GB-500: Introdução a Workflows Científicos e suas Aplicações

Professores: Luiz Gadelha, Kary Ocaña

Programa de Verão do LNCC, 2017
Laboratório Nacional de Computação Científica

18 de abril de 2018

Histórico do Swift

- Versões do Swift:
  - Swift/K (2007):
    - workflows paralelos e distribuídos.
    - http://swift-lang.org.
  - Swift/T (2013):
    - workflows para HPC (p.ex. máquinas Top 500).
    - http://swift-lang.org/Swift-T.
  - PARSL (2015):
    - biblioteca do Python que incorpora ambiente de execução do Swift.
    - workflow passa a ser especificado em Python.
    - http://parsl-project.org/.

Armstrong, T. et al. (2015). Swift: Extreme-scale, Implicitly Parallel Scripting. In P. Balaji (Ed.), Programming Models for Parallel Computing (pp. 219–245). MIT Press.
https://www.researchgate.net/publication/284444451_Swift_Extreme-scale_Implicitly_Parallel_Scripting.

Swift/T

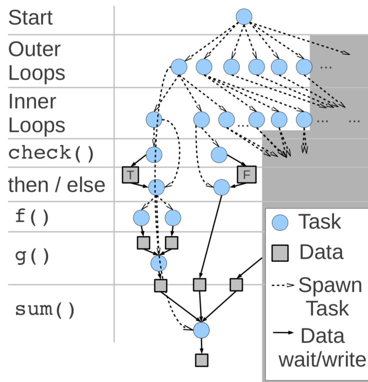
- Utiliza balanceamento de carga para avaliação/execução do workflow.
- O Swift/T estende o modelo de *dataflow* do Swift/K com sincronização via troca de arquivos para aplicações com granularidade mais fina usando funções e dados em memória.
- Escalabilidade já testada até 128.000 *cores*.

Wozniak, J. M. et al. (2013). Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. In 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (pp. 95–102). IEEE. https://doi.org/10.1109/CCGrid.2013.99.

Lusk, E., Butler, R., Pieper, S. C. (2018). Evolution of a minimal parallel programming model. The International Journal of High Performance Computing Applications, 32(1), 4–13. https://doi.org/10.1177/1094342017703448.

Swift/T

```
 1  int X = 1000, Y = 1000;
 2  int A[][];
 3  int B[];
 4  foreach x in [0:X-1] {
 5  foreach y in [0:Y-1] {
 6    if (check(x, y)) {
 7     A[x][y] = g(f(x),f(y));
 8    } else {
 9     A[x][y] = 0;
10    }}
11  B[x] = sum(A[x]);
12  }
```



Wozniak, J. M. et al. (2013). Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. In 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (pp. 95–102). IEEE. https://doi.org/10.1109/CCGrid.2013.99.

Swift/T

- A avaliação do laço externo é distribuída para diversos processadores.
- Similarmente, a avaliação do laço interno é distribuída para diversos processadores.
- No Swift/K a avaliação do script do workflow era centralizada, causando gargalos de execução em casos com um número muito grande de tarefas de curta duração.

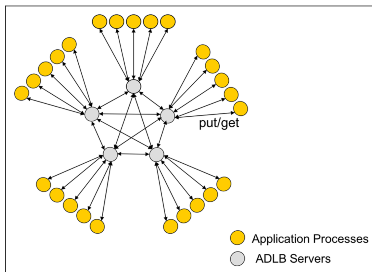
Wozniak, J. M. et al. (2013). Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. In 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (pp. 95–102). IEEE. https://doi.org/10.1109/CCGrid.2013.99.

Swift/T

- Workflows adequados para o Swift/T:
  - Dependências de dados e de coordenação não-triviais (avaliação dinâmica de expressões funcionais com paralelismo implícito).
  - Estrutura computacional irregular ou difícil de prever (balanceamento de carga dinâmico).
  - Orquestração em larga escala de funções de bibliotecas ou programas executáveis.

Swift/T: Ambiente de execução

- Ambiente de execução baseado no *Asynchronous Dynamic Load Balancer* (ADLB).
- ADLB é uma biblioteca MPI para distribuição de tarefas para nós "trabalhadores".
- Filas que proveem tarefas para os nós trabalhadores são implementadas de forma distribuída para obter escalabilidade.



Lusk, E., Butler, R., Pieper, S. C. (2018). Evolution of a minimal parallel programming model. The International Journal of High Performance Computing Applications, 32(1), 4–13. https://doi.org/10.1177/1094342017703448.

Laboratório Nacional de Computação Científica

PARSL

- Parsl é uma biblioteca para o Python que permite gerenciar workflows científicos paralelos.
- O paralelismo é implícito e baseado nas dependências das aplicações componentes do workflow.
- O motor de execução inclui diversas opções de execução, desde multithreading local a execução em supercomputadores com Swift/T.

`http://parsl-project.org/.`

Exemplo de workflow PARSL

```
@App('python', dfk)
def pi(total):
    import random
    width = 10000
    center = width/2
    c2  = center**2
    count = 0
    for i in range(total):
        # Drop a random point in the box.
        x,y = random.randint(1, width),random.randint(1, width)
        # Count points within the circle
        if (x-center)**2 + (y-center)**2 < c2:
            count += 1
    return (count*4/total)
```

Exemplo de workflow PARSL

```
@App('python', dfk)
def mysum(a,b,c):
    return (a+b+c)/3

a, b, c = pi(10**6), pi(10**6), pi(10**6)
avg_pi  = mysum(a, b, c)
```

Common Workflow Language (CWL)

- A *Common Workflow Language* (CWL) é uma linguagem padronizada para especificação de workflows.
- Análoga ao OPM e ao PROV para descrição de proveniência.
- Permite:
  - declarar ferramentas de linha de comando (programas) e especificar como estes são executados;
  - definir execuções encadeadas destes programas.
- Suporte em SGWfCs como Galaxy e Taverna.

`http://www.commonwl.org`

Common Workflow Language (CWL)

Exemplo, arquivo `1st-tool.cwl`:

```
cwlVersion: cwl:draft-3
class: CommandLineTool
baseCommand: echo
inputs:
  - id: message
    type: string
    inputBinding:
      position: 1
outputs: []
```

Definição do comando `echo` **do Linux (Fonte:** `http://www.commonwl.org`**)**

Common Workflow Language (CWL)

Exemplo, arquivo `echo-job.yml`:

```
message: Hello world!
```

Definição de entrada para o comando `echo` **(Fonte**: `http://www.commonwl.org`**)**

Execução:

```
$ cwl-runner 1st-tool.cwl echo-job.yml
[job 140199012414352] $ echo 'Hello world!'
Hello world!
Final process status is success
```

CWL: Definição de Workflow

```
cwlVersion: cwl:draft-3
class: Workflow
inputs:
  - id: inp
    type: File
outputs:
  - id: classout
    type: File
    source: "#compile/classfile"
steps:
  - id: untar
    run: tar-param.cwl
    inputs:
      - id: tarfile
        source: "#inp"
    outputs:
      - id: example_out
  - id: compile
    run: arguments.cwl
    inputs:
      - id: src
        source: "#untar/example_out"
    outputs:
      - id: classfile
```

CWL: Execução de Workflow

```
$ cwl-runner 1st-workflow.cwl 1st-workflow-job.yml
[job untar] /tmp/tmp94qFiM$ tar xf /home/example/hello.tar Hello.java
[step untar] completion status is success
[job compile] /tmp/tmpu1iaKL$ docker run -i --volume=/tmp/tmp94qFiM/Hello.java:/var/lib
-volume=/tmp/tmpu1iaKL:/var/spool/cwl:rw --volume=/tmp/tmpfZnNdR:/tmp:rw --workdir=/var
-read-only=true --net=none --user=1001 --rm --env=TMPDIR=/tmp java:7 javac -d /var/spoo
[step compile] completion status is success
[workflow 1st-workflow.cwl] outdir is /home/example
Final process status is success
{
  "classout": {
    "path": "/home/example/Hello.class",
    "checksum": "sha1$2f7ac33c1f3aac3f1fec7b936b6562422c85b38a",
    "class": "File",
    "size": 416
  }
}
```

Python Toil

- Toil é uma biblioteca para especificação e execução de workflows para Python.
- Os workflows são especificados em CWL ou no próprio script Python.
- Permite a execução de scripts em infraestruturas como Amazon AWS, Microsoft Azure, nuvens Open Stack e Google Compute Engine.
- Instalação: `$ pip install toil`

`https://toil.readthedocs.io`

CWL: Execução de Workflow

```
from toil.job import Job

def helloWorld(message, memory="2G", cores=2, disk="3G"):
    return "Hello, world!, here's a message: %s" % message

j = Job.wrapFn(helloWorld, "You did it!")

if __name__=="__main__":
    parser = Job.Runner.getDefaultArgumentParser()
    options = parser.parse_args()
    print Job.Runner.startToil(j, options) #Prints Hello, world!, ...
```

Execução:

```
$ python HelloWorld.py file:my-job-store
```

Prática com Swift

- Usando o Swift na SDumont.
- Tutorial:

  `http://swift-lang.org/swift-tutorial/doc/tutorial.html`

- Estudo de caso de filogenia: SwiftPhylo.
- Estudo de caso de geoprocessamento: MODIS.
- Exercício: criptografia de arquivos em paralelo com OpenSSL.

Usando o Swift na SDumont

```
$ module load swift/0.96.2
$ swift -version
Swift 0.96.2 git-rev: ... heads/release-0.96-swift 6281
```

```
$ swift -sites cpu_dev -config swift.conf modis.swift
$ swift -sites cpu,mesca2 -config swift.conf modis.swift
```

Estudo de caso de filogenia: SwiftPhylo

- Entrada: arquivos com sequências de organismos.
- Saída: árvore filogenética.

```
$ wget https://github.com/stamatak/standard-RAxML/archive/master.zip
...
$ wget https://mafft.cbrc.jp/alignment/software/mafft-7.390-linux.tgz
...
$ wget https://github.com/mmondelli/swift-phylo/archive/master.zip
$ unzip master.zip
$ rm master.zip
$ cd swift-phylo-master/
$ export SWIFT_PHYLO=$(pwd)/bin
$ export PATH=$SWIFT_PHYLO:$PATH
$ swift -config swift.conf swiftPhylo.swift
```

Estudo de caso de geoprocessamento: MODIS

- Entrada: imagens do satélite MODIS.
- Saída: imagem com maior uso de terra urbano.

```
$ wget https://github.com/swift-lang/apps/archive/master.zip
$ unzip master.zip
$ rm master.zip
$ cd apps-master/MODIS/
$ rm swift.conf
$ mkdir modis-2002
$ cd modis-2002
$ wget http://www.lncc.br/~lgadelha/modis-data.tbz
$ tar xvfj modis-data.tbz
$ cd ..
$ swift modis.swift
```

Exercício: criptografia de arquivos em paralelo

- Entrada: arquivos do MODIS (`modis-2002/`).
- Saída: arquivos do MODIS criptografados com o algoritmo AES (256 bits).
- Utilizar o programa `openssl`:

```
openssl enc -e -aes-256-cbc -k 123456 \
            -in h35v10.modis -out h35v10.aes
```