Санкт-Петербургский Государственный Университет

Математико-Механический факультет

Кафедра системного программирования

Автоматическое масштабирование в облаке Windows Azure

Курсовая работа студента 361 группы

Кавокина Александра Сергеевича

Научный руководитель Сартасов С.Ю.

Санкт-Петербург

2013 год

Оглавление

Введение....2
1. Постановка задачи....3
2. Реализация решения....4
2.1. Инструменты....4
2.2. Правила реагирования и Performance Counters....4
3. Результаты....7
Список используемых источников....8
Приложение 1. Скрипт создания счетчиков....9
Приложение 2. Правила автомасштабирования....10

Введение

В последние годы в мире все большую популярность набирают облачные технологии. Облачный сервис предоставляет возможность разработки и выполнения приложений и хранения данных на серверах, расположенных в распределенных датацентрах. Клиент оплачивает только ресурсы и мощности, которые задействованы в приложении, и только фактическое время использования этих ресурсов. В основе работы Windows Azure лежит запуск виртуальной машины для каждого экземпляра приложения или его подсистемы. Разработчик определяет необходимый объем для хранения данных и требуемые вычислительные мощности (количество виртуальных машин), после чего платформа предоставляет соответствующие ресурсы. Когда первоначальные потребности в ресурсах изменяются, в соответствии с новым запросом заказчика платформа выделяет под приложение дополнительные или освобождает неиспользуемые ресурсы [1].

В интересах разработчиков минимизировать свои затраты. Неразумное использование ресурсов Windows Azure приведет к значительным финансовым потерям. Поэтому при написании программного продукта следует учитывать, когда будет необходимо подключать новые ресурсы, а когда от части из них можно избавиться. Можно следить за нагрузкой вручную. Но такое решение достаточно дорогое, и возможны ошибки в виду человеческого фактора. Поэтому хотелось бы иметь автоматизированный комплекс по выполнению масштабирования в облаке.

1. Постановка задачи

Для нормального функционирования сервиса требуется, чтобы его мощности всегда могли справляться с нагрузкой на него. Но моментально среагировать на скачок нагрузки невозможно, виртуальная машина подключается некоторое время (до часа). Если сервер будет тормозить в течение часа, он может потерять популярность. Поэтому надо научить приложение предугадывать пики нагрузки и подключать новые ресурсы заранее. Причем ошибки в предугадывании также могут сильно сказаться на доходах, поскольку содержание вычислительных ресурсов стоит денег и не хотелось бы, чтобы они работали вхолостую. Также существует «реактивный» способ реагирования [5]. В этом случае новые машины подключаются, если нагрузка превысила некоторое пороговое значение, и отключаются, если стала ниже. Но в таком случае следует иметь в виду время отклика сервиса, время на подключение новой роли. Помимо этого, требуется ввести стабилизацию, чтобы количество ролей не менялось постоянно из-за «дребезга» возле пороговых значений.

В настоящее время развивается студенческий проект WarSpot, в рамках которого используется Windows Azure. Пользователи отправляют запросы в очередь. Роли берут из этой очереди запросы и обрабатывают их. Если сообщений слишком много (или мало), и роли не справляются (простаивают без дела), то срабатывают правила автомасштабирования и количество ролей корректируется. Требовалось найти пороговые значения для автомасштабирования.

Также стало известно, что за неполный час работы счет выставляется как за полный календарный час. Например, если роль была запущена в 10:50, а остановлена в 11:10, то счет будет выставлен за 2 часа – один час за использование в период с 10:50 до 11:00, а второй час за использование в период с 11:00 до 11:10, хотя реально роль отработала 20 минут. То есть в данном случае роль можно было и не отключать сразу, а придержать работающей еще 50 минут. Следовательно, при реализации нужно еще учитывать и этот фактор. И, как уже говорилось, требовалось стабилизировать полученные правила [6].

Таким образом, была поставлена задача:

1) найти пороговые значения, связанные с количеством сообщений в очереди, при которых требуется подключать или отключать новые ресурсы, и стабилизировать изменение количества ролей;
2) реализовать в рамках проекта автомасштабирование, учитывая вычисленные описанные выше подробности.

2. Реализация решения

2.1. Инструменты

В качестве средств для разработки был выбран C#, поскольку проект, в рамках которого была поставлена данная задача, реализовывается в среде .NET.

Существует блок автоматического масштабирования Wasabi, являющийся частью пакета интеграции Enterprise Library для Windows Azure. Данный блок позволяет определить, как приложение Windows Azure будет автоматически обрабатывать изменения уровня нагрузки. Он помогает свести к минимуму эксплуатационные расходы, в то же время, обеспечивая высокую производительность и доступность приложения для пользователей. Он также помогает уменьшить количество задач, выполняемых операторами вручную [6].

Программный блок работает на основе набора пользовательских правил, которые управляют поведением приложения при изменении нагрузки. Правила бывают двух типов: *правила ограничений*, задающие минимальное и максимальное количество экземпляров роли в приложении Windows Azure, и *правила реагирования*, которые изменяют текущее количество экземпляров роли на основе счетчиков или показателей, полученных из приложения. Правила хранятся в XML-файле [4].

Для подсчета нагрузки в правилах используются Performance Counters. Чтобы создать пользовательские счетчики в реестре, требуется запускать другое приложение с правами на это, поскольку исполняемая среда на Windows Azure не имеет прав для записи в него. С этой целью использовался PowerShell [2].

2.2. Правила реагирования и Performance Counters

При реализации стало понятно, что невозможно найти константы, удовлетворяющие требуемым условиям, поскольку при изменении количества ролей они постоянно изменяются. Следовательно, требуется искать функциональную зависимость от количества ролей, в частности, количество сообщений на роль.

Как уже говорилось ранее, Windows Azure выставляет счет по часам. Таким образом, пришлось учитывать временные ограничения. Но также необходимло учитывать, что на сильные всплески нельзя не реагировать, поскольку сервер может отказать при перегрузке.

При создании правил автомасштабирования (см. приложение 2) возникла проблема. На локальном облаке невозможно использовать правила, поскольку локальный Computer

Emulator не поддерживает Windows Azure Service Management API. Исходя из этого, изначальную задачу пришлось разбить на несколько подзадач.

В процессе работы был создан DiagnosticMonitorConfiguration, с помощью которого добавлялись счетчики в блок автомасштабирования. Эти счетчики были созданы при помощи скрипта на PowerShell (см. приложение 1).

В реальном облаке процесс создания счетчиков запускается перед стартом роли аналогичным образом [3].

Обновление счетчиков выполняется в отдельном потоке в отдельном классе PerformanceCounterManager. После этого результаты обновления счетчиков планировалось выводить в таблицу WADPerformanceCountersTable, но в локальном облаке это невозможно, поскольку счетчики сохраняются в реестр и локализуются там под версию OS Windows. Таким образом, в таблицу добавлялись не обновленные значения счетчиков (рис. 1).

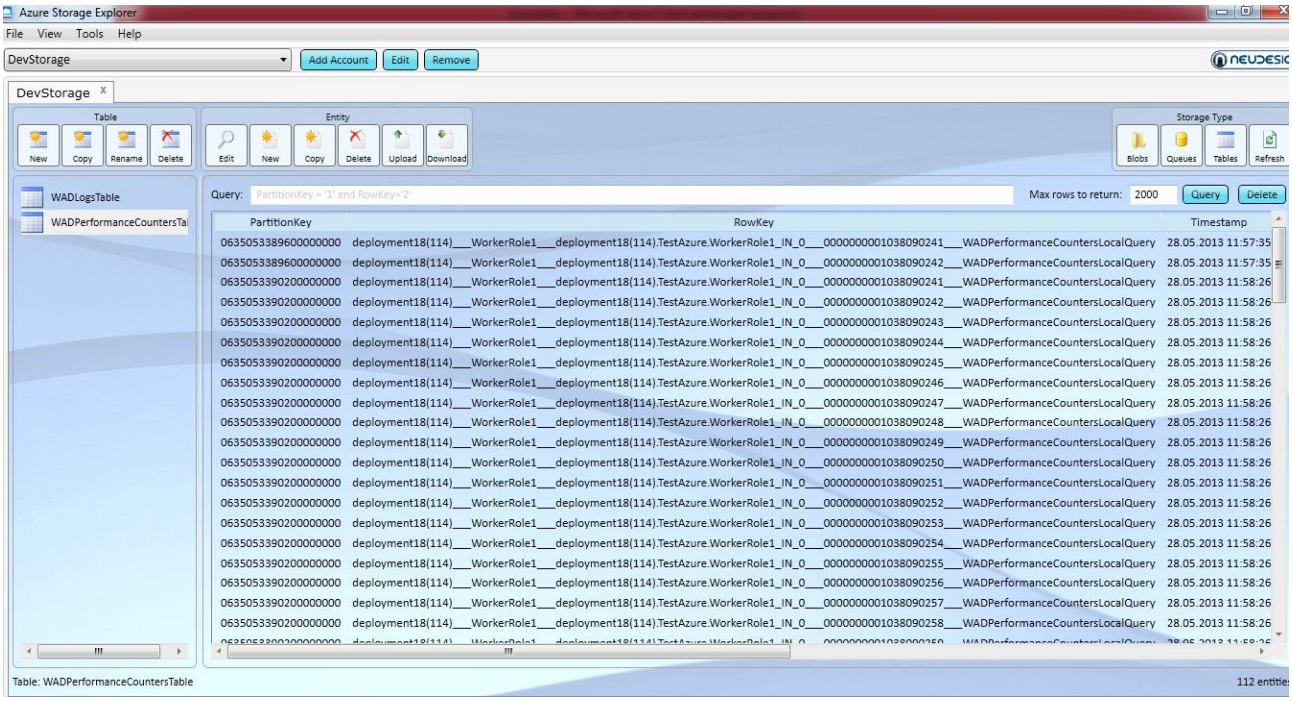

Рис. 1. Значения счетчиков в Azure Storage Emulator

Чтобы убедиться в том, что счетчики правильно обновляются, была использована Консоль управления Microsoft (MMC) (рис. 2).

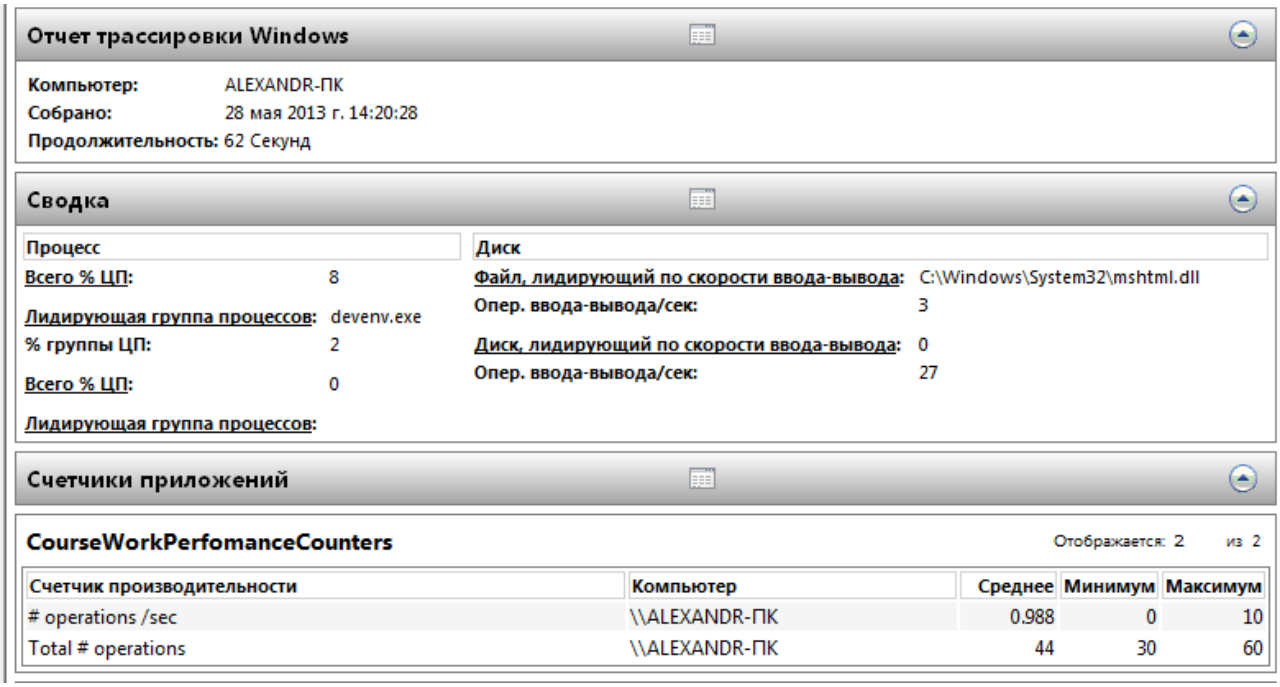

Рис. 2. Результаты обновления в MMC

3. Результаты

В результате проделанной работы были получены следующие результаты:

1) найдена функциональная зависимость пороговых значений для автомасштабирования от количества ролей;
2) написаны правила автомасштабирования;
3) реализован функционал по внедрению пользовательских счетчиков: их создание, обновление и считывание.

Таким образом, полученные результаты можно использовать в дальнейшем в данной предметной области.

Список используемых источников

1) MSDN Microsoft. Windows Azure, 2013
   http://msdn.microsoft.com/en-us/library/windowsazure/dd163896.aspx
2) MSDN Microsoft. Real World: Creating Custom Performance Counters for Windows Azure Applications with PowerShell, 2011
   http://msdn.microsoft.com/en-us/library/windowsazure/dd163896.aspx
3) Windows Azure Microsoft, Using performance counters in Windows Azure, 2013
   http://www.windowsazure.com/en-us/develop/net/common-tasks/performance-profiling/?fb=en-en
4) Windows Azure Microsoft, How to Use the Autoscaling Application Block, 2013
   http://www.windowsazure.com/en-us/develop/net/how-to-guides/autoscaling/?fb=ru-ru#header-0
5) MSDN Microsoft, Автоматическое масштабирование и Windows Azure, 2013
   http://msdn.microsoft.com/ru-ru/library/hh680945(v=pandp.50).aspx
6) MSDN Microsoft, Создание более эластичного приложения TailSplin Surveys, 2013
   http://msdn.microsoft.com/ru-ru/library/hh680942(v=pandp.50).aspx

Приложение 1. Скрипт создания счетчиков

```
$categoryName = "Мои счетчики"

$exists = [System.Diagnostics.PerformanceCounterCategory]::Exists($categoryName)
if ($exists)
{
[System.Diagnostics.PerformanceCounterCategory]::Delete($categoryName)
}

$counterData = new-object System.Diagnostics.CounterCreationDataCollection

$counter = new-object  System.Diagnostics.CounterCreationData
$counter.CounterType = [System.Diagnostics.PerformanceCounterType]::RateOfCountsPerSecond32
$counter.CounterName = "# операций в секунду"
$counterData.Add($counter)

$counter = new-object  System.Diagnostics.CounterCreationData
$counter.CounterType = [System.Diagnostics.PerformanceCounterType]::NumberOfItems32
$counter.CounterName = "Итого операций"
$counterData.Add($counter)

[System.Diagnostics.PerformanceCounterCategory]::Create($categoryName, $categoryName,
[System.Diagnostics.PerformanceCounterCategoryType]::SingleInstance, $counterData)
```

Приложение 2. Правила автомасштабирования

```
<?xml version="1.0" encoding="utf-8" ?>
<rules xmlns="http://schemas.microsoft.com/practices/2011/entlib/autoscaling/rules">
  <constraintRules>
    <rule name="Default" description="Always active"
          enabled="true" rank="1">
      <actions>
        <range min="1" max="10" target="WorkerRole"/>
      </actions>
    </rule>
  </constraintRules>
  <reactiveRules>
    <rule name="PeekUp"
          description=""
          enabled="true">
      <actions>
        <scale target="WorkerRole" by="1" />
      </actions>
      <when>
        <greater operand="CountOfMsgsForRoleInPeekTime" than="20" />
      </when>
    </rule>
    <rule name="PeekDown"
          description=""
          enabled="true">
      <actions>
        <scale target="WorkerRole" by="-1" />
      </actions>
      <when>
        <lessOrEqual operand="CountOfMsgsForRoleInPeekTime" than="10" />
      </when>
    </rule>
  </reactiveRules>
  <operands>
    <performanceCounter alias="CountOfMsgsForRoleInPeekTime"
       timespan="00:01:00" aggregate="Average" source="WorkerRole"
       performanceCounterName="\PeekCounter" />
    <performanceCounter alias="CountOfMsgsForRoleInUsualTime"
       timespan="00:10:00" aggregate="Average" source ="WorkerRole"
       performanceCounterName="\UsualCounter" />
  </operands>
</rules>
```