# Provisioning Heterogeneous and Unreliable Service Providers

**Sebastian Stein, Nicholas R. Jennings, Terry R. Payne**

University of Southampton,
Southampton, SO17 1BJ, UK
{ss04r,nrj,trp}@ecs.soton.ac.uk

Electronics & Computer Science
University of Southampton

## Introduction

### Service-Oriented Computing

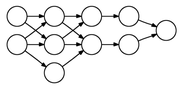
- Key technology for offering and consuming computational services in distributed systems.
- Allows service-consuming agents to execute large **workflows** by dynamically discovering and invoking providers at run-time.

### Problem

- Services are provided by autonomous agents.
- These are typically **heterogeneous** (performance varies between providers) and **unreliable** (they may fail and take an uncertain amount of time to execute).
- Such unreliability is a serious problem when executing critical workflows.

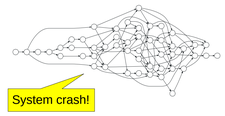
### Workflows

- Plans/templates for achieving a goal.
- Consist of individual tasks and precedence constraints.



### Unreliability Example

- Each service has a 95% success probability.
- Hence, there is only an 8% probability that *no* failure occurs.



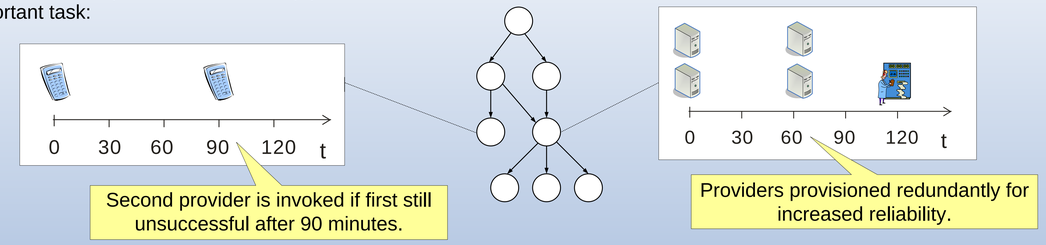

**Research Question:**
How to efficiently and effectively execute large workflows in these uncertain environments?

## Provisioning

- We focus on the **provisioning** of service providers.
- This allows the consumer to flexibly select the providers most suited for the current workflow.
- It may choose to provision a few cheap providers for a less important task:

**Definition:** *Provisioning* is the allocation of service providers to the tasks of an abstract workflow.

- Conversely, it may provision more expensive, reliable providers for a more critical task:




## Heuristic Algorithm

- To provision services, we want to find a provisioning allocation for all tasks ($\alpha^*$) that maximises the expected profit:

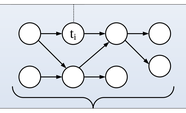
$$\alpha^* = \arg\max_{\alpha}(\bar{r}(\alpha) - \bar{c}(\alpha))$$

Expected reward — Expected cost

- However, we prove that this optimisation problem is **NP-hard**.
- Therefore, we develop a heuristic algorithm, based on a **local search**.
- This starts with a random provisioning allocation for each task and then gradually improves it by performing small changes.
- A **utility estimation function** is central to this algorithm. It estimates the expected reward for the consumer if following a given allocation.
- This function is based on the stochastic performance information available about each provider and proceeds in three stages:

(1) First, it calculates a number of performance parameters for each task $t_i$ in the workflow:

Success Probability ($p_i$) Expected Cost ($c_i$)
Mean Completion Time ($\lambda_i$) Variance ($v_i$)




(2) Then, these are combined across the workflow to give further performance parameters:

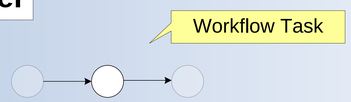
Success Probability (p) Estimated Expected Cost (c)
Estimated Completion Time PDF ($d_W$)

(3) Finally, the overall estimated utility is calculated:

$$\tilde{u} = -c + p\int_0^\infty d_W(x)u(x)dx$$

## Additional Information

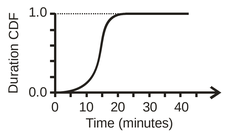
### Service Model



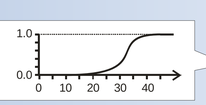

- There may be many suitable providers for each workflow task.
- Some **performance information** is known about each provider $s_i$:

$s_1$

- **Cost:** $c(s_1) = \$100$
- **Failure Prob.:** $f(s_1) = 0.01$
- **Duration**

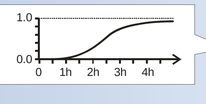



$c(s_{2\text{-}8}) = \$20$
$f(s_{2\text{-}8}) = 0.2$



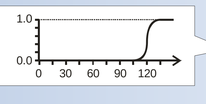

$s_{2\text{-}8}$

$c(s_{9\text{-}23}) = \$1$
$f(s_{9\text{-}23}) = 0.8$




$s_{9\text{-}23}$

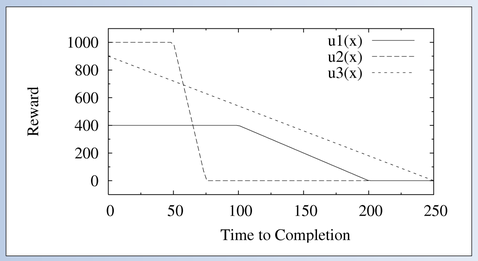
$c(s_{24\text{-}26}) = \$15$
$f(s_{24\text{-}26}) = 0.1$




$s_{24\text{-}26}$

### Utility Model

- We are interested in scenarios where workflows have a tangible value to the consumer. We model this using a **reward function** that is defined by:
  - A **utility value** ($u_{max}$) for the successful completion of a workflow.
  - A **deadline** (d) up to which $u_{max}$ is rewarded.
  - A **penalty** (δ) that is deducted from $u_{max}$ for each time step that the workflow is late.
- Example reward functions:

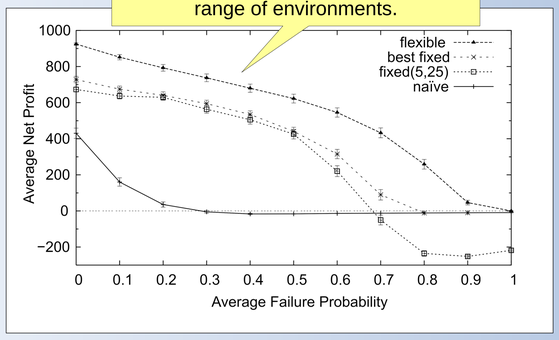



## Results & Conclusions

### Results

- We tested the heuristic algorithm (*flexible*) in randomly generated environments against a number of benchmarks:
  - **Naïve**: Provisions single provider for each task.
  - **Fixed(n,w)**: Uses manually specified redundancy and service time-outs to deal with unreliability.
  - **Best fixed**: Represents upper bound achievable using any fixed(n,w) strategy.

Flexible strategy performs well over a range of environments.




### Conclusions and Future Work

- Our flexible provisioning algorithm enables workflows to be executed in environments where providers are **highly unreliable**.
- The algorithm significantly outperforms current approaches that do not consider unreliability.
- Our abstract model can be applied in a variety of settings and frameworks (e.g., Grids, Web services, peer-to-peer systems).
- In ongoing work, we are currently considering the following extensions to our model:
  - **Advance reservations** of services (in the context of explicit service contracts).
  - Higher **system dynamism**, where the population of providers changes during the execution of a workflow.

### Bibliography

- Stein, S., Jennings, N. R. and Payne, T. R. (2007). Provisioning Heterogeneous and Unreliable Providers for Service Workflows. In *Proceedings of 22nd AAAI Conference on AI* (in press), Vancouver, Canada.
- Stein, S., Jennings, N. R. and Payne, T. R. (2006). Flexible Provisioning of Service Workflows. In *Proceedings of 17th European Conference on AI (ECAI-06)*, pp. 295-299, Riva del Garda, Italy.

Supported by:

BAE SYSTEMS
EPSRC Engineering and Physical Sciences Research Council