Issue Umbrella

Grupo 04

YARN-4945 - Capacity Scheduler Preemption Within a Queue

Introdução

- Tipo: Umbrella
- Prioridade: Major
- Status: Open
- Relator: Wangda Tan
- Criação: April/2016
- Descrição
    - This is an umbrella ticket to track efforts of preemption within a queue (...)

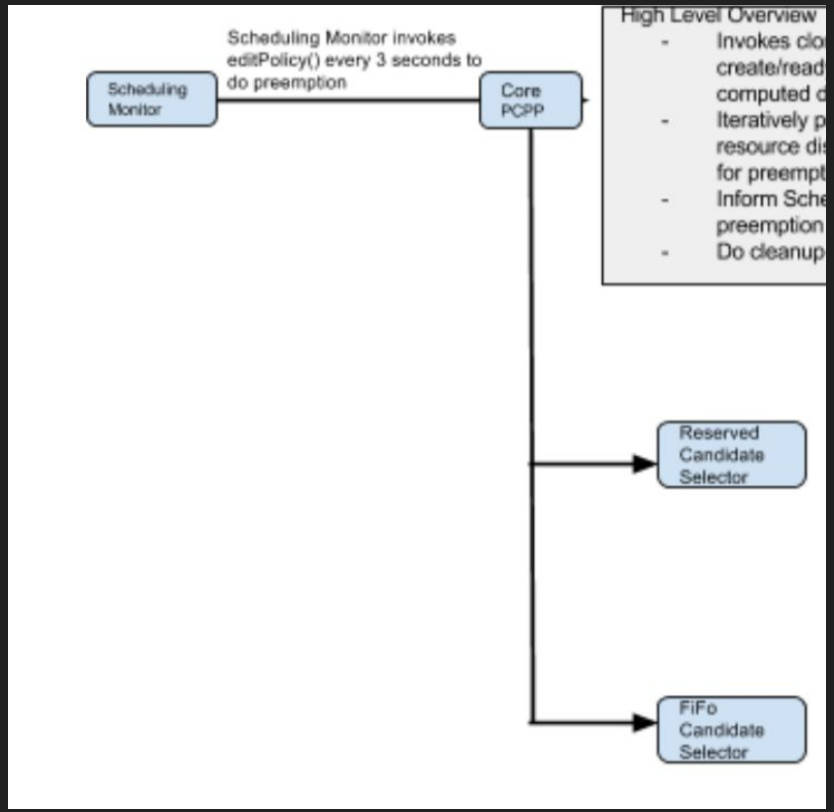
Introdução

- *Capacity Scheduler*
  - Framework do Hadoop cujo o foco é normalizar a distribuição de recursos "preempting" a execução de programas utilizando recursos superutilizados
  - Visa maximizar o throughput e a utilização do cluster distribuindo recursos em filas
- Aplicações usando recursos acima dos limites pré definidos podem levar a subutilização do cluster

Introdução

- Aplicações podem 'passar fome' se outras aplicações da mesma fila tentarem utilizar recursos superutilizados
- Decisão de quais aplicações devem ser suspensas é feita com base em:
  - Prioridade
  - Fairness
  - Limite do usuário

Design anterior



Design anterior

- Scheduling Monitor
  - Invoca o método *editPolicy* a cada 3 segundos para fazer *preemption*
- *Core PCPP*
  - *Itera sobre todos os CandidateSelectors para coletar info a respeito do uso de resources e seleciona possíveis candidatos a preemption*
- *Reserved Candidate Selector*
  - *Calcula o uso ideal de recursos por fila e determina os recursos superutilizados*
  - *Tenta aliviar a demanda de tais recursos liberando containers reservados*
- *FiFo Candidate Selector*
  - *Calcula o uso ideal de recursos por fila e determina os recursos superutilizados*
  - *Tenta aliviar a demanda de tais recursos liberando contêineres submetidos mais recentemente*

Intra-queue preemption

- Foca em normalizar a distribuição de recursos para as aplicações dentro de uma mesma fila
- Intra-queue preemption também tem como objetivo normalizar a distribuição de recursos dentre os usuários
- Quando uma aplicação demanda alto uso de recursos, o módulo tentará obtê-los a partir da suspensão de aplicações de menor prioridade rodando na mesma fila

Novo design

- Over subscribed queue
  - Uma fila que alocou recursos acima de sua capacidade
- Normal running queue
  - Fila rodando dentro de seus limites
  - Todas as as suas aplicações estão abaixo de seus limites predefinidos
- Under served queue
  - Fila que não possui recursos suficientes para servir suas aplicações

Novo design

- Irá prover uma forma de ligar e desligar a política de preemption intra-filas
- Intra-filas deve ocorrer depois da política de inter-filas

Detalhes de implementação

- Cálculo da alocação ideal de recursos para encontrar filas 'under-served'
- Determinar recursos necessários para os apps que mais precisam
- Seleção de candidatos para preemption intra-filas com base em prioridade
  - Iterar em todas as filas a partir das mais 'under-served'
  - Avaliar todos os app ordenados pelo comparador de 'preemption'
  - Com base no mapa *resourceToObtain*, determinar os recursos necessários por contêiner
  - Selecionar conteiners de baixa prioridade para preemption até a demanda de recursos ser satisfeita
  - Retorna essa lista para o módulo PCPP

YARN-4945 - Casos de uso

Casos possíveis intra-queue

- Cenários com prioridade apenas
- Cenários com limite de usuário apenas
- Cenários com prioridade e limite de usuário juntos

Prioridade: Configuração

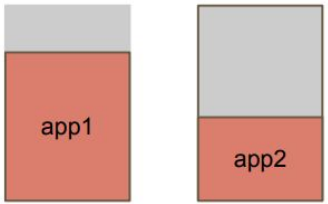
- yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled = true;
- root.qA.capacity = 70%
- root.qB.capacity = 30%
- Cluster resource = 100 (qA.used=70, qB.used=30)
- Rodando na qA

Prioridade: sem preemption

Input:

app1 , p1, u1 <pending=0 , used=50 >

app2 , p1, u1 <pending=20 , used=20>

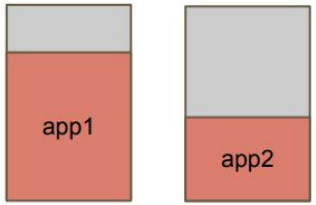


Before

Preempted:

app1 , p1, u1 <preempted=0, pending=0 , used=50>

app2 , p1, u1 <preempted=0, pending=20 , used=20>



After

Prioridade: com preemption

Input:

app1 , p1, u1 <pending=20 , used=50 >

app2 , p1, u1 <pending=20 , used= 20>

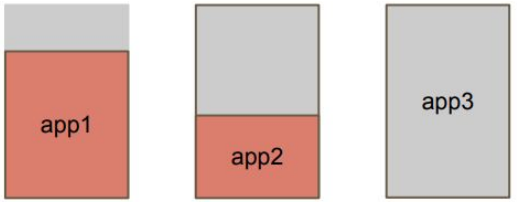
app3 , p3, u1 <pending=30 , used= 0>

Preempted:

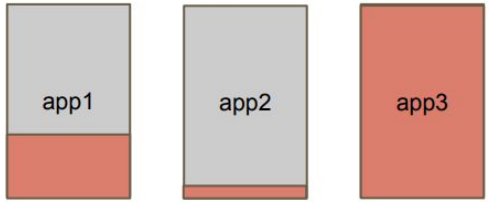
app1 , p1, u1 <preempted=11, pending=31 , used=39>

app2 , p1, u1 <preempted=19, pending=39 , used=1>

app3 , p3, u1 <pending=0 , used=30>



Before



After

Limite: sem preemption

- *yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled = true*
- root.qA.capacity = 100%
- Minimum-user-limit-percent = 50%
- Cluster resource = 100 (qA.used=100)

Limite: sem preemption

Input:

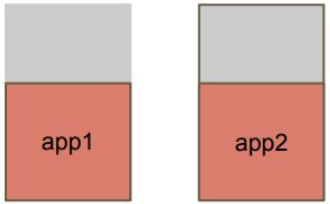
app1 , p1, u1 <pending=20 , used=50 >

app2 , p1, u2 <pending=20 , used= 50>

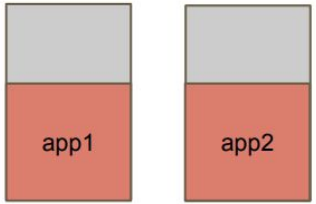
Preempted:

app1 , p1, u1 <preempted=0, pending=20 , used=50>

app2 , p1, u2 <preempted=0, pending=20 , used=50>



Before



After

Limite: com preemption

- *yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled = true*
- root.qA.capacity = 100%
- Minimum-user-limit-percent = 33%
- Cluster resource = 100 (qA.used=100)

Limite: com preemption

Input:

app1 , p1, u1 <pending=20 , used=25 >

app2 , p1, u2 <pending=20 , used= 25>

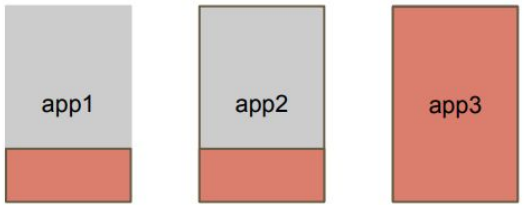
app3 , p1, u3 <pending=30 , used= 50>

Preempted:

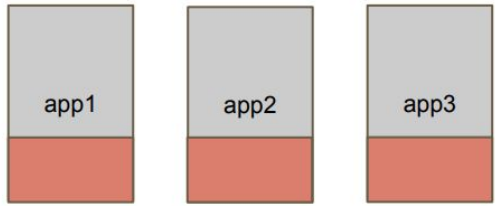
app1 , p1, u1 <preempted=0, pending=12 , used=33>

app2 , p1, u2 <preempted=0, pending=12 , used=33>

app3 , p1, u3 <preempted=16, pending=46 , used=34>



Before



After

Prioridade e Limite: configuração

- *yarn.resourcemanager.monitor.capacity.preemption.intra-queue-preemption.enabled = true*
- root.qA.capacity = 100%
- Minimum-user-limit-percent = 50%
- Cluster resource = 100 (qA.used=100)

Prioridade e Limite: com preemption

Input:

app1 , p1, u1 <pending=20 , used=20 >

app2 , p2, u1 <pending=20 , used= 20>

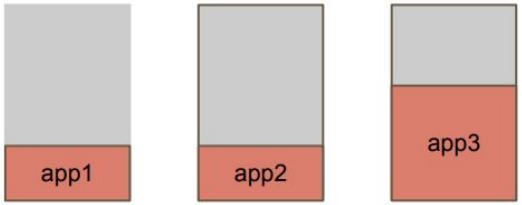
app3 , p3, u2 <pending=40 , used= 60>

Preempted:

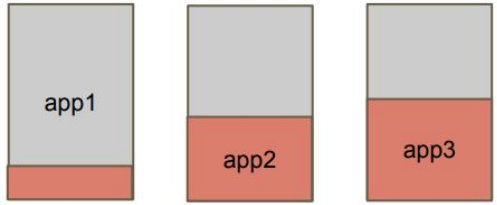
app1 , p1, u1 <preempted=10, pending=30, used=10>

app2 , p2, u1 <preempted=0, pending=0 , used=40>

app3 , p3, u2 <preempted=10, pending=50 , used=50>



Before



After

YARN-4945 - Patch

Overview

- Implementa Preemption por Prioridade por Apps
- Modifica a *ProportinalCapacityPreemptionPolicy* para selecionar contêineres com baixa prioridade para ser suspendido
- Finalizada em Set/2016

Setup

```
@Before
public void setup() {
  super.setup();
  policy = new ProportionalCapacityPreemptionPolicy(rmContext, cs, mClock);
}
```

Arquitetura

```
@Test
public void testSimpleIntraQueuePreemption() throws IOException {
  String labelsConfig =
      "=100,true;";
  String nodesConfig = // n1 / n2 has no label
      "n1= res=100";
  String queuesConfig =
      // guaranteed,max,used,pending,reserved
      "root(=[100 100 80 120 0]);" + //root
          "-a(=[11 100 10 50 0]);" + // a
          "-b(=[40 100 40 60 0]);" + // b
          "-c(=[20 100 10 10 0]);" + // c
          "-d(=[29 100 20 0 0])"; // d
```

Aplicativos

```
String appsConfig=
    "a\t" // app1 in a
    + "(1,1,n1,,6,false,25);" + // app1 a
    "a\t" // app2 in a
    + "(1,1,n1,,5,false,25);" + // app2 a
    "b\t" // app3 in b
    + "(4,1,n1,,34,false,20);" + // app3 b
    "b\t" // app4 in b
    + "(4,1,n1,,2,false,10);" + // app4 b
    "b\t" // app4 in b
    + "(5,1,n1,,1,false,20);" + // app5 b
    "b\t" // app4 in b
    + "(6,1,n1,,1,false,10);" + // app6 in b
    "c\t" // app1 in a
    + "(1,1,n1,,10,false,10);" +
    "d\t" // app1 in a
    + "(1,1,n1,,20,false,0)";

buildEnv(labelsConfig, nodesConfig, queuesConfig, appsConfig);
policy.editSchedule();
```

Teste

```
  buildEnv(labelsConfig, nodesConfig, queuesConfig, appsConfig);
  policy.editSchedule();

  verify(mDisp, times(1)).handle(argThat(
      new TestProportionalCapacityPreemptionPolicy.IsPreemptionRequestFor(
          getAppAttemptId(4))));
  verify(mDisp, times(19)).handle(argThat(
      new TestProportionalCapacityPreemptionPolicy.IsPreemptionRequestFor(
          getAppAttemptId(3))));
}
```

Comentários finais

- Dependência de diversos outros módulos do sistema
- Complexidade da feature
- Falta de clareza em comentários/barreira idiomática