ГОЛУБОЙ РАСПРЕДЕЛ

Делаем систему распределенных вычислений на Windows Azure

Есть у нас в редакции специальные весы, на которых мы взвешиваем добрые и злые деяния корпорации Microsoft. Гениальная Windows XP склоняет их в сторону добра, а вот то, что они сделали с офигенной, опередившей свое время Windows Mobile 6.x, мощно тянет рычажные весы в обратную сторону. Но сегодня речь идет не об операционных системах, а о модных и популярных облачных платформах, в сфере которых корпорация тоже неплохо отметилась.

ВВЕДЕНИЕ

Раньше многие из ученых не могли и мечтать о получении доступа к мощностям суперкомпьютеров. Покупка их времени по-прежнему остается дорогим удовольствием. Однако сейчас многие компании на рынке облачных вычислений предоставляют тестовый доступ к своим ресурсам, используя которые можно за небольшую плату создать собственный кластер.

Есть множество проектов, посвященных распределенным вычислениям для решения важных задач: проектирование ускорителей элементарных частиц, поиск в космическом шуме сигналов внеземных цивилизаций... В нашем проекте мы будем решать задачу создания распределенной системы для поиска простых чисел.

Из курса математики тебе должно быть известно, что простое число — это натуральное число (то есть целое положительное), которое имеет ровно два различных натуральных делителя: единицу и само себя. К простым числам относятся: 2, 3, 5, 7, ... перечислять можно бесконечно долго. Один из самых известных алгоритмов для вычисления простых чисел — решето Эратосфена. Однако существует ряд специализированных алгоритмов, которые предназначены для определения простоты чисел специального вида. Например, тест Люка — Лемера может использоваться для поиска простых чисел Мерсенна, которые имеют следующий вид: $M_p = 2^p - 1$ (простота числа $2^p - 1$ указывает на простоту числа p). Именно данный алгоритм использует проект распределенных вычислений GIMPS (mersenne.org). В январе 2013 года с помощью этого проекта было найдено очередное число Мерсенна: M57885161, которое состоит из 17 425 170 десятичных цифр.

Американская некоммерческая правозащитная организация Electronic Frontier Foundation (EFF, Фонд электронных рубежей) обещала награду за нахождение простых чисел, состоящих более чем из 108 и 109 десятичных цифр, в размере 150 тысяч и 250 тысяч долларов (www.eff.org/awards/coop).

Описываемый проект предназначен для поиска простых чисел Мерсенна. И кто знает — может быть, именно ты сможешь найти следующее простое число и немного подзаработать.

Одна из компаний, предоставляющих тестовый доступ к своим облачным ресурсам, — Microsoft. Для тестирования Windows Azure она дает один месяц «бесплатный кредит» в размере 200 долларов. Этот кредит ты можешь использовать для покупки сервисов Azure по своему усмотрению. Например, за эти деньги можно купить три экземпляра Worker-сервиса и SQL базу данных размером 2 Гб. Для получения кредита потребуется банковская карта. Здесь: goo.gl/3XGuC описан способ использования виртуальной карты взамен реальной. Конечно, 200 долларов — это не так много, как хотелось бы, но, попросив помощи у друзей и родственников, ты вполне сможешь создать небольшую вычислительную сеть.

Для разработки проекта системы тебе понадобится Windows Azure SDK для .NET, который можно скачать по следующей ссылке: goo.gl/jCKQP.

СТРУКТУРА СИСТЕМЫ

В структуру распределенной системы вычислений входит сервер, несколько вычислительных клиентов, а также клиентское приложение, запускаемое на компьютере пользователя (управляющий клиент). Каждый из элементов системы — проект, отдельно созданный в Visual Studio.

Сервер и вычислительный клиент представляют собой облачные сервисы, которые реализованы в виде Worker-ролей. Клиент, запускаемый на компьютере пользователя, реализован в виде оконного приложения на основе технологии Windows Forms. При развертывании вычислительного клиента можно указать, что должно выполняться несколько экземпляров роли. В рамках одного облачного сервиса несколько экземпляров одной роли будут выполняться независимо, так, как если бы они выполнялись на разных компьютерах.

Задача сервера — раздавать облачным клиентам диапазоны чисел для их проверки на простоту, а также регистрировать клиенты, выполняющие вычисления. Каждый облачный клиент выполняет одинаковые действия: он запрашивает у сервера диапазон чисел, которые должен проверить на простоту, проводит проверку и результаты проверки сохраняет в базу данных. Клиент, выполняющийся на пользовательском компьютере, служит для управления процессом вычислений, а также для взаимодействия с базой данных, в которой находятся результаты проверки чисел. Рисунок поясняет потоки данных между элементами распределенной системы. Сервер, функционирующий в облаке, должен запуститься до начала работы клиентов, так как перед началом вычислений клиент должен зарегистрироваться на сервере.

Протокол работы системы следующий.

1. Запускается сервер: при запуске сервер считывает значение максимального проанализированного на простоту числа из базы данных; это значение будет использоваться далее в качестве инициализирующего; по умолчанию после запуска сервера ему запрещается раздавать данные для обработки вычислительным клиентам.
2. Запускаются и регистрируются на сервере вычислительные клиенты.
3. После запуска вычислительный клиент с помощью таймера начинает запрашивать у сервера данные для вычислений, однако эти данные не будут доступны, пока управляющий клиент не разрешит серверу раздачу данных.
4. Каждому вычислительному клиенту сервер выдает одно число (p), определяющее начало диапазона из 100 чисел, которые должен про-

верить клиент; после того как вычислительный клиент получает очередное задание, он начинает анализ чисел на простоту с помощью теста Люка — Лемера и записывает результаты анализа в базу данных.

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Сервер

Сервер описываемой системы представляет собой WCF-службу. Клиент взаимодействует со службой, запущенной на сервере, посредством конечной точки. В состав конечной точки входит:

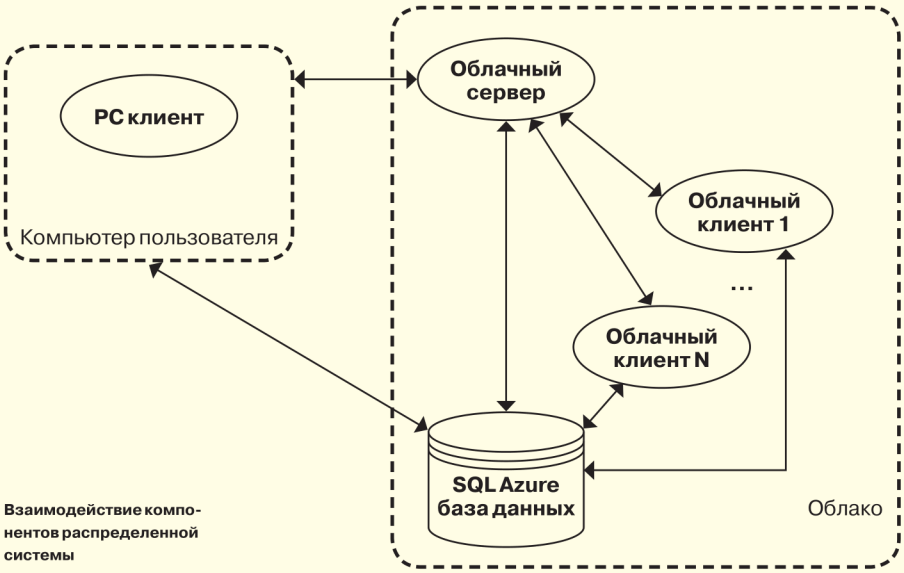
- привязка, которая задает способ связи клиента со службой; в нашем проекте служба взаимодействует с клиентами по протоколу TCP, поэтому для создания привязки используется класс NetTcpBinding;
- адрес, по которому выполняется подключение к конечной точке; для протокола TCP адрес записывается в следующем формате: net.tcp://<сервер>:<порт>/<служба>;
- контракт; учитывая возможности развития проекта, я реализовал дуплексный контракт, то есть в этом случае клиент и служба смогут обмениваться сообщениями. Дуплексный контракт определяет операции, которые может вызывать клиент на службе и служба на клиенте. Данный контракт описывается в виде двух интерфейсов: IClient (операции, которые может вызывать сервер на клиенте) и IServer (операции, которые может вызывать клиент на сервере).

Клиент для получения данных от сервера вызывает на стороне сервера метод GetData. После вызова данного метода вычислительному клиенту отправляется степень p числа Mp для его проверки на простоту с помощью теста Люка — Лемера.

Одновременно к серверу для получения вычислительного задания может обратиться несколько клиентов. Поэтому для обеспечения потокобезопасности кода используется оператор lock. В случае если два потока одновременно пытаются получить значение p, один из потоков переходит в состояние ожидания (то есть блокируется), пока другой поток исполняет критическую секцию кода.

Вот так выглядит генерация набора анализируемых чисел:

```
Int32 GetData()
{
if ((WorkerRole.control) && ↵
(WorkerRole.p >= -1))
{
    lock (WorkerRole.locker)
    {
        // Выдача начального числа p
        // диапазона чисел
        // [p; p + 100]
        if (WorkerRole.p == -1)
        {
            WorkerRole.p = 101;
            return 1;
        }
        else
        {
            Int32 tmp = WorkerRole.p;
            WorkerRole.p = ↵
            WorkerRole.p + 100;
            return tmp;
        }
    }
}
```



Взаимодействие компонентов распределенной системы

```
else
    return -1;
}
```

Код, выполняющий регистрацию клиента на сервере, размещается в методе Register(string userName, string passHash, bool isCreate). В зависимости от значения флага isCreate он выполняет различные действия:

- isCreate == true — на сервере создается учетная запись пользователя;
- isCreate == false — клиент регистрируется на сервере.

Создание учетной записи пользователя на сервере выполняет метод AddUser. При попытке регистрации проверяется наличие информации о пользователе в базе данных (эту проверку выполняет метод SearchUser). В случае если пользователь не найден в базе или хеш введенного пароля не соответствует хешу пароля в базе данных, на стороне клиента выводится сообщение об ошибке.

Учет пользователей ведется с помощью таблицы users, в которую входят три поля:

- id — уникальный идентификатор (автоинкрементное поле);
- name — имя пользователя;
- passhash — хеш пароля пользователя, сгенерированного по алгоритму SHA-512.

Вот скрипт создания таблицы для хранения информации об учетных записях:

```
CREATE TABLE [dbo].[users] (
  Id ( INT IDENTITY(1,1) NOT NULL ↵
  PRIMARY KEY,
  name ( NVARCHAR(MAX) NOT NULL,
  passhash ( NVARCHAR(MAX) NOT NULL
)
```

Давай рассмотрим классы, используемые в методах AddUser и SearchUser для работы с базой данных. Переменная cs, передаваемая в качестве параметра в конструктор класса SqlConnectionStringBuilder, представляет собой строку для подключения к SQL Azure базе данных. В данной переменной хранится адрес сервера базы данных, идентификатор и пароль пользователя для доступа к серверу, а также некоторые параметры подключения. Само же подключение создается с помощью класса SqlConnection. Экземпляр класса SqlCommand используется для хранения выполняемой SQL-команды. Данный класс предоставляет возможность подстановки значений в исходное SQL-выражение. Класс SqlDataReader предназначен для получения результатов выполнения SQL-команды.

Организуем поиск учетной записи:

```
public string SearchUser(string name) {
    // Создание строки для подключения
    // к базе users
    SqlConnectionStringBuilder ↵
    connString2Builder = new ↵
    SqlConnectionStringBuilder(cs);
    connString2Builder.InitialCatalog =↵
    DatabaseName;
    // Подключение к базе данных
    SqlConnection conn = new ↵
    SqlConnection(connString2Builder.↵
    ToString());
    SqlCommand command = null;
    SqlDataReader reader;
    conn.Open();
    // Выборка информации
    // о пользователе
    string CommandText = "select * from ↵
    users where (name = @name)";
    command = new SqlCommand↵
    (CommandText, conn);
    command.Parameters.Add("@name", ↵
    SqlDbType.NVarChar);
    command.Parameters["@name"].↵
    Value = name;
    reader = command.ExecuteReader();
    // Получение результатов выполнения
    // запроса
    string nm = "";
    string hs = "";
    while (reader.Read()) {
        hs = reader["passhash"].↵
        ToString().Trim();
        nm = reader["name"].ToString().↵
        Trim();
```

```
    }
    conn.Close();
    return hs;
}
```

И создаем учетную запись:

```
public void AddUser(string name, ↩
string passhash)
{
    // Создание строки для подключения
    // к базе users
    SqlConnectionStringBuilder ↩
    connString2Builder = new ↩
    SqlConnectionStringBuilder(cs);
    connString2Builder.InitialCatalog =↩
    DatabaseName;
    // Текст команды
    string CommandText = "insert into ↩
    users (name, passhash) values ↩
    (@name, @passhash)";
    // Подключение к базе данных
    SqlConnection conn = new ↩
    SqlConnection(connString2Builder.↩
    ToString());
    SqlCommand command = new ↩
    SqlCommand(CommandText, conn);
    command.Parameters.Add("@name", ↩
    SqlDbType.NVarChar);
    command.Parameters["@name"].Value =↩
    name;
    command.Parameters.Add("@passhash",↩
    SqlDbType.NVarChar);
    command.Parameters["@passhash"].↩
    Value = passhash;
    conn.Open();
    // Добавление учетной записи
    int rowsAdded = command.↩
    ExecuteNonQuery();
    conn.Close();
}
```

Вычислительный клиент

Основная (и единственная) задача вычислительного клиента — проверять, является ли число Mp простым. Проверить простоту числа Mp = 2p – 1 можно с помощью такой последовательности действий (тест Люка — Лемера):

1. Задать число $L_0 = 4$.
2. В цикле вычислить значение $L_{i+1} = (L_i^2 - 2) \bmod (M_p)$.

Таким образом, число Mp является простым, если остаток от деления числа Lp – 2 на Mp будет равен нулю.

Давай рассмотрим пример определения простоты числа Mp = 31 (p = 5, Mp = $2^5 - 1 = 31$):

1. $L_0 = 4$.
2. $L_1 = (4^2 - 2) \bmod 31 = 14$.
3. $L_2 = (14^2 - 2) \bmod 31 = 8$.
4. $L_3 = (8^2 - 1) \bmod 31 = 0$.

Тест пройден успешно, следовательно, число 31 простое.

Тест Люка — Лемера:

```
bool isPrime(Int32 n) {
    // Проверка на четность
    if (n % 2 == 0) return (n == 2);
    else
    {
        for (int i = 3; i <= ↩
        (int)Math.Sqrt(n); i += 2)
            // Число не простое
            if (n % i == 0)
                return false;
        // Выполнение теста
        BigInteger M_p = BigInteger.↩
        Pow(2, n) - 1;
        BigInteger L = 4;
        for (int i = 3; i <= n; i++)
            L = (L * L - 2) % M_p;
        return L == 0;
    }
}
```

В тесте Люка — Лемера двойка возводится в некоторую степень. Результат возведения может быть достаточно большим числом. Большим настолько, что для его хранения не хватит типа decimal. Поэтому в клиентском приложении для работы с большими целыми числами используется класс BigInteger, который находится в пространстве имен System.Numerics (библиотека System.Numerics.dll). Переменная типа BigInteger может содержать любое целое значение.

Хранение информации о результатах проверки чисел выполняется с помощью таблицы numdata, в которую входят следующие поля:

- id — уникальный идентификатор (автоинкрементное поле);
- num — проверяемое число;
- isprime — флаг, содержащий результат проверки (простое / не простое);
- date — время записи информации в таблицу;
- client — информация о клиенте, выполнившем проверку.

Вот скрипт для создания таблицы, хранящей результаты вычислений:

```
CREATE TABLE [dbo].[numdata] (
  Id (      INT  IDENTITY(1,1) NOT NULL,
  [num]     BIGINT             NOT NULL,
  [isprime] INT                NOT NULL,
  [date]    DATETIME           NOT NULL,
  client (  VARCHAR (MAX)      NOT NULL,
  PRIMARY KEY CLUSTERED (Id ( ASC)
);
```

Вставку в таблицу numdata информации о результатах вычислений выполняет метод InsertRow(Int32 num, bool isprime). Логика работы данного метода подобна логике метода AddUser.

Управляющий клиент

Управляющий клиент контролирует отправку данных от сервера клиентам с помощью вызова метода Control. Этот метод позволяет запретить или разрешить раздачу вычислительным клиентам данных для обработки.

Если выполняется попытка запуска раздачи данных (start == true), тогда сервер выбирает из базы данных максимальное вычисленное значение числа (с помощью метода SelectId) и использует это значение в качестве начального для дальнейших вычислений.

Если выполняется попытка останова раздачи данных сервером (start == false), тогда значение флага control устанавливается в false (данный флаг служит индикатором работы сервера).

Метод для управления сервером выглядит так:

```
public void Control(bool start)
{
    // true
    if (start)
    {
        // Обновляем значение p
        WorkerRole.k = WorkerRole.↩
        SelectId();
        if (WorkerRole.p >= -1)
            WorkerRole.control = start;
        else
            WorkerRole.control = false;
    }
    else
        WorkerRole.control = start;
}
```

Создание учетной записи клиента и регистрацию клиента на сервере выполняет метод CreateOrRegister(bool). Все зависит от значения передаваемого ему параметра: true — создание учетной записи, false — вход клиента на сервер. Этот метод вызывает метод Register на стороне сервера.

Разработанный проект можно развернуть в облаке или локальном эмуляторе Azure. Развертывание сервиса в облаке выполняется после выбора пункта Publish в контекстном меню созданной Worker-роли. Запуск проекта в локальном эмуляторе — по нажатию на <F5> в среде разработки.

Локальный эмулятор предоставляет возможность одновременного запуска и сервера, и вычислительного клиента. Для отладки в локальном эмуляторе вычислительного клиента нужно в файле WorkerRole.cs, входящем в проект клиента, изменить адрес сервера (переменная endPoint), а в интерфейсе управляющего клиента указать следующий адрес сервера: net.tcp://localhost:3030/CloudService.

ЗАКЛЮЧЕНИЕ

Проект, описанный в статье, конечно же, можно использовать не только для вычисления простых чисел. Он может найти свое применение в распределенной обработке звука, изображений, видео.

Возможности совершенствования проекта можно связать с полноценной реализацией дуплексного режима обмена данными. В таком случае клиент не должен будет постоянно опрашивать сервер, а запросит данные после того, как получит от сервера сообщение об их доступности.

ПУБЛИКАЦИЯ ПРОЕКТА

Настройки для публикации проекта ты можешь получить по ссылке goo.gl/JVZHM. Примечание: сначала зайди в свой аккаунт Azure.

САЙТЫ ПО РАСПРЕДЕЛЕННЫМ ВЫЧИСЛЕНИЯМ

Получить информацию и присоединиться к проектам распределенных вычислений ты сможешь на сайтах https://distributed.ru, www.boinc.ru.