Санкт-Петербургский государственный университет

Кафедра системного программирования

Pacemaker: оптимизация построения графа

Иван Евгеньевич Азарников, группа 20.Б11-мм

Научный руководитель: К.К. Смирнов, старший преподаватель кафедры ИАС

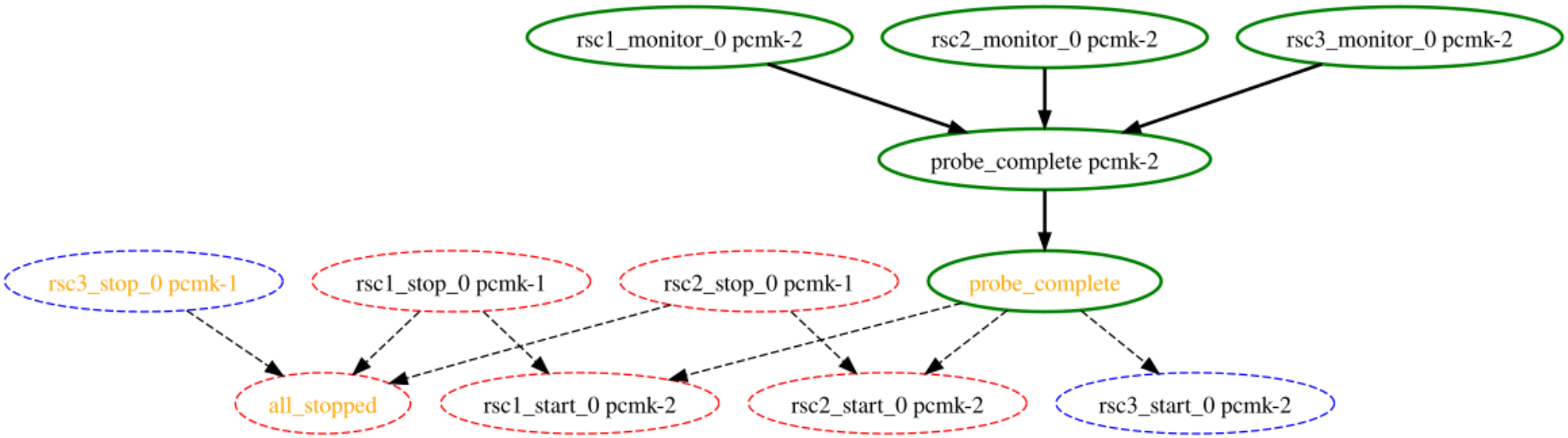
Санкт-Петербург
2022

Введение

- Отказоустойчивый кластер — группа серверов, гарантирующая минимальное время простоя за счёт аппаратной избыточности.
- Pacemaker - менеджер ресурсов, один из компонентов отказоустойчивых кластеров от ClusterLabs
- Восстановление работоспособности сервисов при старте или сбое узлов кластера

Введение

- Строится ориентированный граф, в узлах которого находятся действия, который нужно выполнить для перевода кластера в работоспособное состояние.
- Pacemaker медленно работает с большим количеством ресурсов



Постановка задачи

Целью работы является ускорение построения графа действий
Задачи:

- Развернуть тестовый пример от YADRO, изучить построенный flame graph
- Выразить на DDlog ограничения Pacemaker
- Изучить существующий алгоритм построения графа
- Повысить скорость построения графа действий:
  - Реализовать алгоритм построения графа на DDlog
  - Добавить взаимодействие с реляциями в Pacemaker
- Сравнить две реализации на тестовом примере

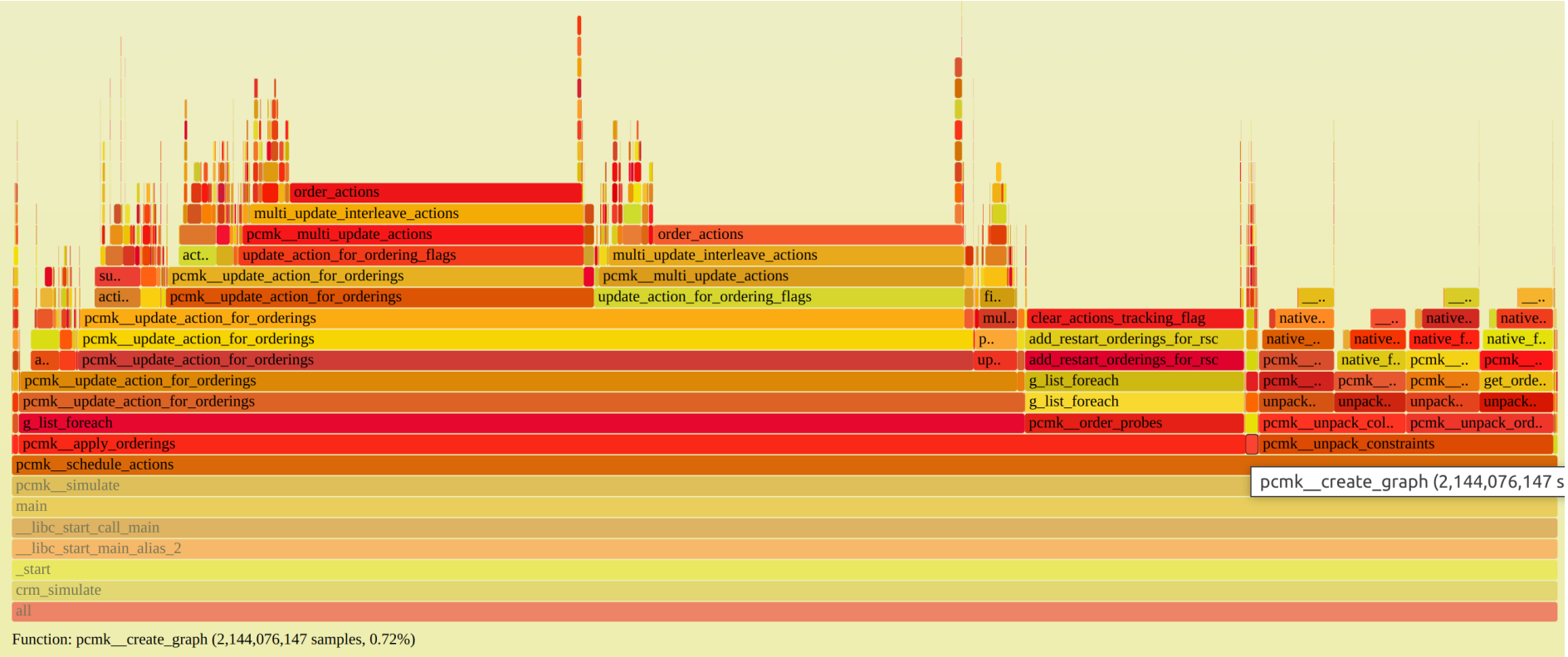
Pacemaker constraints

Ограничения:

- Location constraint
- Colocation constraint
- Ordering constraint

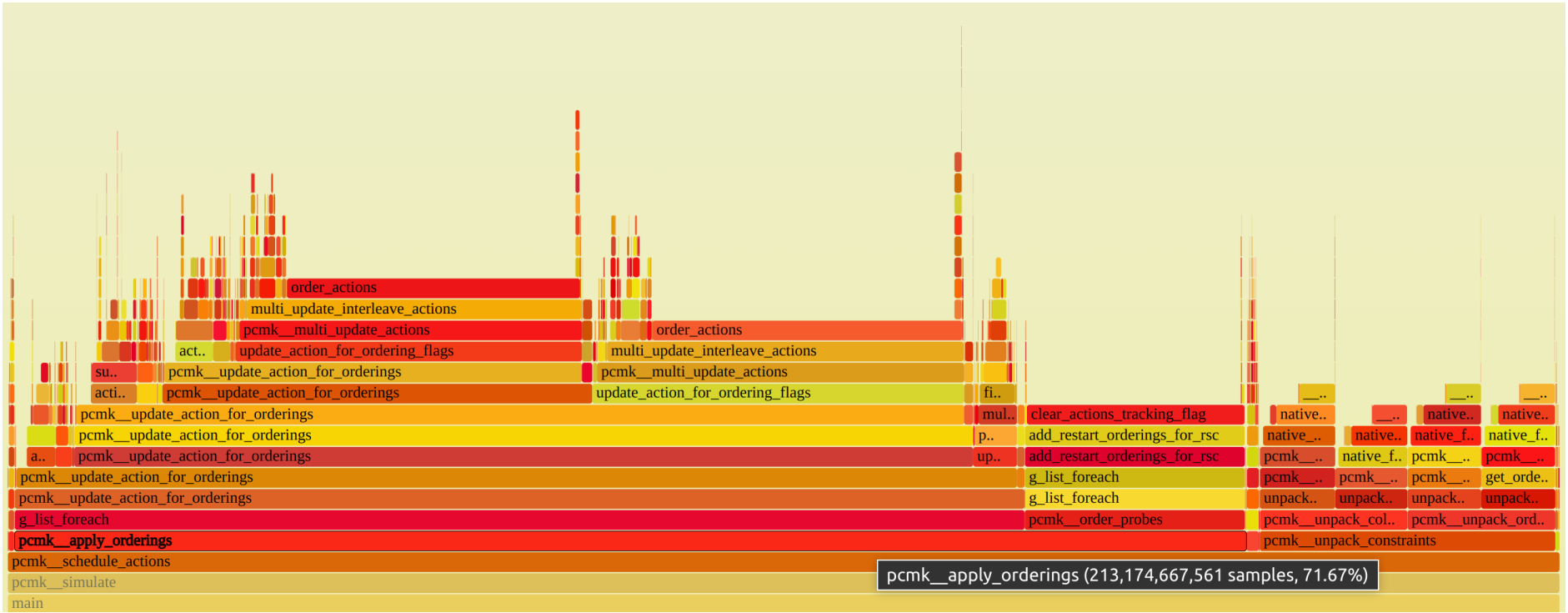
```
<rsc_location id="loc-1" rsc="Webserver" node="sles-1" score="200"/>
<rsc_location id="loc-2" rsc="Webserver" node="sles-3" score="0"/>

<rsc_colocation id="coloc-1" rsc="mlun-1" with-rsc="acl-1"/>

<rsc_order id="order-1" first="IP" then="Webserver"/>
<rsc_order id="order-2" first="Database" then="Webserver"/>
```

Flame graph (Построение графа)



- Вершины графа — действия, ребра — список actions_before
- Основная сложность заключается в применении порядковых ограничений и получении конечного списка actions_before

Flame graph (Применение порядковых ограничений)



actions_before

У каждого действия есть свой список actions_before, который состоит из оберток над действиями, содержащих флаги, необходимые для поддержания порядка

```
typedef struct pe_action_wrapper_s {
  enum pe_ordering type;
  enum pe_link_state state;
  pe_action_t *action;
}
```

- В процессе применения ordering constraint'ов необходимо обновлять действия
- После обновлений списки actions_before могут содержать новые элементы, которые тоже нужно обработать
- Нужно повторно обновлять все действия, так как могли появиться необработанные зависимости

DDlog

DDlog[1] — декларативный язык программирования для инкрементальных вычислений.

- Богатая система типов
- Арифметика
- Интеграция с C, Rust, Java

[1] https://research.vmware.com/projects/differential-datalog-ddlog

Пример. Транзитивное замыкание

```
input relation Edge(x: Node_t, y: Node_t)
relation TC(x: Node_t, y: Node_t)
TC(x, y) :- Edge(x, y).
TC(x, y) :- TC(x, z), Edge(z, y).

start;
insert Edge(1, 2);
insert Edge(2, 3);
commit;
dump TC;

TC{.x = 1, .y = 2}
TC{.x = 2, .y = 3}
TC{.x = 1, .y = 3}
```

Интеграция с C

```c
#include "ddlog.h"

ddlog_prog *prog = ddlog_run(1, true, NULL, NULL);
table_id EdgesTableID = ddlog_get_table_id(prog, "Edge");
ddlog_record **struct_args = (ddlog_record**)malloc(2 * sizeof(ddlog_record*));

struct_args[0] = src;
struct_args[1] = dst;
ddlog_record *new_record = ddlog_struct("Edge", struct_args, 2);

ddlog_transaction_start(prog);
ddlog_cmd *cmd = ddlog_insert_cmd(EdgesTableID, new_record);

ddlog_apply_updates(prog, cmd, 1);
ddlog_transaction_commit(prog);
```

Обновление флагов

Pacemaker:

```
if (pcmk_is_set(other->type, pe_order_then_cancels_first)
  && !pcmk_is_set(then->flags, pe_action_optional)) {
  pe__set_action_flags(other->action, pe_action_optional)
}
```

DDlog:

```
ActionOptional (action) :-
  ActionRel (_, action, _),
  is_set(action.flags, pe_action_optional()).


ActionOptional(first.action) :-
  OrderThenCancelsFirst(then, first),
  not is_set(then.flags, pe_action_optional()).
```

Планируемое взаимодействие с DDlog

- При установке/удалении флага запрос modify добавляется в транзакцию
- При добавлении действия в список actions_before запрос insert добавляется в транзакцию
- В конце обработки каждого действия транзакция отправляется для выполнения DDlog'ом
- По окончании всех обновлений все реляции выгружаются в Pacemaker

Ссылки

- Репозиторий с кодом:
  `https://github.com/esvault/ddlog_immersion`
- Pacemaker: `https://github.com/ClusterLabs/pacemaker`
- DDlog: `https://github.com/vmware/differential-datalog`