Архитектура балансувальника навантаження для Nordugrid ARC

Свірін П.В.

Національний технічний університет України "Київський політехнічний інститут", пр. Перемоги, 37, Київ, 03056, Україна

paul.svirin@gmail.com

Анотація. Для забезпечення вимог користувачів з продуктивності та ефективності виконання завдань грід - система повинна реалізувати ефективний алгоритм розподілу завдань між доступними на даний час обчислювальними ресурсами. Основна мета такого балансування навантаження в грід-системі - скоротити час виконання завдання користувача і забезпечить ефективність використання обчислювальних ресурсів для виключення ситуації, коли одні ресурси простоюють, а інші перевантажені виконанням завдань користувачів.

Доповідь присвячена питанням побудови брокерів для грід-середовища на рівні метапланувальника.

Ключові слова

Розподілені обчислення, грід, Nordugrid Arc, балансування навантаження

1 Вступ

Платформа Nordugrid ARC[1] – реалізація так званого "middleware" - посередника між клієнтом та програмним забезпеченням обчислювального ресурсу. Суть цього шару полягає в тому, що middleware надає єдиний інтерфейс для подачі завдань на виконання і контролю за виконанням, отже для клієнтської сторони не є необхідним реалізовувати всі можливі інтерфейси обчислювальних вузлів.

В Nordugrid ARC є можливість подачі завдання напряму на обчислювальний елемент, що не завжди є ефективним, оскільки в момент подачі невідома завантаженість цього елементу, отже, більш ефективним рішенням є використання брокерів – програмних компонентів, що перенаправляють завдання на обчислювальний елемент, на якому він виконається найшвидше і з найменшим навантаженням для системи.

2 Архітектура брокера в в Nordugrid ARC

Брокер[3] – це програмний модуль, що визначає найбільш відповідний ресурс, на якому буде можливо запустити деяку задачу. В системі Nordugrid ARC даний програмний модуль знаходиться на клієнтському боці і аналізує дані, отримані з інформаційної системи.

Модуль брокера для ARC складається з двох типів класів:

- базовий клас брокера, який реалізує операцію мінімізації списка знайдених кандидатів для виконання завдання (множини об'єктів класу ExecutionTarget).
- спеціалізовані брокери, що реалізують інтерфейс базового класу згідно необхідного алгоритму

Пошук ресурсів, ща здатні виконати завдання проводить об'єкт класу TargetGenerator, який повертає список ресурсів-кандидатів. Метод PreFilterTargets проводить співставлення і порівняння об'єктів у

списку для видалення дублікатів або ресурсів з неповною відповідністю для виконання завдання або неповною інформацією, опублікованою даним ресурсом. Після проведення фільтрації списку ресурси мають бути сортовані у порядку, в якому найбільш відповідний для виконання завдання ресурс ставиться на початок. Алгоритми сортування реалізуються у класах, що наслідуються від базового класу брокера (клас Broker), а саме в методі SortTargets класів-наслідників.

Брокери, реалізовані в Nordugrid ARC

- Random – проводить сортування кандидатів випадковим чином;
- Benchmark – проводить сортування згідно даних оцінки продуктивності ресурсі, яка задається у вигляді строки в налаштуваннях кластеру. При відсутності показника продуктивності у налаштуваннях ресурса використовується строка CINT2000.
- FastestQueueBroker – сортує список кандидатів в залежності від коефіцієнту заповненості черги ресурсу (наприклад, довжина черги, поділена на загальну кількість процесорів). При наявності в більш ніж одного кандидата черги довжиною 0 даний брокер запускає базовий алгоритм балансування навантаження.
- Data – даний брокер сортує ресурси згідно об'єму даних, що вже зберігаються в кешу обчислювального ресурсу, який представляє ExecutionTarget. Загальна ідея даного брокера: завдання мають подаватись на ті ресурси, на яких необхідні дані вже знаходяться в кешу, таким чином, зменшуючи навантаження на мережу. Для отримання інформації про кеш використовується інтерфейс CacheCheck для сервісу A-REX.

Приклад запиту CacheCheck для сервісу A-REX:

```
<CacheCheck>
<TheseFilesNeedToCheck>
<FileURL>http://knowarc1.grid.niif.hu/storage/Makefile</FileURL>
<FileURL>ftp://download.nordugrid.org/test/README</FileURL>
</TheseFilesNeedToCheck>
</CacheCheck>
```

Приклад відповіді на запит CacheCheck для сервісу A-REX:

```
<CacheCheck> <TheseFilesNeedToCheck>
<FileURL>http://knowarc1.grid.niif.hu/storage/Makefile</FileURL>
<FileURL>ftp://download.nordugrid.org/test/README</FileURL> </TheseFilesNeedToCheck>
</CacheCheck>
<CacheCheckResponse> <CacheCheckResult> <Result>
<FileURL>http://knowarc1.grid.niif.hu/storage/Makefile</FileURL> <ExistInTheCache>true</ExistInTheCac
</Result> <Result>
<FileURL>ftp://download.nordugrid.org/test/README</FileURL> <ExistInTheCache>true</ExistInTheCache>
<FileSize>176</FileSize> </Result>
</CacheCheckResult> </CacheCheckResponse>
```

Реалізація власного брокера в системі ARC

Для побудови власного брокера треба розробити клас-наслідник від Arc::Broker. Наведемо приклад декларації класу, побудований на основі класу Arc::RandomBroker. Файли, в яких знаходяться реалізації брокерів знаходяться в директорії $ARC_ROOT/src/hed/acc/Broker/ , декларація та реалізація класу Arc::Broker — в $ARC_ROOT/src/hed/libs/client

```
#ifndef __ARC_MyBroker_H__
#define __ARC_MyBroker_H__

#include <arc/client/Broker.h>

namespace Arc {

  class MyBroker
```

```
    : public Broker {
  public:
    MyBroker(const UserConfig& usercfg);
    ~MyBroker();
    static Plugin* Instance(PluginArgument *arg);

  protected:
    virtual void SortTargets();
  };

} // namespace Arc

#endif // __ARC_MyBroker_H__
```

Методом класу, в якому проходить сортування цільових обчислювальних ресурсів у порядку спадання відносно їх відповідності вимогам запиту, є SortTargets. Список PossibleTargets, яким маніпулює цей метод об'явлений в класі Arc::Broker і передається в дочірній клас шляхом наслідування. Після завершення сортування слід виставити TargetSortingDone = true; , що показує те, що сортування завершено.

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <cstdlib>
#include <algorithm>

#include <arc/client/ExecutionTarget.h>

#include "MyBroker.h"

namespace Arc {

  MyBroker::MyBroker(const UserConfig& usercfg)
    : Broker(usercfg) {}

  MyBroker::~MyBroker() {}

  Plugin* MyBroker::Instance(PluginArgument *arg) {
    BrokerPluginArgument *brokerarg = dynamic_cast<BrokerPluginArgument*>(arg);
    if (!brokerarg)
      return NULL;
    return new MyBroker(*brokerarg);
  }

  void MyBroker::SortTargets() {

    std::list<ExecutionTarget*>::iterator iter = PossibleTargets.begin();

    logger.msg(VERBOSE, "Matching against job description, following targets possible for MyBroker: %

    for (int i = 1; iter != PossibleTargets.end(); iter++, i++)
      logger.msg(VERBOSE, "%d. Cluster: %s; Queue: %s", i, (*iter)->DomainName, (*iter)->ComputingSha

    int i, j;
    std::srand(time(NULL));

    for (unsigned int k = 1; k < 2 * (std::rand() % PossibleTargets.size()) + 1; k++) {
      std::list<ExecutionTarget*>::iterator itI = PossibleTargets.begin();
      std::list<ExecutionTarget*>::iterator itJ = PossibleTargets.begin();
      for (int i = rand() % PossibleTargets.size(); i > 0; i--)
        itI++;
```

```
      for (int i = rand() % PossibleTargets.size(); i > 0; i--)
        itJ++;
      std::iter_swap(itI, itJ);
    }

    logger.msg(VERBOSE, "Best targets are: %d", PossibleTargets.size());

    iter = PossibleTargets.begin();

    for (int i = 1; iter != PossibleTargets.end(); iter++, i++)
      logger.msg(VERBOSE, "%d. Cluster: %s; Queue: %s", i, (*iter)->DomainName, (*iter)->ComputingSha

    TargetSortingDone = true;
  }

} // namespace Arc
```

Останній крок: внести опис брокера в масив PLUGINS_TABLE_NAME,що знаходиться у файлі DescriptorsBroker.cpp:

```
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include "FastestQueueBroker.h"
#include "RandomBroker.h"
#include "BenchmarkBroker.h"
#include "DataBroker.h"

Arc::PluginDescriptor PLUGINS_TABLE_NAME[] = {
  { "FastestQueue", "HED:Broker", 0, &Arc::FastestQueueBroker::Instance },
  { "Random", "HED:Broker", 0, &Arc::RandomBroker::Instance },
  { "Benchmark", "HED:Broker", 0, &Arc::BenchmarkBroker::Instance },
  { "Data", "HED:Broker", 0, &Arc::DataBroker::Instance },
  { NULL, NULL, 0, NULL }
};
```

В якості алгоритму сортування в українському сегменті (приблизно 20 кластерів) [5][6] достатньо використати стандартну реалізацію алгоритму швидкого сортування std::qsort зі стандартних бібліотек C++ [2].

На даний момент модуль балансування в Nordugrid ARC використовує виключно дані щодо кластерів, отримані з інформаційної системи ARC Infosys. Але в ARC 1.0 введено можливість розробки сервісів, що запускаються в цій платформі. Таким чином, можливе отримання додаткових відомостей щодо обчислювальної системи кластеру (наприклад, стану окремих вузлів в середині самого кластеру) для більш ефективної політики балансування.

На разі в українському сегменті використовується алгоритм випадкового вибору, що є статичним, тобто не враховує поточного стану обчислювального ресурсу. Отже, потрібні алгорими з більшими можливостями. Для ефективного розподілу завдань серед кластерів слід :

* визначити набір характеристик, за якими буде проводитись оцінка завантаженості обчислювального кластеру;
* розробити порядок кроків для ранжування списку кластерів та відсіювання тих кластерів, що не підходять під вимоги завдання, для якого ведеться пошук кластеру призначення.

3 Висновки

Серед алгоритмів балансування навантаження ресурсів грід-системи можна розрізнити статичні і динамічні в залежності від параметрів, що враховуються під час прийняття рішення про призначення і

виконання завдання. В статичному підході використовується середньостатистична інформація про роботу системи, ігноруючи її поточний стан. Статичні алгоритми частіше застосовуються для балансування періодичних завдань з жорсткими вимогами щодо часу завершення. Напроти, динамічні алгоритми виконують балансування в реальному часі і динамічно визначають можливість поліпшення балансування навантаження з врахуванням вже призначених для оброблення і розподілених задач. Оскільки серед задач в грід-середовищі періодичними є меншістю, більш доцільно використовувати динамічні алгоритми. Задача розподілу ресурсів пов'язана не лише з вибором механізму розподілу, але, можливо, більше визначається інформацією, доступною для брокера, і її актуальністю. Підтримка оновлення інформації про стан ресурсів може бути досить дорогою і вартість її зростає разом із розміром системи. Чим краще управління інформаційними потоками в грід- системі, тим ефективніший процес розподілу ресурсів. Це положення можуть бути використані для суттєвого поліпшення брокерських можливостей нового покоління проміжного забезпечення NorduGrid ARC.

Література

[1] http://www.nordugrid.org

[2] Comparison of several sorting algorithms.http://warp.povusers.org/SortComparison/

[3] Abhishek Singh. Grid Resource Broker. http://www.cse.buffalo.edu/faculty/miller/Courses/Grid-Seminar/GridResourceBroker.pdf

[4] http://gstat.egi.eu/gstat/ldap

[5] А.Загородний, Г. Зиновьев, Е. Мартынов, С. Свистунов - Украинский академический грид : Українсько-македонський науковий збірник , Випуск 4 ,Київ 2009, Вид.Национальна бібліотека України імені В.І.Вернадського, с.140-150.

[6] Петренко А.І. Національна Grid - інфраструктура для забезпечення наукових досліджень і освіти. http://netallted.cad.kiev.ua/downloads/Grid.pdf

[7] В. С. Пономаренко, С. В. Листровой, С. В. Минухин, С. В. Знахур. Методы и модели планирования ресурсов в GRID-системах // ИНЖЭК - 2008 – 408 стор.