самый легкий способ получения текущего состояния виртуальной машины, а хотелось бы видеть эти данные в динамике, в связи с чем требуется написание отдельного модуля для сборки статистики.

Рассматривая анализ производительности, стоит выявить, что гостевая ОС показывает сравнимую производительность с ОС, схожей по аппаратному обеспечению с созданной гостевой ОС (Используя бенчмарки, разница составляет примерно 4-7%)

В целом виртуализация на уровне оборудования имеет много преимуществ:

- Виртуальные машины не занимают места в серверных стойках
- Достаточно просты в администрировании, если есть доступ к гипервизору
- Предоставляют широкие возможности по автоматизации процесса
- Имеют много возможностей по перераспределению нагрузки и миграции систем

При этом стоит отметить, что даже у этого подхода есть недостатки, присущие виртуализации, основанной на представлении – требуется более мощное аппаратное обеспечение для серверов-хостов, а также в случае отказа физического сервера – отказывают и все виртуальные машины.

Исходя из вышесказанного, можно сделать вывод, что виртуализация имеет много преимуществ, и в определенных ситуациях может стать решением многих проблем. При этом в зависимости от конечной цели можно использовать различные подходы, предоставляющий разный уровень нагрузки как на сервер, так и разный уровень затрат на обслуживание.

Список использованных источников:

1. Электронный ресурс, Hyper-V Started Guide, https://technet.microsoft.com/en-us/library/cc732470%28v=ws.10%29.aspx?f=255&MSPPError=-2147217396
2. Aidan Finn, Mastering Hyper-V Deployment, 2010

ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ ВЕБ-СЕРВИСОВ НА ОСНОВЕ МОДЕЛИ АКТОРОВ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Смоляков К. Ю.

Ганжа В. А. - канд. физ.-мат. наук, доцент

В настоящее время широкое распространение получила парадигма «облачных вычислений». Она подразумевает использование нескольких вычислительных узлов и хранилищ различными клиентскими устройствами: персональными компьютерами, смартфонами, встроенным оборудованием, датчиками и т.д. Интерфейсом и связующим звеном для всех этих устройств обычно выступают TCP/HTTP веб-сервисы. Для того чтобы упростить создание распределенных систем с низким временем отклика и высокой доступностью, используются фреймворки на основе модели акторов: Microsoft Orleans для платформы .NET, Akka для платформы JVM, и другие реализации.

Основной задачей данной работы является обзор и анализ высокоуровневых паттернов проектирования распределенных систем для дальнейшего использования при решении прикладных задач. Основная цель — максимально эффективное использование системных ресурсов и преимуществ программной модели акторов.

Модель акторов впервые была упомянута в 1973 году. Широкое распространение в решении задач параллельного программирования в распределенных системах получила сравнительно недавно. Модель акторов представляет собой систему изолированных друг от друга объектов, которые общаются между собой посредством асинхронной передачи сообщений. Внутреннее состояние каждого актора может быть изменено только самим актором при получении сообщения. Каждая операция по обработке полученного сообщения выполняются в контексте одного программного потока. Таким образом прикладному программисту не нужно беспокоиться о проблемах параллельного программирования.

В дальнейшем для решения прикладных задач модель акторов была расширена. Так, например, в компании Microsoft была разработана библиотека Orleans, которая использует собственную расширенную модель «виртуальных акторов», основными принципами которой являются:

1. **Вечное существование.** В Orleans нет возможности создавать и удалять экземпляры акторов. Акторы являются чисто логическими сущностями и всегда существуют (виртуально), к ним всегда можно обратиться по известному ключу.
2. **Автоматическая активация.** Среда выполнения Orleans автоматически создает и удаляет экземпляры акторов на основе запросов клиентов и имеющихся ресурсов. В любой момент времени у каждого актора может быть несколько активаций, либо не быть ни одной.
3. **Независимость от размещения.** Экземпляр актора может быть создан в разных узлах системы, в зависимости от времени и других факторов. Иногда актор может оставаться чисто виртуальное сущностью, то есть не иметь физического размещения.
4. **Автоматическое масштабирование.** В данный момент Orleans поддерживает два режима

активации акторов. Первый разрешает существование только одного экземпляра актора в системе (single activation mode). Второй режим позволяет создавать несколько экземпляров в зависимости от загрузки (stateless worker mode).

Основными преимуществами модели «виртуальных акторов» являются высокая продуктивность разработки и автоматическая масштабируемость. Таким образом для использования нет необходимости быть экспертом в области распределенных систем.

В процессе исследования были выявлены следующие критерии, при выполнении которых использование модели акторов может быть оправданным:

- Большое количество слабо связанных программных сущностей (100 – 1 млн).
- Программные сущности достаточно малы и могут быть однопоточными.
- Взаимодействия типа запрос - ответ или начало – мониторинг – завершение.
- Заранее известно, что понадобится более одного сервера.
- Нет необходимости в глобальной координации операций, только между небольшим количеством программных сущностей за один раз.

Критерии, при которых применение модели акторов может быть проблематичным:

- Необходимость прямого доступа к памяти одного объекта другому.
- Маленькое количество больших сущностей, необходима длительная (возможно многопоточная) обработка данных
- Необходима глобальная координация и целостность данных

Для решения наиболее часто возникающих задач при помощи модели акторов могут быть использованы следующие паттерны:

1. Умный кэш (Smart Cache).

Используется для оптимизации производительности доступа к хранилищу данных. Позволяет ускорить операции чтения, так как кэширующий актор хранит нужные данные в оперативной памяти. Позволяет буферизовать операции записи, снизив при этом нагрузку на базу данных.

Кэш называют «умным», так как кэшируемые данные хранятся в локальных переменных экземпляра актора и могут быть представлены различными сложными структурами данных: очередями, сортированными множествами, хеш-таблицами.

Паттерн можно использовать в следующих случаях:

- Хранилище данных сильно замедлят время отклика системы в целом
- Операции чтения превалируют над операциями записи
- Нет необходимости делать сложные запросы к данным, либо данные представлены документами
- Необходимы быстрые операции записи, но допустима частичная потеря данных

Примеры использования: таблица результатов для real-time игры, очередь задач, сервис конвертирования валют

2. Распределенная сеть / граф (Distributed Network / Graph)

Является разновидностью известного паттерна publish / subscribe, позволяет моделировать сложные взаимосвязи с помощью объектной модели.

Паттерн можно использовать, когда о произошедшем событии нужно асинхронно уведомлять нескольких участников системы. Может быть реализован различными способами: с помощью потоков данных (например, Orleans Streaming), либо прямой передачей сообщения от актора-источника к акторам-подписчикам.

Примеры использования: социальные сети, групповые чаты, «интернет вещей»

3. Редуктор (Reducer)

Обеспечивает иерархическую структуру вида, необходимую для получения агрегата значений, хранящихся во внутреннем состоянии многих акторов. Иерархия в этом случае выстраивается в следующем порядке (снизу-вверх): актор-значение, актор-промежуточное значение, актор-агрегат. Обновление значение происходит единовременно, либо по таймеру. В качестве агрегата можно вычислять среднее значение, сумму, количество или другие показатели.

Примеры использования: «интернет-вещей», сбор телеметрических и статистических данных

Существуют также и другие, более специфичные паттерны использования модели акторов, такие как Cadence, Resource Manager, Distributed Computation, Stateful Service Composition и др.

Список использованных источников:

1. Bernstein P., Bykov S., Geller A., Kliot G., Thelin J. Orleans: Distributed Virtual Actors for Programmability and Scalability, no. MSR-TR-2014-41, 2014 – 13p.
2. Параллельные вычисления с помощью акторов [Электронный ресурс]. — Режим доступа: https://github.com/anton-k/ru-neophyte-guide-to-scala/blob/master/src/p14-actors.md
3. Tuna H., Martinez C. Applied Actor Model with Orleans [Электронный ресурс]. — Режим доступа: https://github.com/hataytuna/Distributed/blob/master/Applied%20Actor%20Model%20with%20Orleans%20v1.1.pdf
4. Bykov S., Kliot G., Roberts M., Thelin J. Orleans Best Practices, Microsoft Research, 2014 — 14p.