Genetic Algorithm-based Optimization of Service Composition and Deployment

Yves Vanrompay
Peter Rigole
Yolande Berbers

SIPE08 Workshop

Overview

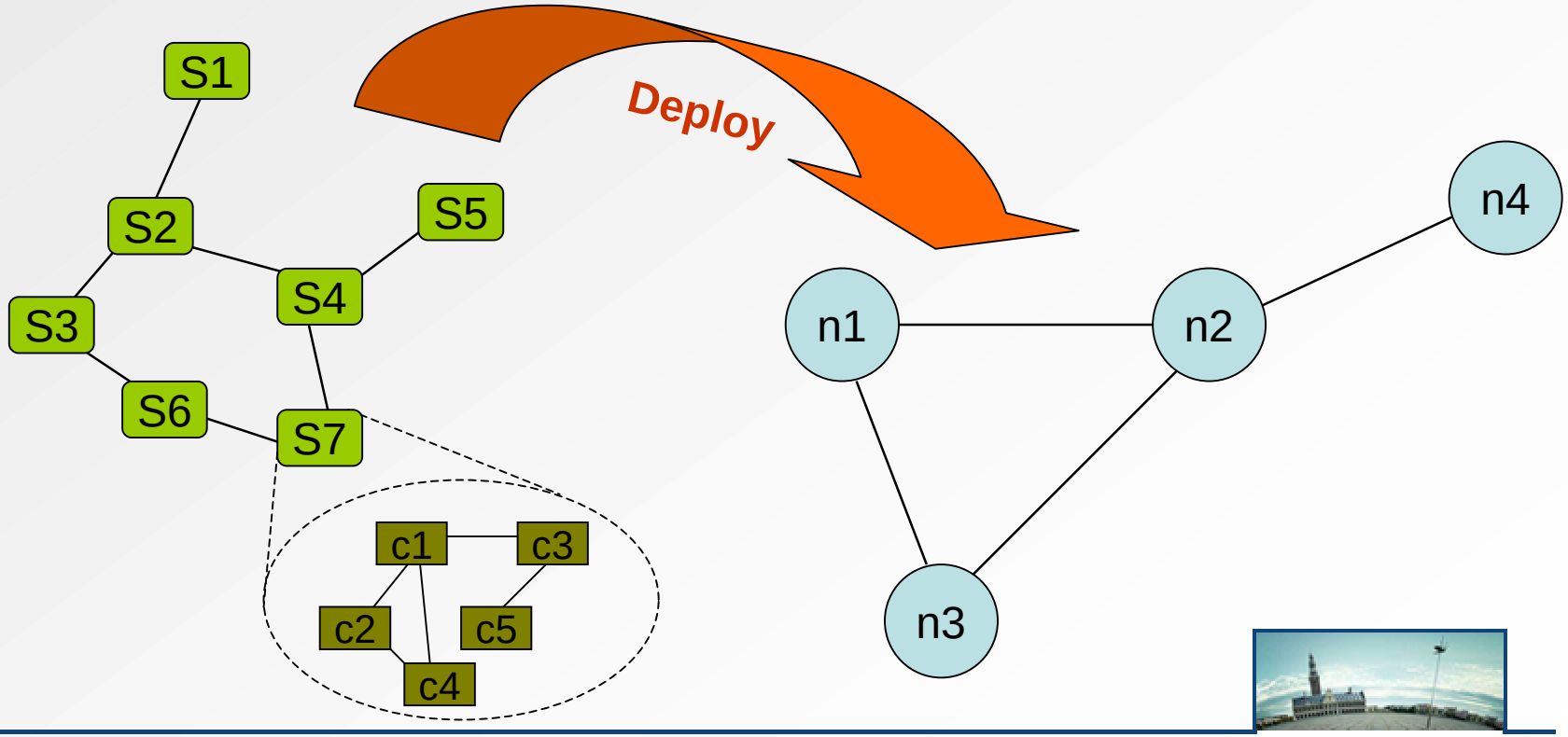
1. Introduction
2. Requirements and motivating application
3. Genetic Algorithms
4. Service composition and deployment
5. Conclusions

Introduction

- Mobile systems
  - Sense and react to context information
  - To adapt to changing user needs and availability of resources
- Architecture-centric approach
- Services implemented as a collection of component types
- Evaluation of all possible service variants is not feasible
  - Scalable approach needed

Introduction

- Deploy a composition of abstract services
- as concrete services
- onto a set of connected nodes



Requirements

- Self-organization of service deployment configuration through learning
- Deal with scalability in number of variants
- Limited resources on mobile devices
- Limited or costly bandwidth situations
- Distribute adaptation reasoning among nodes
- Models include QoS properties of software and hardware

Motivating application

- *InstantSocial*
- Distributed platform for sharing content, gaming, ...
- Formed by devices joining federation ad-hoc
- Perceived by users as ordinary site
- Services are scattered across nearby devices and integrated by distributed application server
- Server consists of dynamic composition of services (also replicated)
- Network configuration evolves: services are redeployed or migrated to other devices

Genetic Algorithms

- GA is part of evolutionary computing
  - rapidly growing area of artificial intelligence
  - inspired by Darwin's evolution theory
    - solution to a problem evolves towards better fitness
- Evolutionary Computing
  - introduced in the 1960s by I. **Rechenberg**
    - in "***Evolution strategies***"
- Genetic Algorithms
  - invented by John **Holland**
    - "***Adaption in Natural and Artificial Systems***" (1975)
  - in 1992 John **Koza** called his method:
    - "***genetic programming***" (GP)

Biological Background

- Chromosomes:
  - are present in each cell of living organisms
  - consist of genes, which are blocks of DNA strings
    - presenting a model of the whole organism
  - **genome** is a complete set of organism's chromosomes
- Reproduction:
  - recombination or **crossover**:
    - a whole new chromosome is formed from parent genes
  - **mutation**:
    - elements of DNA are slightly changed: "copy-errors"
  - **fitness**:
    - measured by success of the organism in its life

Encoding

Encoding of a chromosome

- usually binary or integer string
- each element represents a characteristic of the solution
  - Chromosome 1: 1101100100110110
  - Chromosome 2: 1001111000011110
  - ...
- e.g. knapsack problem as binary encoding
  - 1 on position *i* : object *i* in knapsack
  - 0 on position *i* : object *i* not in knapsack

Search Space

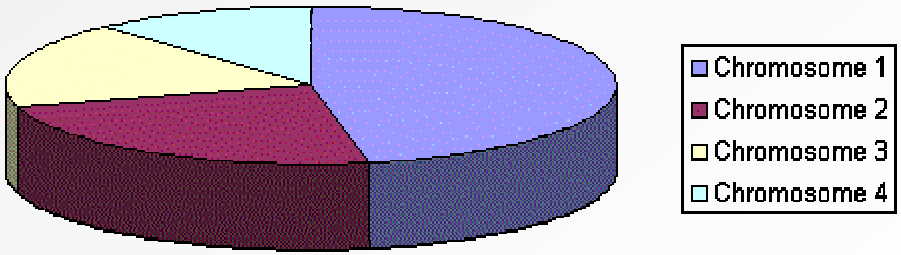
- Goal:
  - “Find the solution that is best among other solutions”
- Search space:
  - the space of all feasible solutions
  - each solution is marked by its “fitness” for the problem
- NP-hard problems:
  - not solvable in “polynomial time”
  - where to start looking for a solution?
  - approximation with polynomial algorithm:
    - "guess" the solution
      - by some nondeterministic algorithm
    - and then check it (calculate feasibility & fitness)

Genetic Algorithm

1. **[Start]** Generate random population of *n* chromosomes
2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x*
3. **[New population]** Create a new population
   repeat the following steps until the new population is complete:
   1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   2. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children).
      If no crossover was performed, offspring is an exact copy of parents.
   3. **[Mutation]** With a mutation probability mutate new offspring at each locus (locus = position in chromosome).
   4. **[Accepting]** Place new offspring in a new population
4. **[Replace]** Use new generated population for a further run of algorithm
5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Go to step **2**

GA parameters

- Population size:
  - small: exploration of search space is limited
  - large: algorithm slows down
- Elitism:
  - at least one best solution is copied without changes,
  - so the best solution found can survive to end of run.
- Parent selection:
  - probability based on fitness of chromosome
    - e.g. roulette wheel selection:



Crossover

- Goal:
  - selecting genes from parent chromosomes
  - creating a new offspring
- Random crossover point:
  - Chromosome 1 : 11011 | 00100110110
  - Chromosome 2 : 10011 | 11000011110
  - Offspring 1 : 11011 | 11000011110
  - Offspring 2 : 10011 | 00100110110
- Crossover probability:
  - offspring population fraction generated by crossover
    - e.g.: 70% crossover and 30% copies of parents

Mutation

- Goal:
  - prevents all solutions from falling into a local optimum
- Random changes to the new offspring:
  - Original offspring 1: 1101111000011110
  - Original offspring 2: 1101100100110110
  - Mutated offspring 1: 1100111000011110
  - Mutated offspring 2: 1101101100110100
- Mutation probability:
  - fraction of a chromosome that is mutated

Two-Step Solution

1. Service Composition Step
   - which concrete services can we allocate on each node?
     - given the abstract services that we want to use

2. Service Allocation Step
   - which nodes do we deploy the concrete services on?
     - taking into account:
       - communication costs
       - distance to feasible composite deployment

Service composition

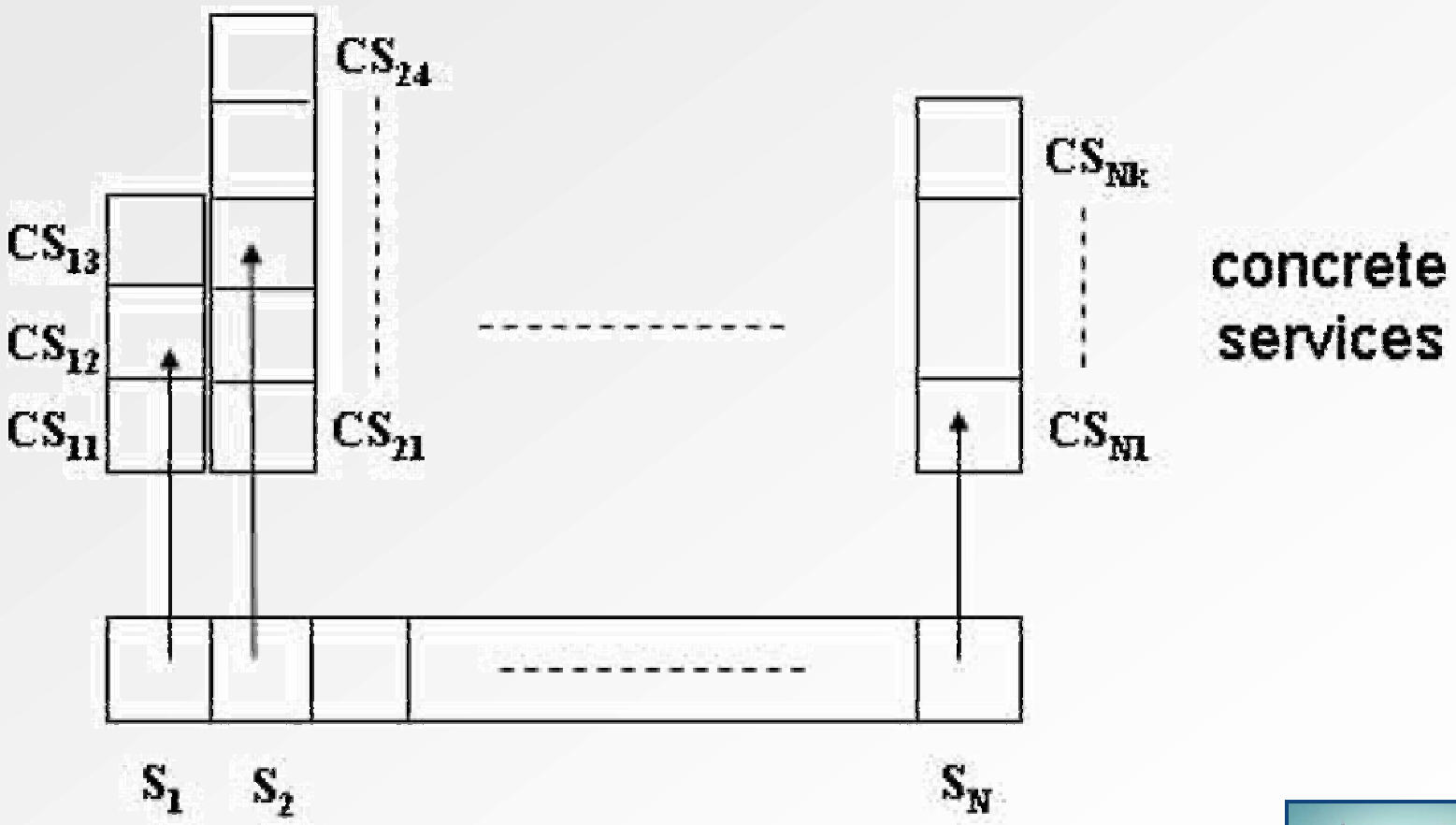
- Service composition
  - composed of abstract services
- Abstract Service
  - represents a service's functionality
  - has concrete services implementing the service
- Concrete Services
  - have their own QoS description
  - are implemented as set of components

Service Composition

- Perform a service selection step at each node
  - Calculate the optimal fitness of Service *i* on each node *i*
    - $F(g_{ij}) = F(S_{ij})$
  - not regarding communication cost
    - Bandwidth, latencies, remote deployment overhead, etc.
- Goal:
  - determine a set of concrete services
  - to be bound to the abstract services
- Requirements:
  1. QoS constraints in SLA need to be met
  2. other QoS parameters need to be optimized

Choosing Concrete Services using GA

Chromosome encoding:



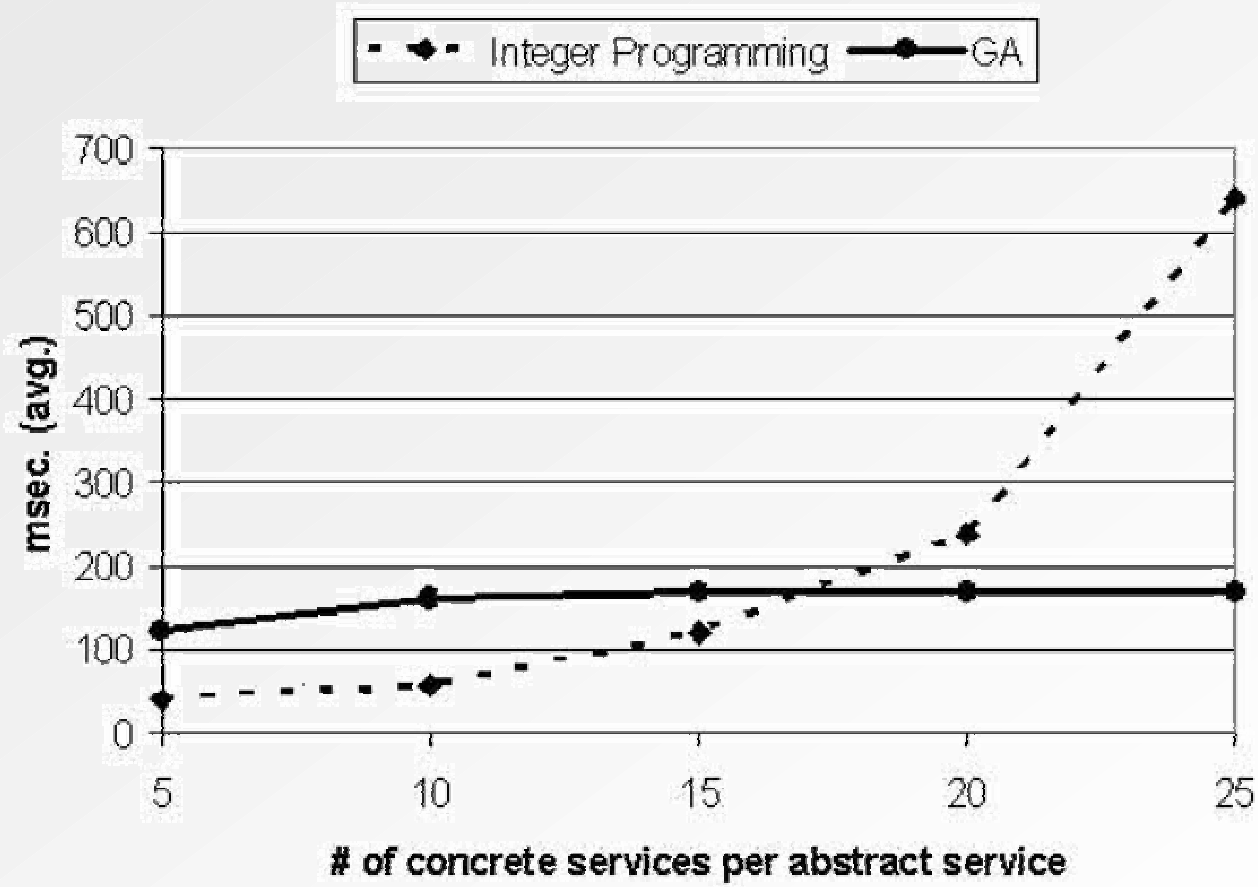
Fitness Function

- $F(g_{ij}) = Q(g_{ij}) – D(g_{ij})$

- $Q(g_{ij})$: weighted function of QoS parameters
- $Q(g_{ij}) = (w_1.Avail + w_2.Rel) / (w_3.Cost + w_4.Resp)$

- $D(g_{ij})$: distance from constraint satisfaction
- Constraints for the service variant to be able to run on node j

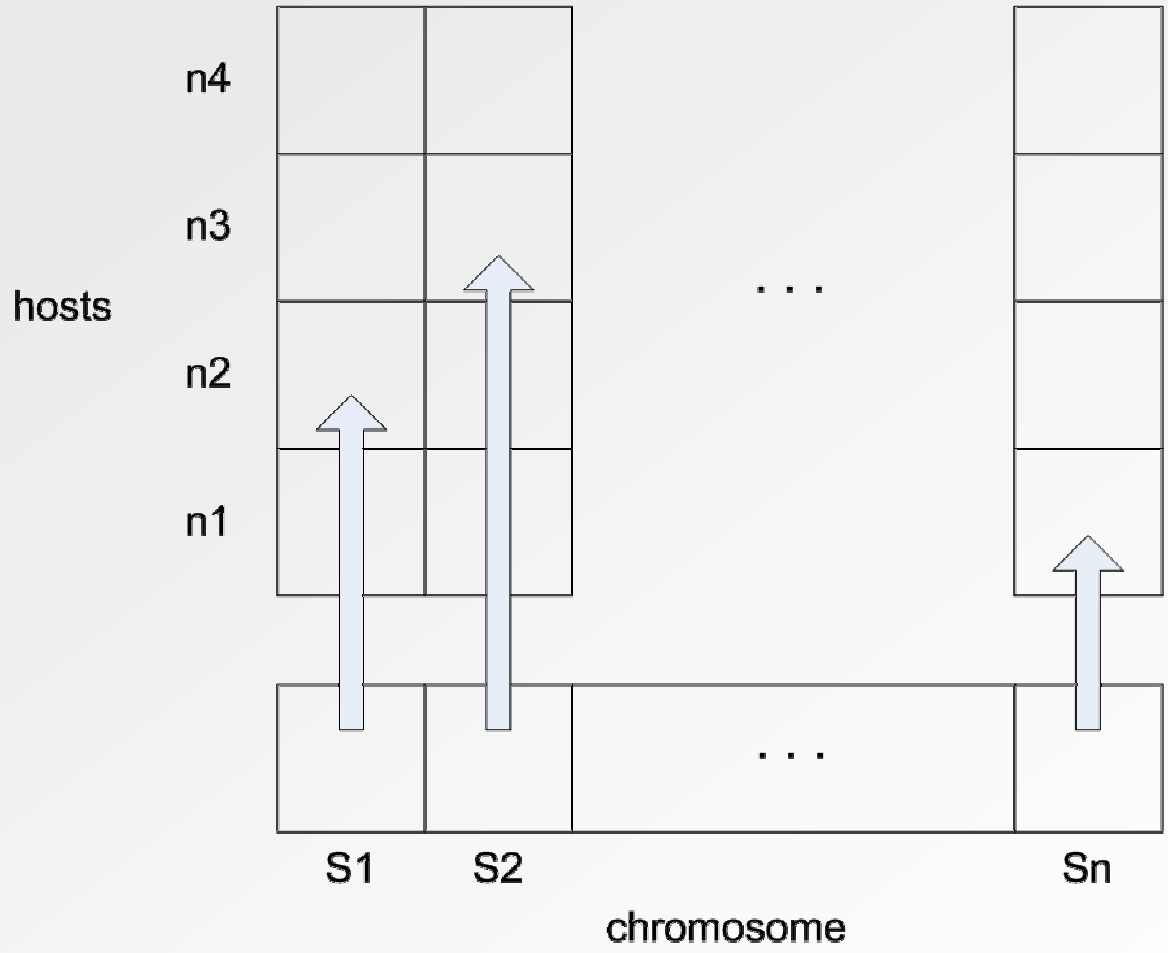
Comparison with Integer Programming



Step 2: Service Allocation Step

- Perform a service allocation step
  - Choose the node to allocate each service to
    - given $F(S_{ij})$
  - taking into account
    - communication costs
    - distance to feasible composite deployment

Step 2: Chromosome Encoding



Step 2: Fitness Function

- Fitness of allocation:

  $F(G) = \text{sum}(F(S_{ij})) - C(G) - D'(G)$

  - $F(S_{ij})$ = fitness of service deployment on each node
  - $C(G)$ = fitness of communication overhead
    - cost for bandwidth, latencies, etc.
  - $D'(G)$ = distance from feasible solution
    - penalty for QoS constraints that are not met

- Assumption:
  - $F(S_{ln}) + F(S_{kn}) = F(S_{ln}, S_{kn})$ *if* (S_{ln}, S_{kn}) is feasible
    - with (S_{ln}, S_{kn}) the composite deployment of S_l and S_k on host n
    - with $F(S_{ln}, S_{kn})$ the composite fitness of (S_{ln}, S_{kn})

Conclusions

- Scalable approach to compose and deploy services in a distributed way
- Future work: Implementation/evaluation in the middleware

Thank you

Questions?

www.ist-music.eu