**Исследование Gossip протокола**

**Саськов Л. К.**

Введение

Gossip протокол – это эффективный метод обмена информацией в распределенных системах.

Gossip протокол используется для обеспечения надежного и масштабируемого обмена информацией между узлами. Он работает путем случайного выбора узлов для обмена сообщениями, что позволяет избежать проблем, связанных с централизованным подходом.

В этом отчете мы рассмотрим вариант реализации Push Gossip. В данном варианте ноды посылают сообщения друг-другу, нода не запрашивает информацию самостоятельно (No pull).

Фреймворк для исследования

**Экземпляр ноды**

Исследование проводилось с использованием программы Docker, множество docker контейнеров на 1 машине имитировало работу множества нод.

Для работы контейнера было написано Java приложение gossip-worker. Для общения друг с другом контейнеры поддерживали Rest Api протокол, состоящий из 2х методов: push и message.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, Шрифт

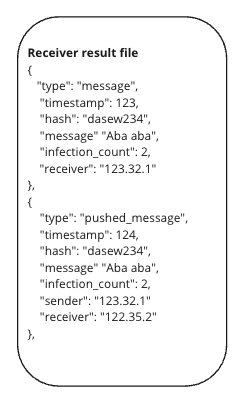
Автоматически созданное описание

Метод **message** предназначался для внешнего клиента – реального отправителя сообщения. Метод **push** для внутреннего распространения информации о пришедшем сообщении кластеру.

При вызове метода **push** контейнер вызывал методы **push** у других случайных нод кластера, количество получателей такого вызова задавалось параметром **infection\_count**.

Контейнер сохранял (логировал) информацию о приходящих push и message вызовах в файл. Благодаря механизму volumes в Docker, все файлы логов контейнеров могли быть прочитаны для анализа с машины, где запускались контейнеры. Также стоит упомянуть, что все контейнеры имели синхронизированное время, ведь запускались на одной реальной машине.

*Пример файла лога контейнера:*

**

*Схема кластера из 5 контейнеров:*

Изображение выглядит как диаграмма, План, зарисовка, Технический чертеж

Автоматически созданное описание

**Скрипты для автоматизации работы с кластером**

Для упрощения анализа и ускорения исследований был разработан ряд скриптов на языке python. Все скрипты описанные ниже запускаются файлом *hot\_star.py*

**Скрипт для формирования кластера**

Файл *cluster\_maker.py*

Cкрипт на питоне, который формирует docker-compose файл.

Основные параметры скрипта:

* package\_loss – процент потери пакетов в сети (был реализован на уровне java приложения gossip-worker, прокидывается туда, как параметр)
* containers\_count – размер кластера

**Скрипт для тестирования**

Файл *tester.py*

Данный скрипт запускает кластер и формирует трафик сообщений.

Основные параметры скрипта:

* infection\_count – количество нод для распространения push-а
* messages\_count – количество сообщений в тесте

**Скрипт агрегирования данных**

За сбор данных со всех файлах логов контейнеров и приведение их в единую таблицу отвечает скрипт *analyzer.py*.

*Кусочек итоговой таблицы, получаемой в ходе работы всех скриптов:*Изображение выглядит как текст, снимок экрана, число

Автоматически созданное описание

Анализ

**Исследуемые статистики**

Для разного процента потери пакетов были вычислены:

* **infected\_perc** - процент распространения сообщения
* **duplicate\_perc** - процент дубликатов в зависимости от infection count
* **mean\_send\_ms** - среднее время распространения сообщения
* **max\_send\_ms** - время последнего (то же самое, что прошлый пункт, только берем не среднее, а максимум)

Те же статистики были вычислены для разного infection count.

**Графики**

После работы скриптов была проанализирована таблица со статистиками для кластера из 10 контейнеров и 50 контейнеров. В данном отчете показаны графики только для 50 контейнеров, чтобы посмотреть графики 10 контейнеров можно открыть файл *graphics.ipynb*. В отчете отображены только графики 3d поверхностей, поскольку они более выразительны и отображают полную картину, однако в файле *graphics.ipynb* можно найти и 2d графики.

**Поверхность процента распространения сообщения (infected\_perc)**

Изображение выглядит как снимок экрана, диаграмма, прямоугольный, Прямоугольник

Автоматически созданное описание

Поверхность представляет из себя скат с вершиной при минимальном проценте потери пакетов и максимальном *infection\_count.* Обратите внимание, что при 90 процентной потере пакетов, увеличение infection\_count слабо помогает, gossip протокол плохо работает в таких условиях. А вот 40 процентов потерь ему не страшны, при *infection\_count = 7* мы достигаем достаточно большого процента распространения около 80 процентов.

Стоит обратить внимание, что максимальный показатель распространения 95 процентов, 100 процентов распространения сложно добиться даже в идеальных условиях, ведь всегда есть вероятность оставить какой-то контейнер без пуша (при условии, что infection\_count меньше размера кластера).

Также не лишнем будет сказать, что релевантным для дальнейших исследований времени распространения сообщений является квадрат [package\_loss\_perc: [0, 50], inf\_count: [5, 9]], данные за пределами этого квадрата показывают, что gossip распространил меньше 50 процентов сообщений.

**Поверхность процента сообщений-дубликатов (duplicate\_perc)**

Изображение выглядит как снимок экрана, Прямоугольник, прямоугольный, линия

Автоматически созданное описание

Поверхность похожа на поверхность распространения сообщений, это означает, что высокий процент распространения ведет к дубликатом и тем самым увеличению нагрузки на сеть.

**Поверхность среднего времени распространения сообщения (mean\_send\_ms)**

**Изображение выглядит как диаграмма, дизайн

Автоматически созданное описание**

Если рассматривать поверхность в рамках релевантного квадрата [package\_loss\_perc: [0, 50], inf\_count: [5, 9]], который был подробно описан в разделе о поверхности распространения то можно прийти к выводу, что перед нами впадина или скат с самой низкой точкой при минимальном проценте потери пакетов и максимальном *infection\_count*. То есть чем меньше потеря пакетов и чем больше infection\_count, тем быстрее происходит распространение сообщения, сообщению приходится пройти меньше промежуточных звеньев, чтобы добраться до адресата.

**Поверхность времени последнего (max\_send\_ms)**

Изображение выглядит как диаграмма, снимок экрана, дизайн

Автоматически созданное описание

Здесь мы видим впадину, аналогичную прошлому графику.

Выводы

1. Не стоит использовать Push Gossip для систем в которых важно полное распространение сообщения, в данном варианте протокола очень сложно добиться 100 процентного покрытия. Используйте Push-Pull gossip.
2. Push gossip хорошо себя показывает при сети с потерей пакетов до 50 процентов.
3. Gossip протокол создает достаточно большую нагрузку на сеть из-за возникающих сообщений-дубликатов, стоит относиться к увеличению параметра infection\_count с осторожностью.