АРХИТЕКТУРНЫЕ АСПЕКТЫ ИСПОЛЬЗОВАНИЯ ОБЛАЧНОЙ ПЛАТФОРМЫ MICROSOFT WINDOWS AZURE PLATFORM ДЛЯ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ

О.А. Авдиенков, В.Н. Маланин

Введение

В настоящее время во многих областях требуются высокопроизводительные компьютерные комплексы, но покупка и обслуживание соответствующего оборудования требует больших финансовых затрат. В последнее время среди суперкомпьютеров доминирует кластерная архитектура (более 80% из top500 - это кластерные решения), занявшая лидирующие позиции в основном благодаря более низкой по сравнению с традиционными параллельными архитектурами стоимости единицы производительности. Но, все равно, далеко не каждая организация, которая нуждается в высокопроизводительных вычислениях, может позволить себе приобрести кластер, способный обеспечить необходимый для неё уровень производительности.

Обратной стороной проблемы является то, что современные высокопроизводительные кластеры, приобретаемые с запасом масштабируемости для решения самых больших задач организации, обычно большую часть времени простаивают либо загружены незначительно.

Альтернативным путем получения больших вычислительных мощностей является использование чужих компьютеров по принципу аренды, например, использование мощностей высокопроизводительного кластера, расположенного в центре обработки данных (ЦОД) провайдера подобных услуг. В данном случае количество вычислительных ресурсов, которые могут быть использованы для решения задачи, будет ограничено размером ЦОД и его загрузкой по решению задач других заказчиков.

В качестве отдельного примера использования «заёмных» мощностей можно привести ГРИД-вычисления, позволяющие задействовать для решения задач большое количество территориально распределенных вычислительных ресурсов. Для таких систем основным узким местом для решения большого класса задач является низкая пропускная способность каналов связи, объединяющих различные группы вычислительных ресурсов, задействованных в выполнении распределенной программы.

В последнее время набирают популярность услуги по предоставлению вычислительных мощностей в «облаках». Облачные (Cloud) вычисления[1] – это технология обработки данных, подразумевающая предоставление вычислительных ресурсов пользователю как интернет-сервиса. При этом компьютерные системы, обеспечивающие данный сервис, локализованы в одном или нескольких ЦОД компании-провайдера платформы для облачных вычислений, объединенных высокоскоростными каналами связи. Важными требованиями для построения "облака" являются высокая степень автоматизации процесса выделения вычислительных ресурсов по запросу пользователя, обеспечение постоянного запаса вычислительных мощностей для удовлетворения возрастающих нужд заказчиков и высокого уровня доступности используемых систем.

С точки зрения пользователя основными отличиями данного класса предложений от традиционных услуг по аренде серверов или кластеров являются:

- возможность оперативного масштабирования пула вычислительных ресурсов, занятых в решении задачи;
- возможность учета и оплаты только тех ресурсов, которые фактически были использованы.

В настоящее время набирает популярность облачное решение от компании Microsoft - Windows Azure Platform (или коротко Azure), запущенное в коммерческую эксплуатацию в начале 2010 года. Windows Azure Platform – это облако класса «платформа как сервис» (Platform-as-a-Services, PaaS)[2], основанное на разработанной Microsoft модели автоматизированного управления компьютерными ресурсами и предоставления их по требованию. На данный момент для разработчиков программного обеспечения доступно описание архитектуры программ для Azure, а также программный интерфейс (API) и инструментарий для их разработки.

Целью данной статьи является анализ возможности применения модели Azure для организации параллельных вычислений, а также рассмотрение архитектуры системы поддержки параллельных вычислений на этой платформе на примере системы Граф-Схемного Параллельного Программирования (ГСПП) [3]. Выбор рассматриваемой системы программирования объясняется тем, что модель ГСПП хорошо подходит для создания распределенных программ с динамически меняющимся количеством процессов, что позволит полностью раскрыть потенциал платформы для реализации параллелизма, а также опытом авторов в реализации системы ГСПП для кластерных систем.

Архитектура программ для платформы Azure

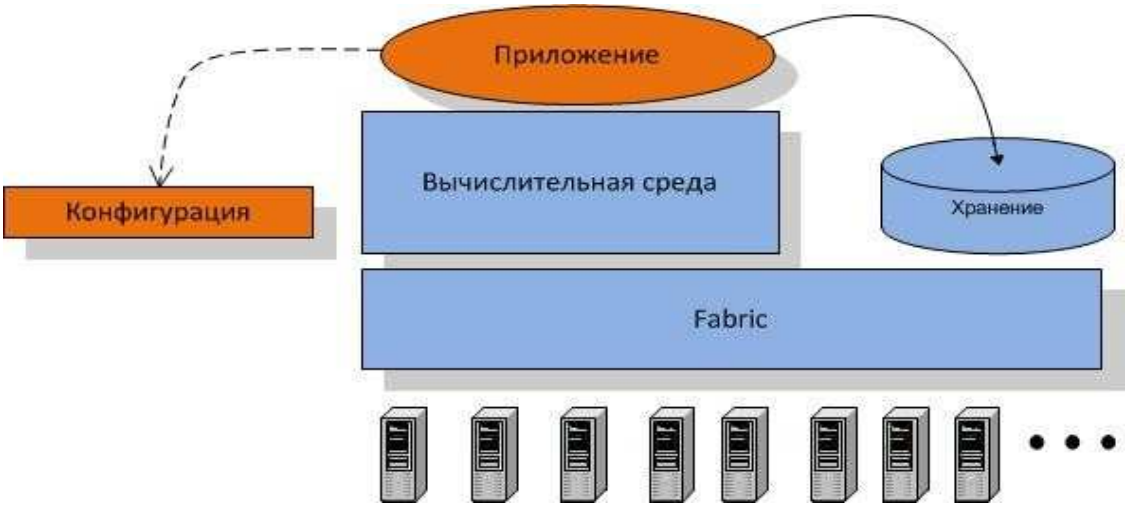

Рис.1. Платформа Azure

Концептуально платформу Azure[4] можно представить как пул вычислительных ресурсов (процессоры, оперативная память и.т.д.), в рамках которого система управления Azure (Fabric в терминах Azure) по запросу создает так называемые роли. Каждой роли может быть выделено до 8 логических процессоров и до 15 гигабайтов оперативной памяти для выполнения соответствующего роли программного кода. Технически при создании новой роли создается отдельная виртуальная машина с операционной системой (ОС) Windows Server, в среде которой и выполняется пользовательский программный код в виде одного или нескольких процессов ОС. Каждая вирутальная машина, содержит также агент Azure, позволяющий пользовательскому программному коду взаимодействовать с системой управления Fabric. Агент предоставляет программный интерфейс (API), который включет функции ведения журнала событий, отправки различных уведомлений владельцу роли и многое другое. Роли изолированы друг от друга, а для коммуникации могут использовать разделяемое хранилище данных Windows Azure.

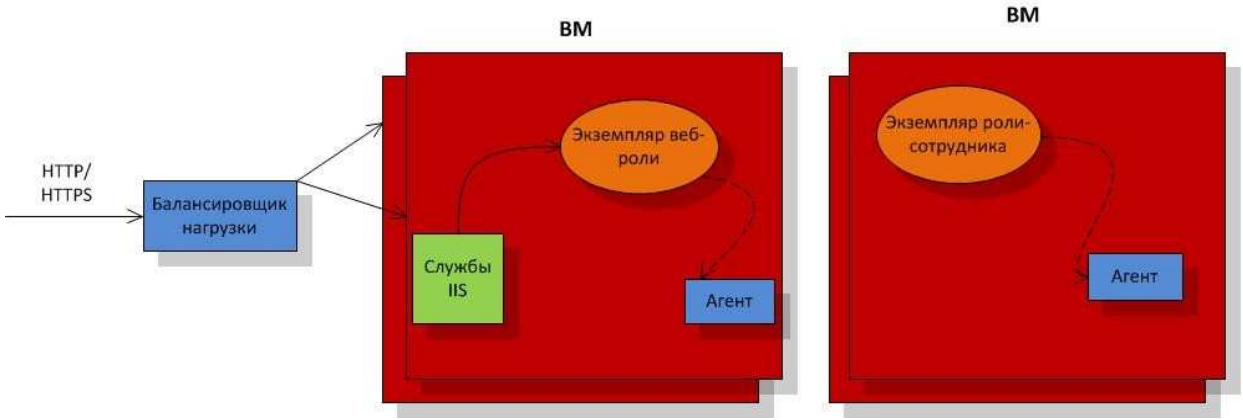

Рис.2. Роли Windows Azure

На схеме (рис.2) видно, что программа для Azure может содержать два типа ролей: веб-роль (Web role) и роль-сотрудник (Worker role). Веб-роль предназначена для получения и обработки входящих запросов по протоколу HTTP или HTTPS через службы веб-сервера Internet Information Services(IIS). Роль-сотрудник не подразумевает взаимодействие с внешними по отношению к Azure системами, и обычно получает данные из хранилища Windows Azure.

Количество ролей и соответствующие им пользовательские программные коды задаются при запуске в конфигурационном файле распределенной программы. С точки зрения архитектуры программа может содержать любое количество экземпляров веб-ролей и ролей-сотрудников. Для масштабирования приложения его владелец должен увеличить в файле конфигурации количество экземпляров нужных ролей, после чего Azure Fabric запустит дополнительные экземляры ролей на новых виртуальных машинах. Увеличение количества экземпляров ролей можно производить во время работы программы, для этого реализован интерфейс управления (Management API).

Хранилище Azure может содержать три типа объектов: Blob (большие бинарные объекты), таблицы и очереди. Для доступа к данным, находящимся в хранилище, нужно создать HTTP запрос по соответствующему адресу, в запрос также включаются ключи доступа для контроля входа.

Пример архитектуры приложения для Azure

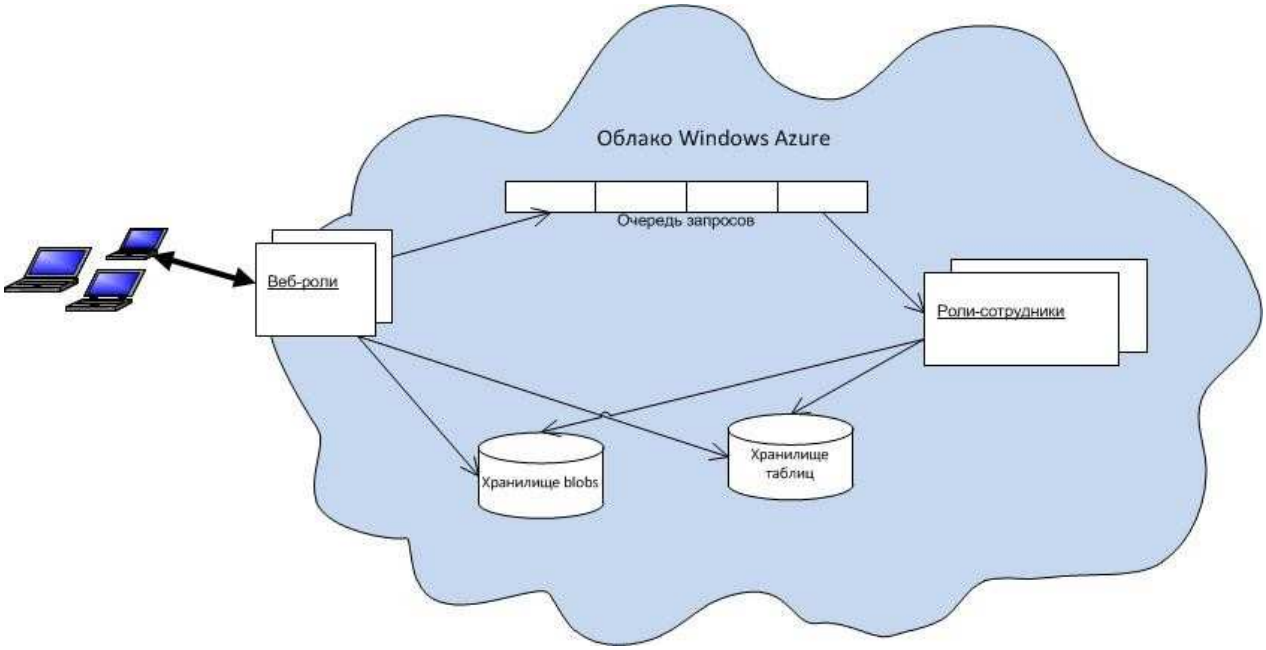

Рис.3. Пример архитектуры приложения для Azure.

Рисунок 3 иллюстрирует вид типичной архитектуры приложения в облаке Azure. Имеется несколько веб-ролей, которые реализуют интерфейсную логику обработки запросов от пользователя. Также существует набор ролей-сотрудников, реализующих бизнес-логику приложения. Веб-роли обмениваются информацией с ролями-сотрудниками посредством наборов запросов, размещаемых в очередях. Результаты работы бизнес-логики могут сохраняться в хранилищах типа Blob и таблицах.

Рассмотрим в качестве примера приложение для хранения фотографий, с помощью которого пользователи могут загружать свои фотографии через веб-интерфейс и сохранять для последующего просмотра. При загрузке фотографий приложение должно автоматически создавать уменьшенный вариант изображений для реализации функции предварительного просмотра и вместе с оригиналами размещать в каталогизированном хранилище.

Такое приложение может использовать представленную выше архитектуру. Веб-интерфейс для загрузки и последующего просмотра фотографий обеспечивается одной или несколькими веб-ролями. Файлы изображений хранятся как большие двоичные объекты в хранилище Blob. Приложение обслуживает ряд таблиц для учета имеющихся фотографий и хранения индексов, используемых для поиска. Роли-сотрудники выполняют изменение размеров поступающих на вход фотографий, сохранение их в хранилище Blob и отвечают за обновление таблиц. При получении от пользователя запроса на загрузку новой фотографии веб-роли создают рабочий элемент и помещают его в очередь запросов. Роли-сотрудники извлекают эти рабочие элементы из очереди и обрабатывают соответствующим образом. В случае успешной обработки роль-сотрудник должна удалить рабочий элемент из очереди во избежание его повторной обработки другой ролью-сотрудником.

Средства Azure для создания параллельного кластера

Рассмотрим, каким образом с помощью платформы Azure можно создать «облачный кластер» для выполнения параллельных программ.

На роль вычислительных узлов такого кластера хорошо подходят роли-сотрудники Azure. Данный выбор позволяет изменять размер кластера, когда это необходимо.

Для передачи данных между узлами кластера лучше всего использовать очереди хранилища Azure, позволяющие реализовать хорошо известный механизм producer/consumer. А именно, когда в очередь помещается задание на вычисление, свободный узел получает данные из очереди на вычисление, а результат своих действий помещает либо в другую очередь, либо в хранилище. Причем очередь в Azure устроена так, что если роль-сотрудник извлекает из нее задание на выполнение, то сообщение в очереди становится невидимым на заданный промежуток времени. Если во время вычислений в роли-сотруднике происходит сбой, данное сообщение восстанавливается в очереди и будет обработано другой ролью-сотрудником.

Важно также, что очереди позволяют следить за тем, насколько эффективно роли-сотрудники справляются с вычислительной нагрузкой. Увеличение длины очереди свидетельствует о том, что роли-сотрудники не справляются с нагрузкой и, следовательно, требуется увеличить количество экземпляров ролей-сотрудников в конфигурационном файле. Если очередь большую часть времени не содержит заданий, стоит уменьшить количество ролей-сотрудников, так как часть из них простаивает.

Рассмотренные инструменты являются базовыми для построения параллельной программы на платформе Azure. Однако для упрощения процесса разработки параллельных программ необходимы

высокоуровневые средства, позволяющие просто проектировать, реализовывать и выполнять запуски сложных программ на данной платформе. В качестве примера архитектуры системы, поддерживающей высокоуровневое параллельное программирование, приведем разработанную среду выполнения граф-схемных параллельных программ для платформы Azure.

Язык граф-схемного параллельного потокового программирования

Язык граф-схемного параллельного потокового программирования (ЯГСПП) ориентирован на крупноблочное (модульное) потоковое программирование задач. Он также может эффективно применяться для моделирования распределенных систем, систем массового обслуживания и др., то есть систем, информационные связи между компонентами которых структурированы и управляются потоками данных, передаваемых по этим связям.

Язык позволяет эффективно и единообразно представлять в программах три вида параллелизма:

- параллелизм информационно-независимых фрагментов;
- потоковый параллелизм, обязанный своим происхождением конвейерному принципу обработки данных;
- параллелизм множества данных.

Полезными с позиции программирования особенностями ЯГСПП являются возможности визуального графического представления программы и простого структурирования, отражения декомпозиционной иерархии при ее построении путем использования отношения «схема-подсхема».

Граф-схемная параллельная программа (ГСПП) представляется в виде пары <ГС,I>, где ГС – граф-схема, I – интерпретация.

Граф-схема или просто схема позволяет визуально представлять строящуюся из отдельных компонентов (модулей) программу решения задачи. Интерпретация сопоставляет каждому модулю множество подпрограмм, а связям между модулями – типы данных, передаваемых между подпрограммами модулей в процессе выполнения ГСПП.

Основным «строительным» блоком ГСПП является модуль, графическое представление которого приведено на рисунке (рис.3) ниже. Все входы и выходы модулей строго типизированы.

Язык граф-схемного параллельного потокового программирования

Язык граф-схемного параллельного потокового программирования (ЯГСПП) ориентирован на крупноблочное (модульное) потоковое программирование задач. Он также может эффективно применяться для моделирования распределенных систем, систем массового обслуживания и др., то есть, систем, информационные связи между компонентами которых структурированы и управляются потоками данных, передаваемых по этим связям.

Язык позволяет эффективно и единообразно представлять в программах три вида параллелизма:

- параллелизм информационно-независимых фрагментов;
- потоковый параллелизм, обязанный своим происхождением конвейерному принципу обработки данных;
- параллелизм множества данных.

Полезными с позиции программирования, особенностями ЯГСПП являются возможности визуального графического представления программы и простого структурирования, отражения декомпозиционной иерархии при ее построении путем использования отношения «схема-подсхема».

Граф-схемная параллельная программа (ГСПП) представляется в виде пары <ГС,I>, где ГС – граф-схема, I – интерпретация.

Граф-схема или просто схема позволяет визуально представлять строящуюся из отдельных компонентов (модулей) программу решения задачи. Интерпретация сопоставляет каждому модулю множество подпрограмм, а связям между модулями – типы данных, передаваемых между подпрограммами модулей в процессе выполнения ГСПП.

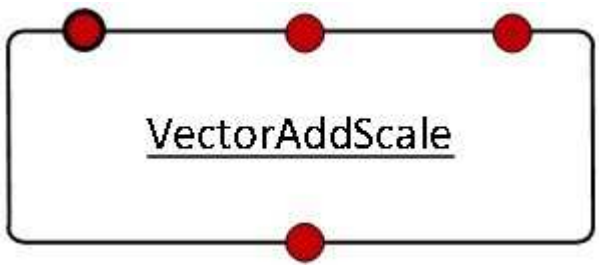

Рис.4. Модуль ГСПП.

Основным «строительным» блоком ГСПП является модуль, графическое представление которого приведено на рисунке ниже (рис.4). Все входы и выходы модулей строго типизированы.

ГС представляет блочную структуру, построенную из модулей путем соединения выходов модуля с входами другого или этого же модуля, причем типы соединяемых входов и выходов должны совпадать. В интерпретации такое соединение называется информационной связью (ИС). По ней данные соответствующего типа передаются от одного модуля к другому.

Возможны связи «один ко многим» и «многие к одному» (рис. 5)

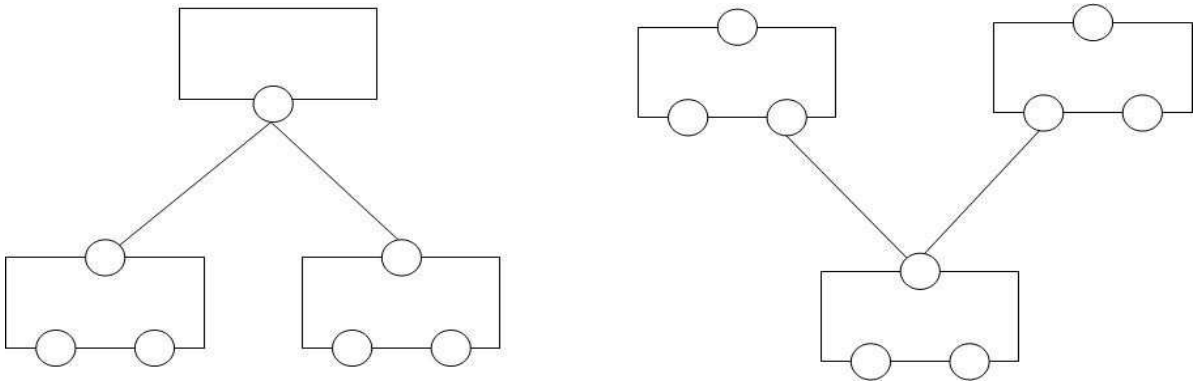

Рис.5 Виды связей

Первое означает, что данные передаются одновременно с выхода модуля на все входы, с которыми он связан, а второе – что данные на вход могут поступать с любого из выходов, с которыми он связан.

Различаются межмодульные и свободные (установочные) связи. Последние позволяют перед выполнением ГСПП задать на соответствующих входах модулей начальные значения. Допускается, что у модуля может не быть входов. В этом случае подпрограмма, сопоставленная этому модулю, является подпрограммой без параметров и она начинает выполняться вместе с началом выполнения ГСПП.

Важным элементом при построении ГСПП является понятие подсхемы, которая даёт программисту эффективные средства структуризации параллельного алгоритма. Такими средствами является возможность представления ГС в виде нескольких уровней, отражающих шаги декомпозиции исходной задачи на подзадачи при построении параллельной программы. Графически подсхема строится аналогично схеме и содержательно представляет самостоятельный фрагмент ГСПП, подставляемый вместо одного или нескольких модулей подсхемы более высокого уровня в иерархии «схема – подсхема».

Операционная семантика ГСПП

Выполнение ГСПП описывается последовательностью состояний, каждое из которых есть множество процессов, индуцируемых при запуске подпрограмм модулей.

При выполнении ГСПП каждой ИС однозначно сопоставляется буфер, в котором хранятся асинхронно поступающие на входы модуля данные (фактические параметры соответствующих подпрограмм). Если на всех входах какой-либо модуля есть данные, помеченные одним и тем же тегом, то индуцируется новый процесс, предполагающий применение подпрограммы, сопоставленной модулю к собранным данным с указанным тегом.

Те модули, которые не имеют входов (модулям сопоставляются подпрограммы с пустым множеством параметров), индуцируют соответствующий процесс только один раз – при инициализации выполнения ГСПП.

При выполнении процесса в его подпрограмме могут использоваться специальные системные команды для обеспечения межмодульного взаимодействия: WRITE (запись во вход); READ (чтение из выхода).

Параметрами команды WRITE являются список выходов, кортеж данных, записываемых в поток и сопоставленный им тег. Поток данных определен заданием ИС, исходящих из выходов. В конечном итоге выполнение данной команды приводит к появлению заданных данных на всех входах, связанных ИС с указанными в параметрах команды выходами.

Параметрами команды READ являются имя список входов, и тег, сопоставленный читаемым из потока данным. Команда READ позволяет процессу читать данные с указанным тегом из входа (сопоставленных им при реализации буферов) модуля, которому принадлежит выполняющаяся подпрограмма. При выполнении команды READ, если запрашиваемые данные еще не поступили в буферную память, ее выполнение задерживается, до тех пор, пока эти данные не поступят. После выполнения команды READ запрашиваемые данные стираются из буферной памяти.

Для более “тонкой” работы с поступающими на вход модулей данными предусмотрена команда CHECK. Эта команда проверяет наличие данных с указанными тегами на указанных входах. Эта команда позволяет процессу самому принимать решение о дальнейших действиях в зависимости от наличия или отсутствия поступающих данных. Результатом ее выполнения является присваивание переменной общего количества данных с указанным тегом на входах, номера которых перечислены в списке входов.

Архитектура среды выполнения программы ГСПП на платформе Azure

Система ГСПП предназначена для поддержки процесса проектирования, реализации и выполнения программ на ЯГСПП. Среда состоит из двух частей: инструментальной среды, отвечающей за взаимодействие с пользователем на всех этапах жизненного цикла программы; среды выполнения ГСПП на вычислительных системах. Данная часть системы может быть реализована по-разному в зависимости от специфики целевой вычислительной системы. В рамках данной статьи опишем архитектурные принципы реализации среды выполнения ГСПП на платформе Azure.

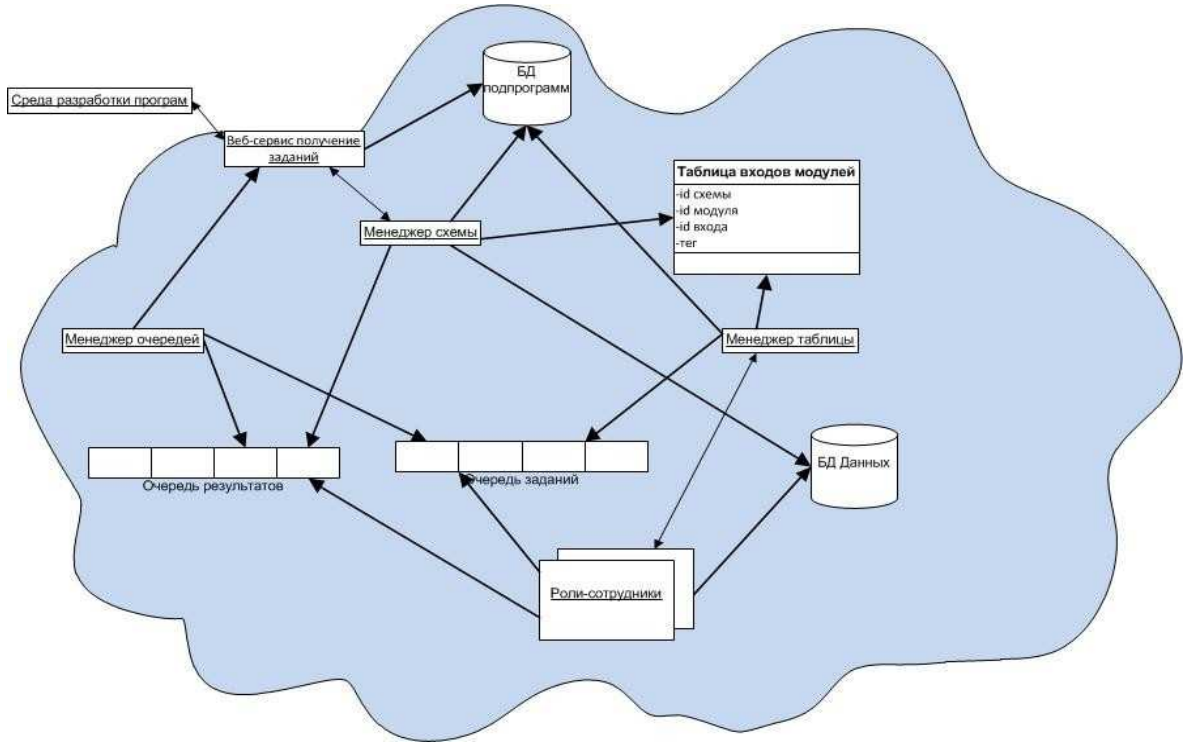

Рис.6 Логическая модель среды выполнения ГСПП на платформе Azure.

На рисунке 6 представлен эскиз логической модели среды выполнения ГСПП на платформе Azure.

На вход, через Веб-сервис получения заданий, поступает программа ГСПП, состоящая из описания схемы вычислений и набора файлов с программным кодом, реализующим логику модулей. Подпрограммы помещаются в БД подпрограмм, а схема ГСПП передается Менеджеру схемы.

Менеджер схемы выполняет анализ граф-схемы, и создает таблицу входов модулей содержащую информацию о текущем состоянии буферов данных, сопоставленных входам модулей. При инициализации программы также проводится размещение значений, подаваемых на установочные входы схемы в БД данных и помещение их идентификаторов в очередь результатов. Разделение самих значений и их идентификаторов позволяет избежать многократного копирования данных большого объема по сети, в случае, если они используются сразу в нескольких модулях. Отдельно рассматривается случай, когда на установочные входы схемы подается очередь значений (к примеру, показания некоторого датчика с периодичностью в несколько секунд), при котором создается отдельный пограничный процесс, отвечающий за правильную трансляцию серии данных в информационные структуры системы.

После запуска программы на выполнение менеджер схемы просматривает очередь результатов, и при нахождении новой порции данных, регистрирует их в соответствие с логической структурой граф-схемы на соответствующих входах в таблице входов модулей.

Менеджер таблицы просматривает данную таблицу, и если на входах какого-либо модуля есть данные с одним тегом, индуцирует новый процесс, предполагающий применение подпрограммы, сопоставленной модулю, к собранным данным с указанным тегом. Для этого он помещает в очередь заданий соответствующее сообщение, содержащее информацию о подпрограмме, которую требуется выполнить, и о данных, к которым её нужно применить.

Роли-сотрудники получают сообщение из очереди заданий, после чего выполняют требуемую подпрограмму для нужных входных данных. В процессе вычислений они помещают идентификаторы выработанных значений в очередь результатов, а сами значения в БД данных.

Менеджер очередей собирает статистику о загруженности очередей, и через веб-сервис передает ее пользователю.

В данный момент среда реализована на тестовой площадке платформы Azure и готовится к проведению тестов производительности.

Недостатки платформы Azure для выполнения высокопроизводительных вычислений

Несмотря на преимущества, которые платформа Azure предоставляет для параллельных вычислений, у нее есть и свои недостатки.

Основным недостатком платформы является отсутствие информации о том, как именно работает компонент Fabric, размещающий роли внутри пула ресурсов, а также данных по скорости передачи данных между элементами платформы. Возможно в будущем эта информация будет представлена разработчиком платформы или установлена путем экспериментирования.

Другим очевидным недостатком использования Azure для высокопроизводительных вычислений является то, что роли запускаются на виртуальных машинах, что ведет к дополнительным накладным расходам по сравнению с работой программ без слоя виртуализации.

Наконец, в настоящий момент доступ к хранилищу происходит через протокол HTTP, что снижает скорость передачи информации.

Несмотря на отмеченные недостатки, на наш взгляд, по мере развития облачные технологии займут свою нишу в сфере параллельных высокопроизводительных вычислений и упростят решение сложных задач.

ЛИТЕРАТУРА::

1. Mache Creeger. Cloud Computing: An Overview. Published in Queue vol. 7, no. 5—
2. see this item in the ACM Digital Library
3. Dion Hinchcliffe. The Next Evolution in Web Apps: Platform-as-a-Service (PaaS). http://bungee-media.s3.amazonaws.com/whitepapers/hinchcliffe/hinchcliffe0408.pdf
4. Кутепов В.П., Котляров Д.В., Маланин В.Н, Панков Н.А. Среда объектно-ориентированного граф-схемного потокового параллельного программирования для многоядерных кластеров. // Материалы шестого Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах», Санкт-Петербург, 12-17 декабря 2006 г. – СПб: Санкт-Петербургский госуниверситет, 2007, Том 1. – С. 253-258.
5. David Chappell. Introducing the Windows Azure Platform. http://davidchappell.com/writing/white_papers.php