Санкт–Петербургский государственный университет

Системное программирование

Лямин Владимир Андреевич

Pacemaker: оптимизация обхода графа

Отчет по учебной практике

Научный руководитель:
ассистент кафедры ИАС Смирнов К. К.

Санкт-Петербург
2023 г.

Содержание

1. Введение

Отказоустойчивый кластер – это группа сервисов, которая гарантирует минимальное время простоя ресурсов [1]. Если один из узлов кластера потерял связь с остальными, то отказоустойчивая кластеризация должна запустить процесс восстановления. В него входят следующие задачи:

- перенести ресурсы с отказавшего узла на рабочие;
- остановить ресурсы на отказавшем узле кластера;
- изолировать отказавший узел.

Одним из компонентов отказоустойчивого кластера является менеджер ресурсов Pacemaker. Его обязанности – это достижение максимальной доступности ресурсов, а также защита их от сбоев [2]. В его задачи входят:

- обнаружение и восстановление сбоев на уровне узлов и сервисов;
- поддержка одного или нескольких узлов на кластер;
- автоматически синхронизируемая конфигурация, которую можно обновлять с любого узла.

Pacemaker является одним из компонентов отказоустойчивого кластера от ClusterLabs. В его обязанности входит восстановление работоспособности ресурсов при старте или сбое узлов кластера. Для перевода кластера в работоспособное состояние строится ориентированный граф, который в своих узлах содержит действия [3].

Далее будем говорить о задачах, которые были поставлены непосредственно перед автором.

2. Постановка задачи

Цель данной работы – это оптимизировать время обработки графа действий. Для этого были поставлены следующие задачи:

- развернуть простой HA кластер на двух виртуальных машинах с несколькими ресурсами;
- развернуть тестовый пример от компании YADRO. Изучить построенный flame graph;
- изучить существующий алгоритм построения и обхода графа действий;
- ускорить работу графа;
- провести замеры производительности и получить численные характеристики исходного и своего решений.

3. Архитектура

Pacemaker — это высокодоступный диспетчер ресурсов кластера, а именно программное обеспечение, которое работает на множестве хостов для сохранения целостности и минимизации времени простоя нужных ресурсов.

3.1 Архитектура кластера

Если смотреть на архитектуру кластера на высоком уровне(рисунок 1), то можно выделить следующие компоненты:

- Resources – это то, ради чего существует кластер, а именно сервисы, которые должны быть высокодоступны;
- Resource agents – это сценарии или компоненты операционной системы, которые запускают, останавливают и контролируют ресурсы с учетом их параметров;
- Fence agents – это скрипты для извлечения узла из кластера
- Cluster membership layer – данный компонент обеспечивает надежный обмен информацией
- Cluster resource manager – Pacemaker предоставляет мозг, который реагирует на события, которые произошли в кластере(присоединение, отказ узла);
- Cluster tools – данные инструменты предоставляют интерфейс для взаимодействия пользователей с кластером.

3.2 Архитектура Pacemaker

Сам Pacemaker состоит из нескольких демонов(рисунок 2), которые работают сообща. pacemakerd – основной процесс Pacemaker, который порождает все остальные демоны и возрождает их, если они неожиданно

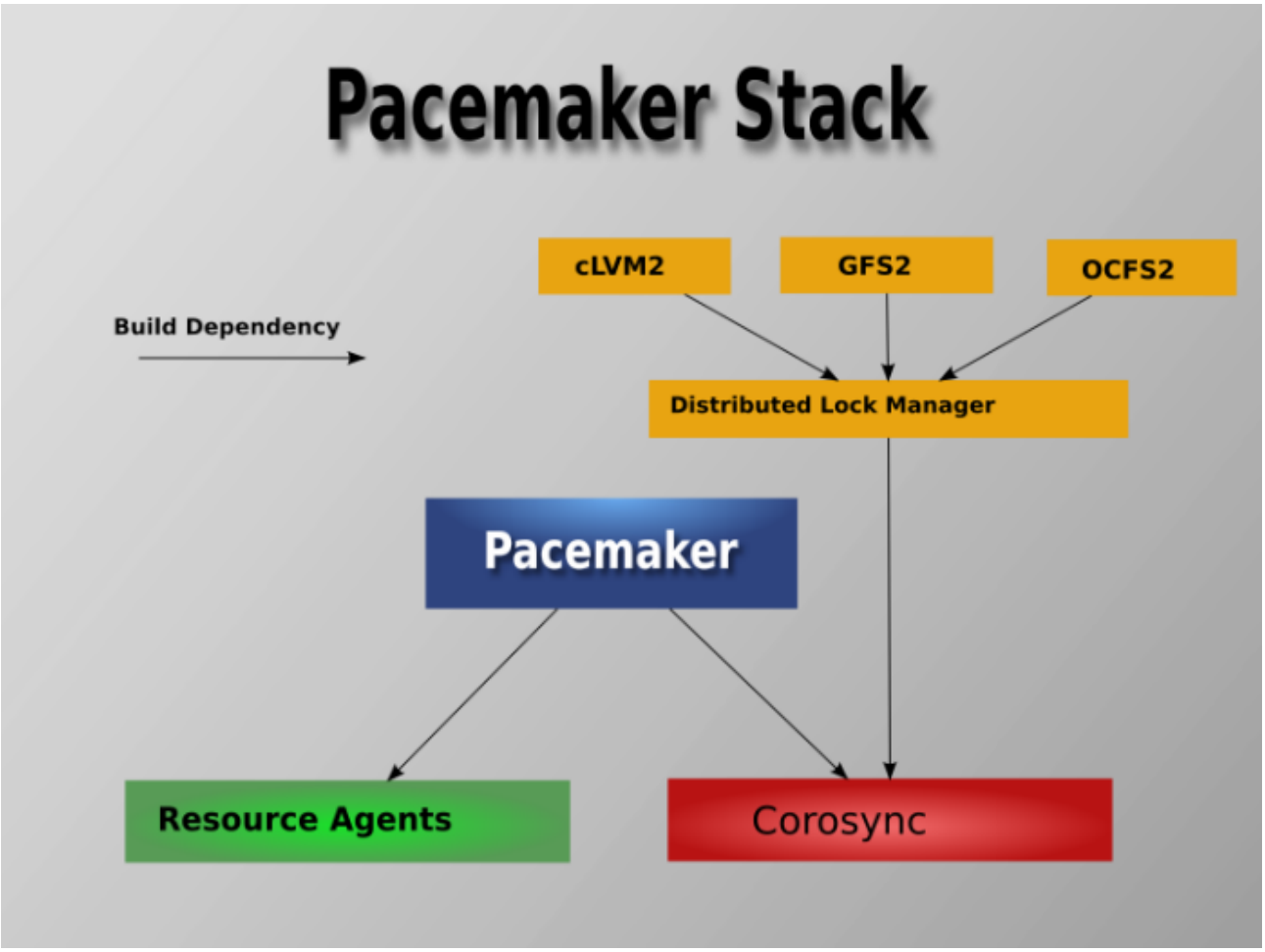

Рис. 1: Архитектура кластера

завершают работу. pacemaker-controld координирует все дейтсвия. Локальный исполнитель pacemaker-execd обрабатывает запросы на выполнение агентов ресурсов на локальном узле кластера и возвращает результат. А также pacemaker-fenced обрабатывает запросы на изоляцию узлов.

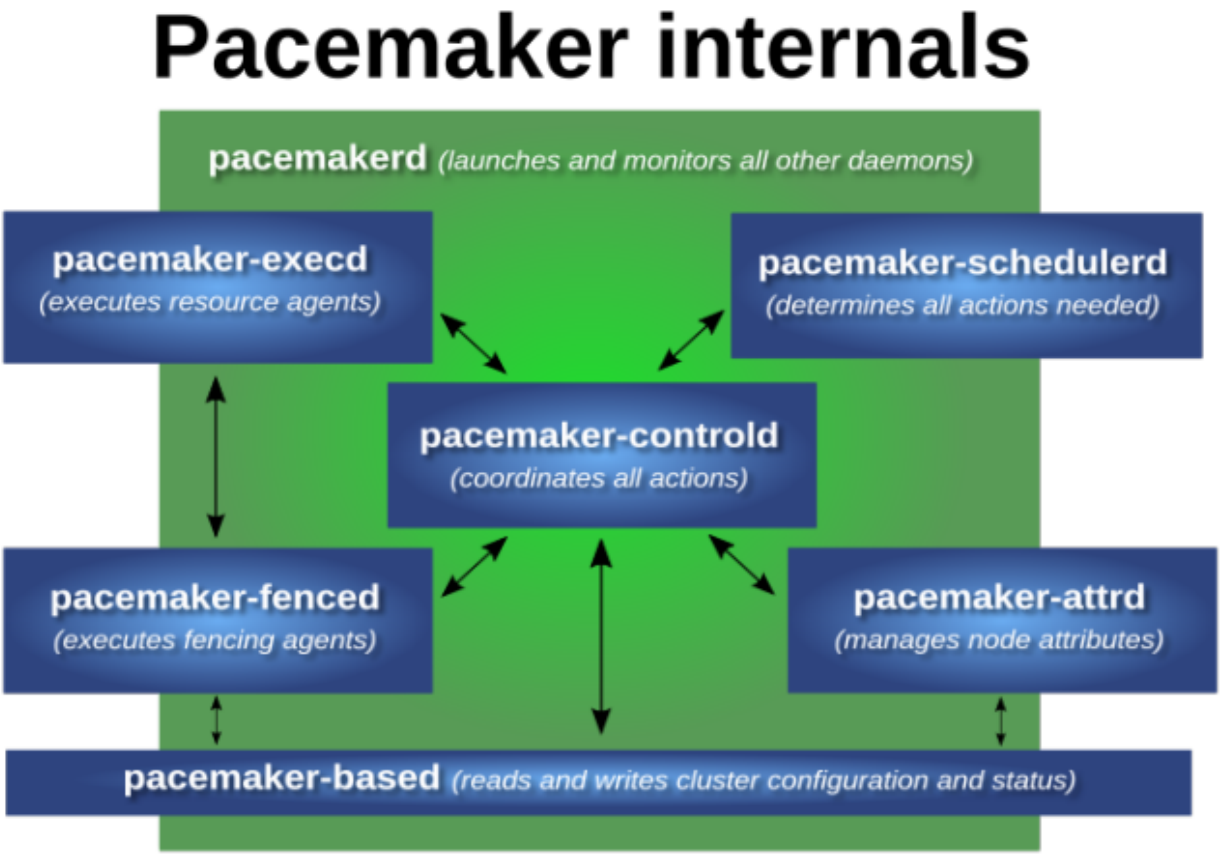

Рис. 2: Архитектура pacemaker

4. Настраивание простого HA кластера с ресурсами

4.1 Подготовка кластера

Для настройки кластера были выбраны две виртуальные машины Ubuntu, так как данный дистрибутив является одним из наиболее развивающихся в мире.

Для начало необходимо удостовериться, что машины могут взаимодействовать между собой. Это выполняется с помощью команды ping. Затем было настроено, чтобы машины могли взаимодействовать между собой посредством их имен, а не IP-адресов.

Следующим этапом идет установка и настройка программного обеспечения кластера. Для этого на каждом узле необходимо разрешить службы сервиса через межсетевой экран. Далее следует проверить установку систем corosync(групповое общение для отказоустойчивых кластеров) и pacemaker. Затем можно запускать кластер.

4.2 Добавление ресурсов в кластер

Первым ресурсом для кластера был выбран плавающий IP-адрес, который может использоваться на любом узле. Независимо от того, где запущены какие-либо службы кластера, конечные пользователи должны иметь возможность связываться с ними по согласованному адресу. К качестве имя для данного ресурса было выбрано имя ClusterIP.

Следующим ресурсом был выбран Apache HTTP Server поскольку он является функцией многих кластеров [5] и относительно прост в настройке. Для его настройки необходимо убедиться, что Apache установлен на обоих узлах кластера. Далее нужно создать страницу сайта для Apache. Пример кода показан на рисунке 3.

```
# cat <<-END >/var/www/html/index.html
 <html>
 <body>My Test Site - $(hostname)</body>
 </html>
END
```

Рис. 3: Пример страницы Apache сервиса

4.3 Настройка ограничений ресурсов

Чтобы снизить нагрузку на какую-либо одну машину, Pacemaker обычно пытается распределить ресурсы по узлам кластера. Однако возможно указать кластеру, что два ресурса связаны и должны работать на одном хосте, так как один ресурс не может без другого. Это делается с помощью команды colocation.

Так как сервис Apache был привязан к ClusterIP, то необходимо, чтобы он запускался позже, чем ресурс ClusterIP. Это можно сделать с помощью команды order. По умолчанию все ограничения порядка являются обязательными. Это означает, что если ClusterIP необходимо остановить, то WebSite должен быть остановлен первым, а если WebSite нужно запустить, то сначала должен запуститься ClusterIP. Это также означает, что

восстановление ClusterIP вызовет восстановление WebSite, что приведет к его перезапуску.

4.4 Проверка кластера

Для проверки настроек кластера была использована команда crm_simulate. Она позволяет симулировать события, которые могут произойти в кластере. На рисунке 4 показан flame-график, который показывает восстановление ресурсов после отключения первого узла. На нем хорошо видно два сервиса: ClusterIP и WebSite.

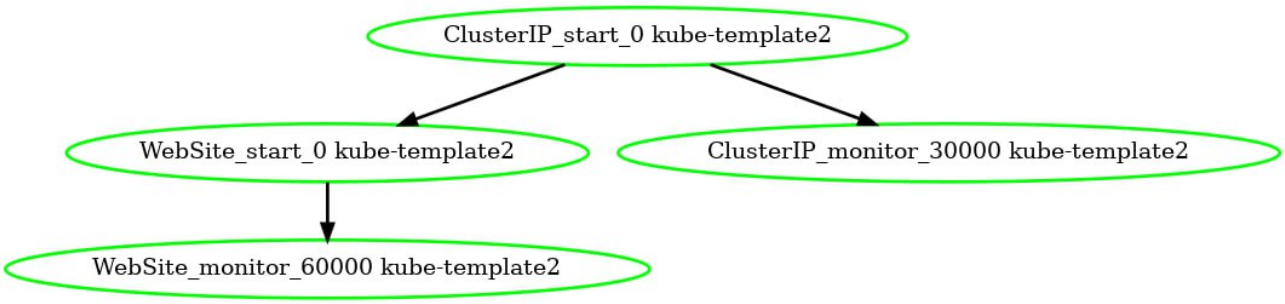

Рис. 4: Граф отказоустойчивого кластера с двумя ресурсами

5. Оптимизация системы Pacemaker

На рисунке 5 показан результат расчёта графа действий в виде flame graph. flame graph – это метод для визуализации процессорного времени потраченного на функции программы. Данный граф читается снизу вверх, также имеет значение относительная толщина каждой полоски и глубина стека. Порядок полосок и их цвет нужен только для удобства чтения графа [4].

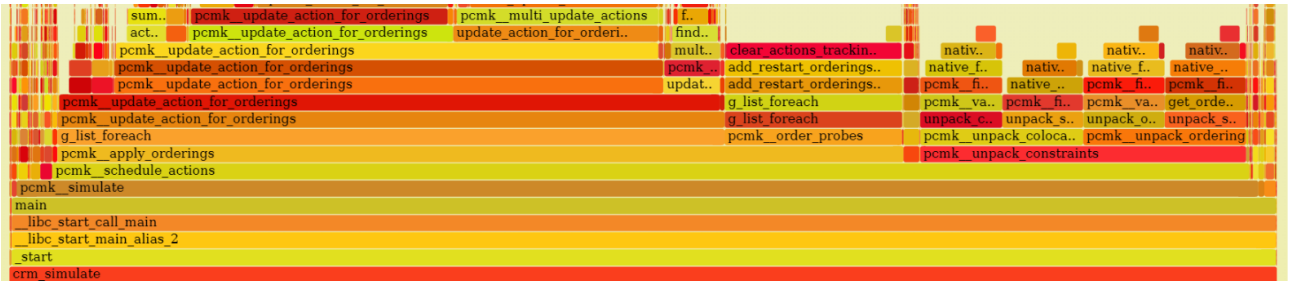

Рис. 5: Flame-график системы pacemaker

На рисунке хорошо видно, что программа состоит из нескольких частей:

- распаковка заданной конфигурации – функции *_unpack_*;
- создание возможных действий для каждого ресурса – функция schedule_resource_actions;
- расчёт последовательности действий – функция pcmk__apply_orderings.

В качестве участка для поиска возможных путей оптимизации программы была выбрана распаковка заданной конфигурации(рисунок 6), так как она занимает достаточно большой объем процессорного времени, а также были подозрения, что работа с данными была сделана не оптимально. Как видно на слайде, наиболее проблемным участком кода является функция native_find_rsc. Принцип работы данного участка кода заключается в том, что программа идет по списку ограничений, а затем для каждого ограничения сопоставляется его ресурс путем полного перебора.

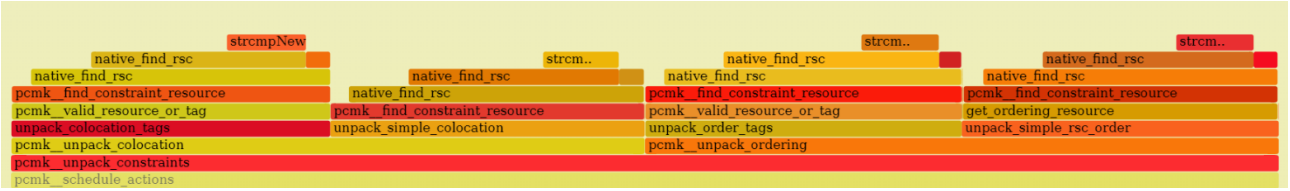

Рис. 6: Распаковка конфигурации в системе pacemaker

5.1 Проведение экспериментов с функциями сравнения строк

В ходе исследования было решено провести эксперименты с различными функциями сравнения строк. Для этого была написана функция сравнения на языке Си и ассемблер, а также взят метод из библиотеки string.h strcmp(). В качестве тестовых данных была выбрана часть идентификаторов ресурсов.

На рисунке 7 можно посмотреть результаты эксперимента. На нем показаны резульаты трех методов при разной степени оптимизации. -O0 – это оптимизация по времени компиляции, т.е. по умолчанию. -O2 – это оптимизация по времени выполнения. Видно, что функция на ассемблере выигрывает только при оптимизации по умолчанию, а далее она начинает проигрывать по времени.

	Strcmp, c	Ccompare, c	Acompare,c
-O0	0,015	0,016	0,010
-O2	0,004	0,08	0,011

Рис. 7: Методы сравнения строк

После проведения данных экспериментов было решено пойти путем индексации массива данных.

5.2 Индексация массива данных

В качестве индексации данных была выбрана хеш-таблица, так как это удобный и эффективный инструмент. В качестве способа по разрешению коллизий был выбран метод списков, так как он является наиболее простым и понятным в реализации. В качестве ключа был выбран идентификатор ресурса, а в качестве данных сам ресурс. Количество ячеек было выбрано на уровне двухсот, так как это наиболее оптимальное значение.

Были проведены несколько экспериментов с хеш-функцией. В них были рассмотрены следующие варианты: сложение кода каждого символа строки, полиномиальный хеш(умножение каждого символа строки на степень некоторого числа), а также сложение каждого третьего символа строки. В итоге наиболее оптимальным оказался последний вариант.

Чтобы реализовать новый алгоритм с хеш-таблицей, была найдена функция, которая запускает цикл по всем ограничениям ресурсов. В ней была инициализирована хеш-таблица. Далее был найден метод, который искал ресурс для каждого ограничения, и в нем был реализован поиск по хеш-таблице.

6. Проведение измерений

После внесения изменений были проведены несколько измерений. Благодаря им было установлено, что время работы функции на виртуальной машине Kali уменьшилось с 43 секунд до 0.12 секунд. А ее доля в процессорном времени программы сократилась с 25% до 0.09%, что является приемлемым результатом. Новый flame-график можно посмотреть на рисунке 8. На нем видно существенное сокращение участка распаковки конфигурации.

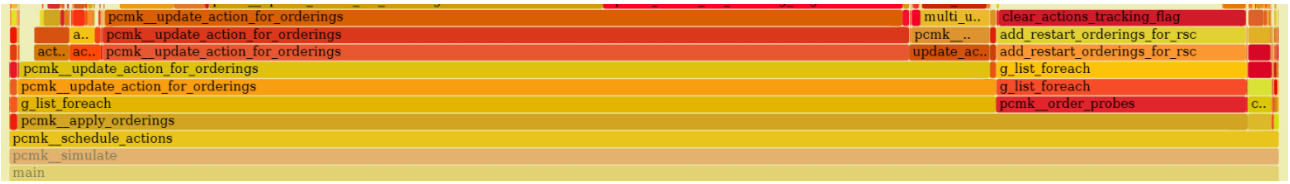

Рис. 8: Flame-график с оптимизированным алгоритмом обработки конфигурации

7. Заключение

7.1 Обзор работы, проделанной в рамках осеннего семестра

- был развернут простой HA кластер на двух виртуальных машинах с несколькими ресурсами;
- развернут тестовый пример от компании YADRO и изучен построенный flame graph;
- изучин существующий алгоритм построения и обхода графа действий;
- ускорена работа графа;
- проведены замеры производительности и получены численные характеристики исходного и своего решений.

Список литературы

[1] Принцип работы отказоустойчивых кластеров – (дата обращения: 27.11.2022). https://docs.ispsystem.ru/vmmanager-admin/klastery/otkazoustojchivye-klastery/printsip-raboty-otkazoustojchivyh-klasterov

[2] Отказоустойчивый кластер (High Availability) на основе Pacemaker – (дата обращения: 27.11.2022). https://docs.altlinux.org/ru-RU/alt-server/9.2/html/alt-server/ch54.html

[3] Pacemaker: обход графа – (дата обращения: 27.11.2022). https://se.math.spbu.ru/diplomas/theme.html?id=28

[4] Clusters from Scratch – (дата обращения: 27.11.2022). https://clusterlabs.org/pacemaker/doc/2.1/Clusters_from_Scratch/pdf/Clusters_from_Scratch.pdf

[5] Flame-графики: «огонь» из всех движков – (дата обращения: 27.11.2022). https://habr.com/ru/company/otus/blog/523148/