Технологии автоматического тестирования

на примере использования решений In-Memory Data Grid

ИСП РАН

Борисенко Олег, Институт Системного Программирования РАН

Задача

Необходимо протестировать для нескольких систем In-Memory Data Grid

- Производительность
- Потребление ресурсов
- Отказоустойчивость
- Масштабируемость
- Поведение при увеличении числа узлов “на лету”

Условия и параметры

- 3 системы (Infinispan, GridGain, Hazelcast)
- N, 2N, 3N клиентов
- M, 2M, 3M серверов с IMDG
- Степени репликации IMDG 2, 3

В сумме минимум 54 варианта.

Важно иметь возможность протестировать и другие системы в случае необходимости: только для GridGain было проведено 1375 тестов с разными параметрами.

Аппаратные ресурсы

1. Блейд-шасси Dell M1000E
2. 8 блейд-серверов Dell M620 со следующими характеристиками:
   1. 2 процессора Intel Xeon E5-2650 v2 (8 cores, 2.6GHz)
   2. 256GB оперативной памяти
   3. 2 сетевых интерфейса с пропускной способностью 10Gbit/s
3. 2 коммутатора Dell MXL 10/40 Gbit/s
4. Система хранения PowerVault MD3820i

http://www.bigdataopenlab.ru/

ИСП РАН

Требования к проведению тестов

- Тестируемые системы не должны влиять друг на друга
- Накладные расходы на тестовую систему должны влиять на результаты незначительно
- Системы должны тестироваться в равных условиях

Инфраструктура для тестов

- Bare-metal
- Виртуальные машины
- Контейнеры

Bare-metal

+ Не имеет промежуточных слоев и накладных расходов на виртуализацию
- Сложно поддерживать систему в “чистом” состоянии — возможны ошибки конфигурации
- Потенциально после каждого тестирования результат следующего становится может быть менее достоверным

Виртуальные машины

+ Удобно использовать: подготовка образа где угодно
+ Гибкая схема развертывания: легко разбивать одну физическую машину на несколько более слабых
- Огромное влияние на скорость сети
- Непредсказуемое влияние на производительность в целом

Контейнеры

+ Можно добиться производильности bare-metal
+ Можно избавиться от накладных расходов на виртуализацию сети
- Сравнительно молодая технология, много известных багов
- Могут быть непредсказуемые ошибки “ядерного характера” (CentOS под Ubuntu)

Схемы развертывания

- Полностью вручную
- Заготовка образов + система скриптов
- Системы оркестрации

Ручное развертывание

- Сложно воспроизвести на другом железе
- Очень легко запутаться и потерять согласованность конфигураций
- Занимает много времени
- Не позволяет добавить к тестируемым системам еще одну и быстро проверить ее в тех же условиях

Заготовка образов

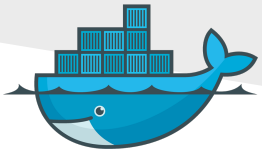
+ Позволяет быстро запускать полностью чистую “базовую” систему.
- Не позволяет использовать динамические параметры, зависящие от других узлов.

Подход отличный, но не достаточный.

Подготовка образов

LXC 

- один образ — один файл (R/W)
- запуск при помощи копирования образа (долго)

Docker 

- UnionFS для образов (R/O)
- запуск из read-only объединения слоев UnionFS (быстро)

Системы оркестрации

+ Позволяют настроить базовую систему до нужного состояния
+ Можно использовать роли для различных хостов
+ Полный контроль за ошибками: каждое действие похоже на транзакцию
- Ограничения синтаксиса, баги в системах оркестрации

Системы оркестрации

Агент — специальная клиентская программа, получающая задания от сервера.

- Puppet (ruby, нужны агенты)
- Chef (ruby+erlang, нужны агенты)
- Salt (python+sshd, агент не обязателен)
- Ansible (python+sshd, агента нет)

Мы выбрали Ansible.

ИСП РАН

Ansible: терминология

- Inventory — файл, описывающий хосты и группы хостов
- Facts — множество сведений, известных Ansible об управляемом хосте
- Task — одно неделимое задание
- Role — набор заданий (tasks)
- Playbook — Набор ролей или заданий с указанием целевой группы хостов

Ansible: пример inventory

```
[physical_hosts]
node01 host="{{ hosts.node01 }}" ansible_ssh_host="{{ host.ipv4_address }}" docker_eth0_ipv4_address="10.10.10.11"
…
[containers]
cont01 host="{{ hosts.node01 }}" ansible_ssh_host="{{ host.docker_eth0_ipv4_address }}" ansible_ssh_user=root
ansible_ssh_port="{{ ssh_port_to_expose }}"
…
[prepare__hosts]
node[01:08]
[client__hosts]
node08
[client__containers]
cont08
[IMDG_M_hosts]
node[01:02]
[IMDG_M_containers]
cont[01:02]
[IMDG_2M_hosts]
node[01:04]
[IMDG_2M_containers]
cont[01:04]
```

ИСП РАН

Ansible: пример facts

```
node08 | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.10.10.25",
            "172.17.42.1"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::569f:35ff:fe41:a9bf",
            "fe80::5484:7aff:fefe:9799",
            "fe80::569f:35ff:fe41:a9bf",
            "fe80::569f:35ff:fe41:a9bf"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "05/21/2014",
        "ansible_bios_version": "2.2.10",
        "ansible_bond0": {
            "active": true,
            "device": "bond0",
            "ipv4": {
                "address": "10.10.10.25",
                …
```

ИСП РАН

Ansible: пример task

```
tasks:
    - template: src=templates/imdg_ips.j2 dest=/root/imdg_ips
    - script: blobs/client/tester.py load "{{ acc_file }}"
    - script: blobs/client/tester.py save "{{ acc_file }}_new"
    - name: run compare
      script: blobs/client/compare.sh "{{ acc_file }}" "{{ acc_file }}_new"
      register: comp
    - debug: var=comp.stdout_lines
```

Задание начинается с символа “-”.

Ansible: пример role

```
---
- name: push client application to the node
  copy: src=../../../blobs/client/imdgtest-client-1.0-SNAPSHOT.jar dest=/root/imdgtest-client-1.0-SNAPSHOT.jar
- name: push client README.md
  copy: src=../../../blobs/client/README.md dest=/root/README.md
- name: install unzip
  apt: name=unzip
# send data to the client
- unarchive: src=../../../blobs/client/data.zip dest=/root/
# slice acc files
- script: ../../../blobs/client/split.sh {{ item[0] }} {{ item[1] }}
  ignore_errors: yes
  with_nested:
    - acc_files
    - clients_count
# slice op files
- script: ../../../blobs/client/split.sh {{ item[0] }} {{ item[1] }}
  ignore_errors: yes
  with_nested:
    - op_files
    - clients_count
```

ИСП РАН

Ansible: пример playbook

```
---

- hosts: "{{ application_type  }}_{{ nodes_group | default('') }}_hosts"
  sudo: yes
  roles:
    - run

- hosts: "{{ application_type  }}_{{ nodes_group | default('') }}_containers"
  roles:
    - common
    - wildfly
    - ganglia_node_monitoring
```

При вызове можно передавать параметры. В нашем случае это `application_type` и `nodes_group`

Ansible: сильные стороны

- Нужен только Python 2.7+ и openssh-server
- Ansible Facts знает все об управляемых системах
- Проект с очень хорошей декомпозицией подсистем: легко расширять
- Есть Python API
- Есть большой репозиторий готовых рецептов
- Для использования готовых плэйбуков и ролей нужно изменить только один файл
- Роли можно комбинировать

Ansible: слабые стороны

- Пока что есть нелепые баги
  - баг с разворачиванием переменных
  - баг со строковыми переменными
- Неочевидная система циклов
  - нельзя сделать цикл по группам хостов
  - нельзя сделать цикл по ролям
  - нельзя использовать имена для параметров цикла
- API не слишком продвинутое
  - Доступны только два класса: Runner и Playbook
  - Документация на API только в комментариях к коду, примеров нет

Процесс подготовки

1. Создать базовый образ системы для Docker с openssh-server на борту
2. Описать необходимые роли для создания, запуска, остановки, уничтожения контейнеров
3. Описать роли для развертывания нужных компонентов в контейнеры (пакеты, зависимости, файлы, конфигурации, запущенные сервисы)
4. Написать скрипт на Python, который будет запускать нужные роли для нужных групп хостов (необязательный шаг) в нужном порядке и необходимыми параллельными событиями (остановка сетевого интерфейса в контейнерах, падение ноды, добавление ноды)
5. Написать необходимые скрипты для сбора статистики (у нас используется Ganglia + JMXetric + rrdtool)
6. Запустить скрипт тестирования для каждой цели (3 в нашем случае)

Пример проекта Ansible

```
├── blobs
│   ├── client
│   │   ├── README.md
│   │   ├── compare.sh
│   │   ├── imdgtest-client-1.0-SNAPSHOT.jar
│   │   ├── split.sh
│   │   └── tester.py
│   ├── ganglia_node_monitoring
│   │   ├── gmetric4j-1.0.7.jar
│   │   ├── jmxetric-1.0.7.jar
│   │   ├── jmxetric.xml
│   │   └── oncrpc-1.0.7.jar
│   ├── ganglia_web_collector / ganglia_apache2.conf
│   ├── infinispan_embedded
│   │   ├── infini.jar
│   │   ├── infinispan.xml
│   │   ├── jgroups.xml
│   │   └── log4j.xml
│   ├── infinispan_server / infinispan-server-6.0.2.Final-bin.zip
│   ├── initial_configuration / pipework
│   ├── jboss / wildfly-8.1.0.Final.zip
│   ├── rrd-collector / rrdCollector.py
│   └── server / imdgtest-server-1.0-SNAPSHOT.war
├── configuration_files
│   ├── infinispan_server / clustered.xml
│   └── wildfly / standalone.xml
├── group_vars / all
├── hosts
├── library
│   └── cloud
│       ├── docker
│       └── docker_facts
├── roles
│   ├── build / tasks / main.yml
│   ├── client / tasks / main.yml
│   ├── common / tasks / main.yml
│   ├── create_dockerfile / tasks / main.yml
│   ├── ganglia_node_monitoring / tasks / main.yml
│   ├── ganglia_web_collector / tasks / main.yml
│   ├── infinispan_embedded / tasks / main.yml
│   ├── infinispan_server / tasks / main.yml
│   ├── kill / tasks / main.yml
│   ├── prepare / tasks / main.yml
│   ├── run / tasks / main.yml
│   ├── stop / tasks / main.yml
│   └── wildfly / tasks / main.yml
├── configuration_files
│   ├── infinispan_server / clustered.xml
│   └── wildfly / standalone.xml
├── group_vars / all
├── hosts
├── prepare_client.yml
├── prepare_imdg_hosts.yml
├── templates
│   ├── IMDGTEST_PROPS.j2
│   ├── docker_base_system.j2
│   ├── gmetad.j2
│   ├── gmond.conf
│   ├── gmond.j2
│   ├── gmond_master.j2
│   └── imdg_ips.j2
└── test_cycle.py
```

ИСП РАН

Краткие выводы для GridGain

- С увеличением числа узлов производительность падает.
- Отсутствует механизм разрешения тупиков.
- Максимальная производительность в наших конфигурациях для 6 узлов ~3900 операций в секунду без репликации, ~2600 операций в секунду с 2 копиями для каждого ключа.
- Иногда система теряет данные.

Спасибо за внимание!

ИСП РАН