Static Analysis of Cloud Elasticity, Compositionally

Cosimo Laneve
cosimo.laneve@unibo.it

Envisage Year 2 Project Review
Brussels, Dec. 1, 2015

http://www.envisage-project.eu
our line of work

in the cloud

- virtual machines
- bandwidth
- memories
- ... 

cost money

guessing at static time how many VM will be used over time

- consume only what is necessary
- only when it is necessary
computing precise upper-bounds of VM usages in a concurrent oo language (**ABS**) with

* VM acquire
* VM release

these operations are **not exotic**

**Amazon Elastic Compute Cloud** has the Action API:

- **RunInstances** that creates VM
- **TerminateInstance** that terminates a VM — it is idempotent —

**Docker Fiware** — for managing virtual containers —

- **container creation**
  
  $ docker run VM 3a09b2588478
  $ docker start 3a09b2588478

- **container release**

  $ docker stop 3a09b2588478
  $ docker rm 3a09b2588478
studying a connection between ABS codes and service metric functions

- these functions formalize the performance quality constrained by SLA documents

- we focus on capacity, which is the maximum amount of VMs used by a service
computing VM usages upper-bounds

DIFFICULTIES 1/3: peak and net costs

```cpp
int m_one() {
    fut<int> x; x = this!foo();
    fut<int> y; y = this!gee();
    int u = x.get;
    int v = y.get;
    return u+v;
}
```

```cpp
int m_two() {
    fut<int> x; x = this!foo();
    int u = x.get;
    fut<int> y; y = this!gee();
    int v = y.get;
    return u+v;
}
```

assume that

* `foo` acquires 3 VMs and releases them upon termination
* `gee` acquires 4 VMs without releasing them

- `m_one` has peak cost 7
- `m_two` has peak cost 4
- Both have net cost 4
DIFFICULTIES 2/3: method’s effects

- the release in free must be recorded
- methods have effects R: the set of formal parameters that are released
- the cost of methods may be negative

```c
Int free(VM x) { release x; return 0; }
```

free has net cost -1

if x has not been already released...
computing VM usages upper-bounds

DIFFICULTIES 3/3: VM identity

```
Int double_free(VM x, VM y) { release x; release y; return 0; }
```

```
Int user_one() {
  VM x; VM y; Fut<Int> f;
  x = new VM(); y = new VM();
  f = this!double_free(x, y);
  Int u = f.get;
  return 0;
}
```

```
Int user_two() {
  VM x; Fut<Int> f;
  x = new VM();
  f = this!double_free(x, x);
  Int u = f.get;
  return 0;
}
```

- the net cost of `double_free` in `user_one` is -2
- the net cost of `double_free` in `user_two` is -1

The cost of `double_free` depends on the identity of the arguments.
our approach

ABS code

SLA metric function

compliance

deadlocks

time

resources

inference

abstract behaviours

- contracts -

compliance

inference

analysis

ABS code

SLA metric function

compliance

deadlocks

time

resources

inference

analysis

abstract behaviours

- contracts -

compliance

deadlocks

time

resources

inference

analysis

abstract behaviours

- contracts -

compliance

deadlocks

time

resources

inference

analysis

abstract behaviours

- contracts -

compliance

deadlocks

time

resources

inference

analysis

abstract behaviours

- contracts -
inference: effects and dead-code stripping

```plaintext
Int double_free(VM x, VM y) {
    release x;
    release y;
    return 0;
}
```

double_free(THISVM, X, Y) {
    X.release;
    Y.release;
} Int, {X, Y}

test(VM x, VM y) {
    Fut<Int> f, g;
    VM z = new VM();
    f = this!double_free(y, z);
    Int u = f.get;
    g = y!create100VM();
    Int v = g.get;
    release z;
    return u + v;
}

test(THISVM, X, Y) {
    new Z;
    new f: double_free(THISVM, Y, Z);
    f.get;
    X.release;
} Int, {X, Y}

invocations that are never performed are not highlighted in the abstract behaviour
inference: conditionals

```java
Int DoWork(Int n) {
    if (n == 0) return(0);
    else {
        VM x; Fut<Int> f, g;
        x = new VM();
        f = x!SimpleWork();
        g = this!DoWork(n-1);
        Int u = f.get;
        release x;
        Int v = g.get;
        return u+v;
    }
}
```

```java
DoWork(THISVM, n){
    (n = 0) { 0 }
    + (n != 0) {
        new X;
        new f: SimpleWork(X);
        new g: DoWork(THISVM, n-1);
        f.get;
        X.release;
        g.get;
    }
} -,{ }
```

conditionals are not abstracted when guards are in presburger arithmetics
SLA metric functions and behavioural interfaces

service metric function:

\[ VM_{\text{Upper\_Bound}}_{\text{DoWork}}(n) \leq n \]

```java
DoWork(T HISVM, n){
    (n = 0) { 0 }
    + (n != 0) { new X ;
    new f: SimpleWork(X) ;
    new g: DoWork(T HISVM,n-1);
    f.get ;
    X.release ;
    g.get ;
    }
} -=,{
}
```

we **demonstrate the correctness** by

- proving (by hand) by using some logics (using **KeYABS**)?
- translating the equations into cost equations and using the cost equation solver (using **SACO** or **CoFloCo**)
service metric function:

\[ VM_{Upper\_Bound}(n, m) \leq \begin{cases} 
2n & \text{if } n \leq m \\
\text{n+m} & \text{if } n > m 
\end{cases} \]

```java
betterThanAmortized(THISVM, N, M, X) {
    (N = 0) {} 
    + (N > 0) {
        (N > M) {
            new V; 
            new F: betterThanAmortized(X, N-1, M, V); 
            F.get; 
        }
        + (N <= M) {
            new V; new W; 
            new F: betterThanAmortized(V, N-1, M, W); 
            F.get; 
        }
    }
} Int, { }
```
pros

a modular technique

- the analysis is defined on abstract behaviours
- the analysis can be re-defined to target different solvers (KeYABS?)

a compositional technique

- methods may be analysed separately
- the abstract behaviours provide a formal connection with SLA metric functions
the inference of abstract behaviour may be not precise

- the abstraction process loses precision on dynamic data types
- there is a trade-off between precision and expressivity if one wants to use automatic solvers
what we achieved

- we have developed **static analysis techniques** that compute **upper bounds of VM usages** in ABS

- we have studied the compliance with SLA metric functions
publications and plans

A. Garcia, C. Laneve, M. Lienhardt: **Static Analysis of Cloud Elasticity.** In PPDP 2015, pp 125-136, ACM.

E. Giachino, S. de Gouw, C. Laneve, B. Nobakht: **Statically and Dynamically Verifiable SLA metrics.** 2015, submitted
— at cs.unibo.it/~laneve/papers/SLA-metrics.pdf

- we are extending the inference to cover full **ABS**
- we are investigating semi-automatic compliance demonstrations by using **KeYABS**
demo at

http://ei.abs-models.org:8082/clients/web/
QUESTIONS