в *MPI* предусмотрен и явный способ сборки и разборки сообщений, содержащих значения разных типов и располагаемых в разных областях памяти.

Для использования данного подхода должен быть определен буфер памяти достаточного размера для сборки сообщения. Входящие в состав сообщения данные должны быть упакованы в буфер при помощи функции:

int MPI_Pack(*void* **data*, *int count*, *MPI_Datatype type*, *void* **buf*,
int bufsize, *int* **bufpos*, *MPI_Comm comm*),

где

data – буфер памяти с элементами для упаковки;

count – количество элементов в буфере;

type – тип данных для упаковываемых элементов;

buf – буфер памяти для упаковки;

bufsize – размер буфера в байтах;

bufpos – позиция для начала записи в буфер (в байтах от начала буфера);

comm – коммуникатор для упакованного сообщения.

Функция *MPI_Pack* упаковывает *count* элементов из буфера *data* в буфер упаковки *buf*, начиная с позиции *bufpos*. Общая схема процедуры упаковки показана на рис. 10.8.

Библиотека ГГТУ им. П.О.Сухого

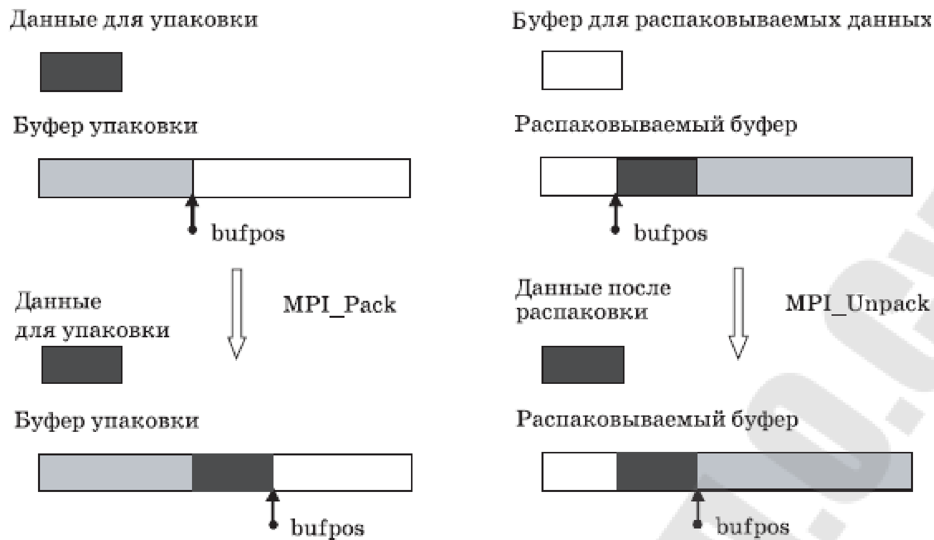


Рис. 10.8. Общая схема упаковки и распаковки данных:
а – упаковка данных; *б* – распаковка данных

Начальное значение переменной *bufpos* должно быть сформировано до начала упаковки и далее устанавливается функцией *MPI_Pack*. Вызов функции *MPI_Pack* осуществляется последовательно для упаковки всех необходимых данных. Так, в вышерассмотренном примере набора переменных *a*, *b* и *n* для их упаковки необходимо выполнить:

bufpos = 0;
MPI_Pack(*&a, 1, MPI_DOUBLE, buf, buflen, &bufpos, comm*);
MPI_Pack(*&b, 1, MPI_DOUBLE, buf, buflen, &bufpos, comm*);
MPI_Pack(*&n, 1, MPI_INT, buf, buflen, &bufpos, comm*);

Для определения необходимого размера буфера для упаковки может быть применена функция:

int MPI_Pack_size(*int count, MPI_Datatype type, MPI_Comm comm, int *size*),

где

count – количество элементов в буфере;
type – тип данных для упаковываемых элементов;
comm – коммуникатор для упакованного сообщения;
size – рассчитаный размер буфера.

После упаковки всех необходимых данных подготовленный буфер может быть использован в функциях передачи данных с указанием типа MPI_PACKED.

После получения сообщения с типом MPI_PACKED данные могут быть распакованы при помощи функции:

Библиотека ГГТУ им. П.О.Сухого

int MPI_Unpack(*void *buf, int bufsize, int *bufpos, void *data, int count, MPI_Datatype type, MPI_Comm comm*),

где

buf – буфер памяти с упакованными данными;

bufsize – размер буфера в байтах;

bufpos – позиция начала данных в буфере (в байтах от начала буфера);

data – буфер памяти для распаковываемых данных;

count – количество элементов в буфере;

type – тип распаковываемых данных;

comm – коммуникатор для упакованного сообщения.

Функция MPI_Unpack распаковывает, начиная с позиции bufpos, очередную порцию данных из буфера *buf* и помещает распакованные данные в буфер data.

Начальное значение переменной *bufpos* должно быть сформировано до начала распаковки и далее устанавливается функцией *MPI_Unpack*. Вызов функции *MPI_Unpack* осуществляется последовательно для распаковки всех упакованных данных, при этом порядок распаковки должен соответствовать порядку упаковки. Так, в приведенном выше примере упаковки для распаковки упакованных данных необходимо выполнить:

bufpos = 0;
MPI_Unpack(*buf, buflen, &bufpos, &a, 1, MPI_DOUBLE, comm*);
MPI_Unpack(*buf, buflen, &bufpos, &b, 1, MPI_DOUBLE, comm*);
MPI_Unpack(*buf, buflen, &bufpos, &n, 1, MPI_INT, comm*);

В заключение предложим ряд рекомендаций по применению упаковки для формирования сообщений. Поскольку такой подход приводит к появлению дополнительных действий по упаковке и распаковке данных, то данный способ может быть оправдан при сравнительно небольших размерах сообщений и при малом количестве повторений. Упаковка и распаковка может оказаться полезной при явном использовании буферов для буферизованного способа передачи данных.

10.5. Управление группами процессов и коммуникаторами

Рассмотрим теперь возможности *MPI* по управлению группами *процессов* и коммуникаторами.

Для изложения последующего материала напомним ряд понятий и определений, приведенных в данной главе.

Библиотека ГГТУ им. П.О. Сухого

Процессы параллельной программы объединяются в *группы*. В группу могут входить все *процессы параллельной программы*; с другой стороны, в группе может находиться только часть имеющихся *процессов*. Один и тот же *процесс* может принадлежать нескольким группам. Управление группами *процессов* предпринимается для создания на их основе коммуникаторов.

Под коммуникатором в *MPI* понимается специально создаваемый служебный объект, который объединяет в своем составе группу *процессов* и ряд дополнительных параметров (контекст), используемых при выполнении операций передачи данных. Парные операции передачи данных выполняются только для *процессов*, принадлежащих одному и тому же коммуникатору. Коллективные операции применяются одновременно для всех *процессов* коммуникатора. Создание коммуникаторов предпринимается для уменьшения области действия коллективных операций и для устранения взаимовлияния разных выполняемых частей *параллельной программы*. Важно еще раз подчеркнуть – коммуникационные операции, выполняемые с использованием разных коммуникаторов, являются независимыми и не влияют друг на друга.

Все имеющиеся в *параллельной программе процессы* входят в состав создаваемого по умолчанию коммуникатора с идентификатором *MPI_COMM_WORLD*.

При необходимости передачи данных между *процессами* из разных групп необходимо создавать определенные в стандарте *MPI*-2 глобальные коммуникаторы (*intercommunicator*). Взаимодействие между *процессами* разных групп оказывается необходимым в достаточно редких ситуациях, в данной книге не рассматривается и может служить темой для самостоятельного изучения.

Управление группами

Группы *процессов* могут быть созданы только из уже существующих групп. В качестве исходной группы может быть использована группа, связанная с предопределенным коммуникатором *MPI_COMM_WORLD*. Также иногда может быть полезным коммуникатор *MPI_COMM_SELF*, определенный для каждого *процесса параллельной программы* и включающий только этот *процесс*.

Для получения группы, связанной с существующим коммуникатором, применяется функция:

int MPI_Comm_group(*MPI_Comm comm*, *MPI_Group *group*),

где

Библиотека ГГТУ им. П.О.Сухого

comm – коммуникатор;

group – группа, связанная с коммуникатором.

Далее на основе существующих групп могут быть созданы новые группы:

создание новой группы *newgroup* из существующей группы *oldgroup*, которая будет включать в себя *n* процессов – их ранги перечисляются в массиве *ranks*:

int MPI_Group_incl(*MPI_Group oldgroup*, *int n*, *int *ranks*,
*MPI_Group *newgroup*),

где

oldgroup – существующая группа;

n – число элементов в массиве *ranks*;

ranks – массив рангов процессов, которые будут включены в новую группу;

newgroup – созданная группа;

создание новой группы *newgroup* из группы *oldgroup*, которая будет включать в себя *n* процессов, чьи ранги не совпадают с рангами, перечисленными в массиве *ranks*:

int MPI_Group_excl(*MPI_Group oldgroup*, *int n*, *int *ranks*,
*MPI_Group *newgroup*),

где

oldgroup – существующая группа;

n – число элементов в массиве *ranks*;

ranks – массив рангов процессов, которые будут исключены из новой группы;

newgroup – созданная группа.

Для получения новых групп над имеющимися группами *процессов* могут быть выполнены операции объединения, пересечения и разности:

создание новой группы *newgroup* как объединения групп *group*1 и *group*2:

int MPI_Group_union(*MPI_Group group*1, *MPI_Group group*2,
*MPI_Group *newgroup*),

где

*group*1 – первая группа;

*group*2 – вторая группа;

newgroup – объединение групп;

создание новой группы *newgroup* как пересечения групп *group*1 и *group*2:

Библиотека ГГТУ им. П.О. Сухого

int MPI_Group_intersection(*MPI_Group group*1, *MPI_Group group*2, *MPI_Group *newgroup*),
где

*group*1 – первая группа;

*group*2 – вторая группа;

newgroup – пересечение групп;

создание новой группы *newgroup* как разности групп *group*1 и *group*2:

int MPI_Group_difference(*MPI_Group group*1, *MPI_Group group*2, *MPI_Group *newgroup*),

где

*group*1 – первая группа;

*group*2 – вторая группа;

newgroup – разность групп;

При конструировании групп может оказаться полезной специальная пустая группа *MPI_COMM_EMPTY*.

Ряд функций *MPI* обеспечивает получение информации о группе *процессов*:

получение количества процессов в группе:

int MPI_Group_size(*MPI_Group group*, *int *size*),

где

group – группа;

size – число процессов в группе;

получение ранга текущего процесса в группе:

int MPI_Group_rank(*MPI_Group group*, *int *rank*),

где

group – группа;

rank – ранг процесса в группе.

После завершения использования группа должна быть удалена:

int MPI_Group_free(*MPI_Group *group*),

где

group – группа, подлежащая удалению

Выполнение данной операции не затрагивает коммуникаторы, в которых используется удаляемая группа.

Управление коммуникаторами

Отметим прежде всего, что в рамках данного контекста рассматривается управление *интракоммуникаторами*, используемыми для операций передачи данных внутри одной группы *процессов*. Как было изложено выше, применение *интеркоммуникаторов* для обменов ме-

Библиотека ГГТУ им.П.О.Сухого

жду группами *процессов* выходит за пределы данного учебного материала.

Для создания новых коммуникаторов существуют два основных способа:

– дублирование уже существующего коммуникатора:

int MPI_Comm_dup(*MPI_Comm oldcom*, *MPI_comm* **newcom*),

где

– *oldcom* – существующий коммуникатор, копия которого создается;

– *newcom* – новый коммуникатор;

– создание нового коммуникатора из подмножества процессов существующего коммуникатора:

int MPI_comm_create(*MPI_Comm oldcom*, *MPI_Group group*,
MPI_Comm **newcom*),

где

oldcom – существующий коммуникатор;

group – подмножество процессов коммуникатора *oldcom*;

newcom – новый коммуникатор.

Дублирование коммуникатора может применяться, например, для устранения возможности пересечения по тегам сообщений в разных частях *параллельной программы* (в том числе и при использовании функций разных программных библиотек).

Следует отметить также, что операция создания коммуникаторов является коллективной и тем самым должна выполняться всеми *процессами* исходного коммуникатора.

Для пояснения рассмотренных функций можно привести пример создания коммуникатора, в котором содержатся все *процессы*, кроме *процесса*, имеющего ранг 0 в коммуникаторе *MPI_COMM_-WORLD* (такой коммуникатор может быть полезен для поддержки схемы организации параллельных вычислений «менеджер – исполнители»):

MPI_Group WorldGroup, *WorkerGroup*;
MPI_Comm Workers;
*int ranks[*1*]*;
*ranks[*0*]* = 0;
// Получение группы процессов в *MPI_COMM_WORLD*
MPI_Comm_group(*MPI_COMM_WORLD*, &*WorldGroup*);
// Создание группы без процесса с рангом 0
MPI_Group_excl(*WorldGroup*, 1, *ranks*, &*WorkerGroup*);

// Создание коммуникатора по группе

MPI_Comm_create(*MPI_COMM_WORLD*, *WorkerGroup*, &*Workers*);

...

MPI_Group_free(&*WorkerGroup*);

MPI_Comm_free(&*Workers*);

Быстрый и полезный способ одновременного создания нескольких коммуникаторов обеспечивает функция:

int MPI_Comm_split(*MPI_Comm oldcomm*, *int split*, *int key*, *MPI_Comm *newcomm*),

где

oldcomm – исходный коммуникатор;

split – номер коммуникатора, которому должен принадлежать процесс;

key – порядок ранга процесса в создаваемом коммуникаторе;

newcomm – создаваемый коммуникатор.

Создание коммуникаторов относится к коллективным операциям, и поэтому вызов функции *MPI_Comm_split* должен быть выполнен в каждом *процессе* коммуникатора *oldcomm*. В результате выполнения функции *процессы* разделяются на непересекающиеся группы с одинаковыми значениями параметра *split*. На основе сформированных групп создается набор коммуникаторов. Для того чтобы указать, что *процесс* не должен входить ни в один из создаваемых коммуникаторов, необходимо воспользоваться константой *MPI_UNDEFINED* в качестве значения параметра *split*. При создании коммуникаторов для рангов *процессов* в новом коммуникаторе выбирается такой порядок нумерации, чтобы он соответствовал порядку значений параметров *key* (*процесс* с большим значением параметра *key* получает больший ранг, *процессы* с одинаковым значением параметра *key* сохраняют свою относительную нумерацию).

В качестве примера можно рассмотреть задачу представления набора *процессов* в виде двумерной решетки. Пусть $p = q * q$ есть общее количество *процессов*; следующий далее фрагмент программы обеспечивает получение коммуникаторов для каждой строки создаваемой топологии:

MPI_Comm comm;

int rank, *row*;

MPI_Comm_rank(*MPI_COMM_WORLD*, &*rank*);

row = *rank* / *q*;

MPI_Comm_split(*MPI_COMM_WORLD*, *row*, *rank*, &*comm*);

Библиотека ГГТУ им. П.О. Сухого

При выполнении данного примера, например, при $p = 9$, *процессы* с рангами (0, 1, 2) образуют первый коммуникатор, *процессы* с рангами (3, 4, 5) – второй и т. д.

После завершения использования коммуникатор должен быть удален, для чего используется функция:

int MPI_Comm_free(*MPI_Comm* **comm*),

где

comm – коммуникатор, подлежащий удалению.

Библиотека ГГТУ им.П.О.Сухого

Контрольные вопросы

1. Распределенная обработка данных, *Cloud Computing*, концепция «облачной» обработки данных.

2. Предпосылки возникновения облачных информационных систем.

3. Технологии построение облачных информационных систем.

4. Требования к облачным информационным системам.

5. Особенности построения облачных информационных систем на основе технологии *Windows Azure*.

6. Основные модели предоставления услуг облачных вычислений: *Software as a Service* (*SaaS*) (ПО-как-услуга), *Platform as a Service* (*PaaS*), Инфраструктура как сервис (*Infrastructure as a Service*, *IaaS*), другие облачные сервисы (*XaaS*).

7. Различия между облачными и кластерными (распределенными или *Grid*-технологиями) вычислениями.

8. Платформа *Windows Azure.*

9. Использование облачной платформы *Windows Azure* для разработки приложений.

10. Инструментальные средства *Windows Azure*.

11. Развертывание приложений на платформу *Windows Azure* с использованием сервисов – *Windows Azure Web Sites* и *Windows Azure Cloud Services*.

12. Создание и развертывание простого веб-приложения *ASP.NET MVC* 4 с помощью сервисов *Windows Azure Web Sites* и *Windows Azure Cloud Services* и *Visual Studio* 2012.

13. Разработка приложений с *Windows Azure Cloud Services*.

14. Использование *Windows Azure* как *Platform-As-A-Service*, архитектура, использование, разработка многослойных приложений *ASP.NET*. Развертывание облачных приложений в облако, сложное масштабирование всех слоев по отдельности.

15. Архитектура *Windows Azure Cloud Services*, конфигурация *Cloud Service*, масштабирование *Cloud Service*, использование *Windows Azure Tools for Visual Studio*, *Windows Azure SDK*.

16. Авторизация и безопасность с *Windows Azure Active Directory.*

17. Введение в технологии аутентификации на базе утверждений, реализация сценариев аутентификации с использованием технологий *Microsoft*, сценарий интеграции облачного приложения с локальной инфраструктурой *Active Directory* для реализации *Single Sign-On* и федеративной аутентификации.

Библиотека ГГТУ им. П.О. Сухого

18. Аутентификация на базе утверждений. Федеративная аутентификация в *Windows Azure* с использованием публичных провайдеров идентификации и доменного каталога *Active Directory*.

19. Программная реализация проверки токенов безопасности на стороне клиента.

20. Программная модель *Windows Identity Foundation*.

21. Технологии *Windows Azure Access Control Service*, *Active Directory Federation Services* 2.0. Многофакторная проверка подлинности *Windows Azure*.

22. Облачные системы хранения и обработки корпоративных данных.

23. Хранение и обработка данных с *Windows Azure Storage* и *Windows Azure SQL Databases.*

24. Сценарий приложения *Cloud Services*, использующего для хранения данных блобы, таблицы и очереди *Windows Azure*.

25. Сценарий приложения *Cloud Services* с хранением данных в базе данных.

26. Введение в масштабирование баз данных *Windows Azure* – федерации, шардинг.

27. Бизнес-аналитика и анализ данных с *SQL Reporting* и *Hadoop.*

28. Введение в бизнес-аналитику.

29. Введение в парадигму *MapReduce*.

30. Приложения для бизнес-аналитики с *SQL Reporting*.

31. Приложение, анализирующее логи, с использованием *M/R Hadoop*, в *Windows Azure*.

32. Доступ к сервисам предприятия с *Windows Azure Service Bus.*

33. Принципы осуществления доступа к сервисам предприятия с использованием *Service Bus* в *Cloud Services* для безопасной и надежной передачи данных.

34. Интеграция облачного приложения с сервисом предприятия.

35. Использование технологии *Windows Azure Service Bus*.

36. *Windows Azure Notification Hubs*.

37. Транзакции в *Windows Azure Service Bus*.

38. Определение дубликатов сообщений.

39. Разработка приложений высокой готовности в *Windows Azure*. Назначение, примеры использования.

40. *DAO*-слой доступа к данным в *Windows Azure*.

Библиотека ГГТУ им. П.О.Сухого

Литература

1. Федоров, А. Windows Azure. Облачная платформа Microsoft / А. Федоров, Д. Мартынов, 2010. – 96 с.

2. OpenStack. – Режим доступа: http://www.openstack.org/. Дата доступа: 05.03.2019.

3. Cloud Foundry Режим доступа: http://www.cloudfoundry.com/. – Дата доступа: 15.03.2019.

4. Топ 10 облачных платформ для бизнеса. – Режим доступа: http://www.livebusiness.ru/news/8937/. – Дата доступа: 15.03.2019.

5. Amazon Web Sevices. – Режим доспупа: http://aws.amazon.com/. – Дата доступа: 15.03.2019.

6. Rockspace. – Режим доступа: http://www.rackspace.com/. – Дата доступа: 15.03.2019.

7. Rackspace Cloud Tools Marketplace. – Режим доспупа: http://www.rackspace.com/cloud/tools/. – Дата доступа: 15.03.2019.

8. Платформа Windows Azure. – Режим доступа: http://www.windowsazure.com/ru-ru/. – Дата доступа: 10.11.2018.

9. Google App Engine – Режим доступа: https://developers.google.com/appengine/?hl=ru. – Дата доступа: 15.03.2019.

10. Force.com. – Режим доступа: http://www.force.com/. – Дата доступа: 15.03.2019.

11. Salesforce.com. – Режим доступа: http://www.salesforce.com/. – Дата доступа: 15.03.2019.

12. VMware vCloud® Suite. – Режим доступа: http://www.vmware.com/products/datacenter-virtualization/vcloud-suite/overview.html. – Дата доступа: 15.03.2019.

13. IBM Smart Cloud. – Режим доступа: http://www.ibm.com/cloud-computing/us/en/. – Дата доступа: 15.03.2019.

Библиотека БГТУ им. П.О. Сухого

Учебное электронное издание комбинированного распространения

Учебное издание

Стефановский Игорь Леонидович
Курочка Константин Сергеевич

ОБЛАЧНЫЕ ТЕХНОЛОГИИ И СРЕДСТВА ОБРАБОТКИ БОЛЬШИХ ОБЪЕМОВ ИНФОРМАЦИИ

**Учебно-методическое пособие
для студентов специальностей 1-40 05 01
«Информационные системы и технологии
(по направлениям)»
и 1-40 80 04«Информатика и технологии
программирования»
дневной и заочной форм обучения**

Электронный аналог печатного издания

Редактор Н. Г. Мансурова
Компьютерная верстка И. П. Минина

Подписано в печать 06.04.21.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 11,62. Уч.-изд. л. 12,28.
Изд. № 20.
http://www.gstu.by

Издатель и полиграфическое исполнение
Гомельский государственный
технический университет имени П. О. Сухого.
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за № 1/273 от 04.04.2014 г.
пр. Октября, 48, 246746, г. Гомель

Библиотека ГГТУ им. П.О. Сухого