Secure Mission-Centric Operations in Cloud Computing

Massimiliano Albanese*, Sushil Jajodia*, Ravi Jhawar†, Vincenzo Piuri†

*George Mason University, USA
†Università degli Studi di Milano, Italy

ARO Workshop on Cloud Security
George Mason University, USA
March 11-12, 2013

Outline

- Motivation
- System overview
  - Mission and Cloud infrastructure
  - Mission deployment framework
- Secure mission deployment
  - Static mission deployment
  - Dynamic mission deployment
  - Incremental vulnerability analysis
- Mission protection
- Dependability in the mission interpreter

Motivation (1)

- Cloud computing is becoming increasingly popular
  - + Flexibility in obtaining and releasing computing resources
  - + Lower entry and usage costs
  - + Effective for applications with high scalability requirements
- Growing interest among users to leverage Cloud-based services to execute critical missions
- Exacerbate the need to ensure high security and availability of the system and the missions

Motivation (2)

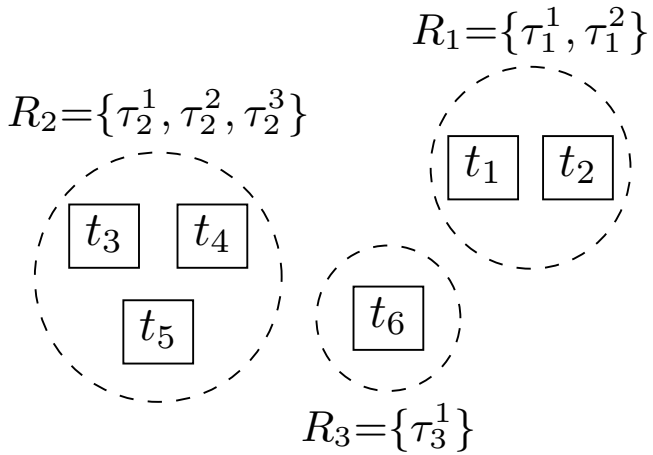
- Cloud computing infrastructure is highly complex
  - Vulnerable to various cyber-attacks
  - Subject to a large number of failures
  - Outside the control scope of the user's organization
- Existing solutions individually focus on the security of the infrastructure and the mission
  - Do not take into account the interdependencies between them
- Techniques to securely operate missions in vulnerable Cloud environments are necessary

Secure Mission Deployment and Protection

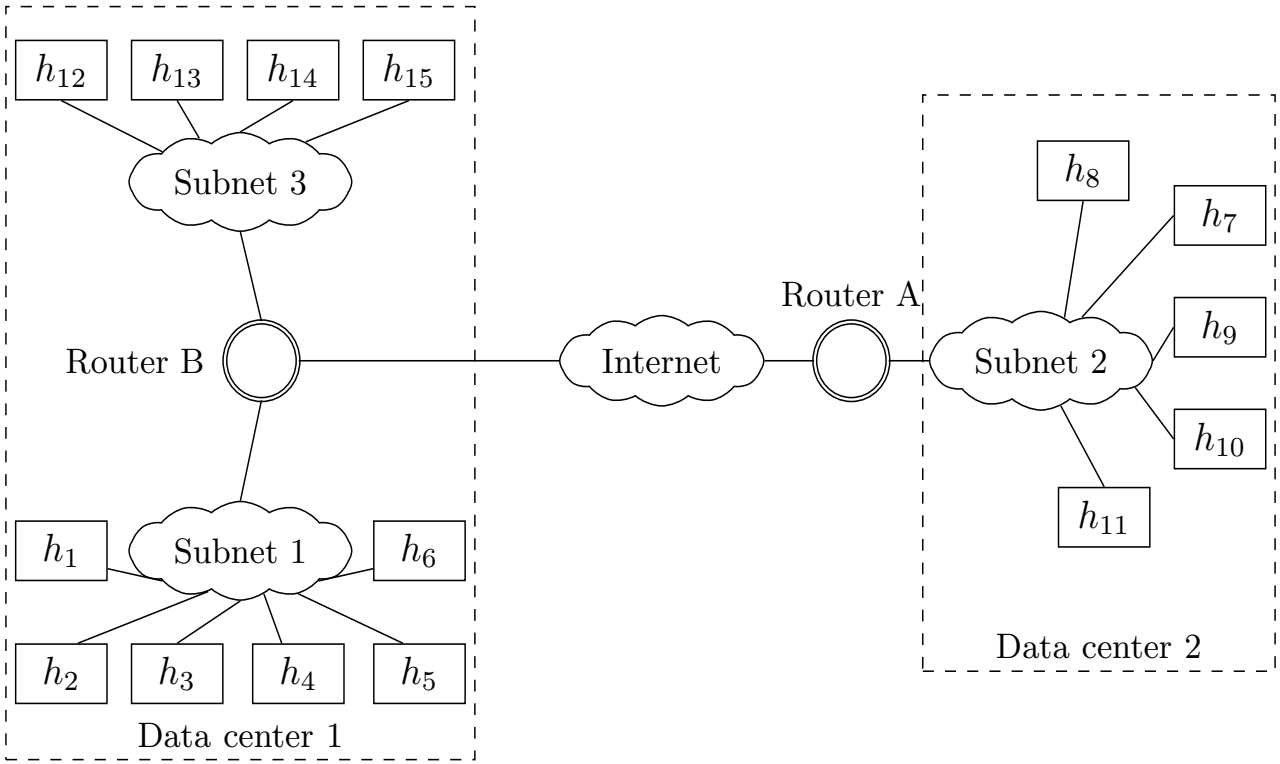
- Deploy mission tasks such that their exposure to vulnerabilities in the infrastructure is minimum
  - Static and dynamic versions of the problem
- Protect the hosts and network links used by the mission
  - Static protection hardens all the resources for entire duration of mission execution
  - Dynamic protection hardens resources corresponding to computation still to be executed
- Response to incidents

System Overview – Mission

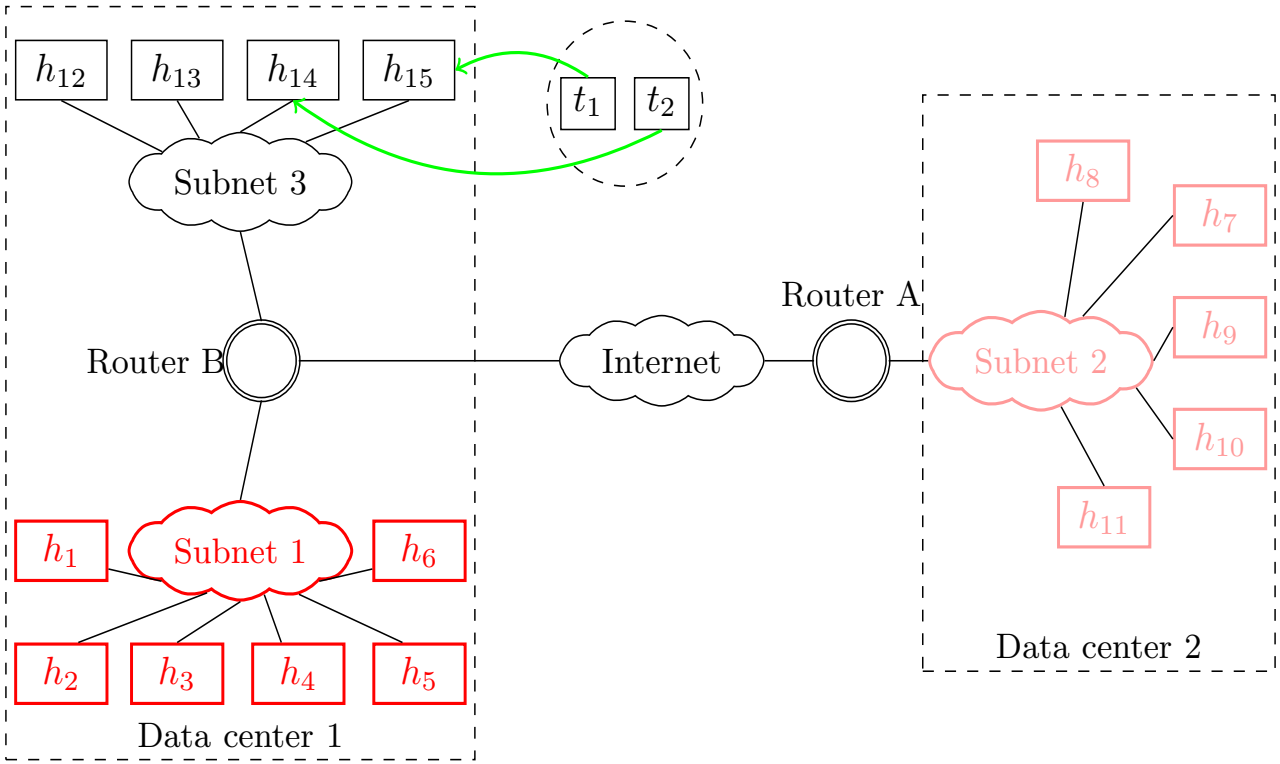
- A mission is a composition of a set of tasks $M=\{\tau_1,\ldots,\tau_m\}$
- Replicate critical tasks to improve the fault tolerance and resilience of the mission
- A replicated task set for τ_k is the set $R_k=\{\tau_k^l\}=\{\tau_k^1,\ldots,\tau_k^{|r_k|}\}$
- Mission is then a composition of all the tasks in the replicated task sets $T=\{t_i\}=\bigcup_{\tau_k\in M} R_k$



System Overview – Cloud Infrastructure

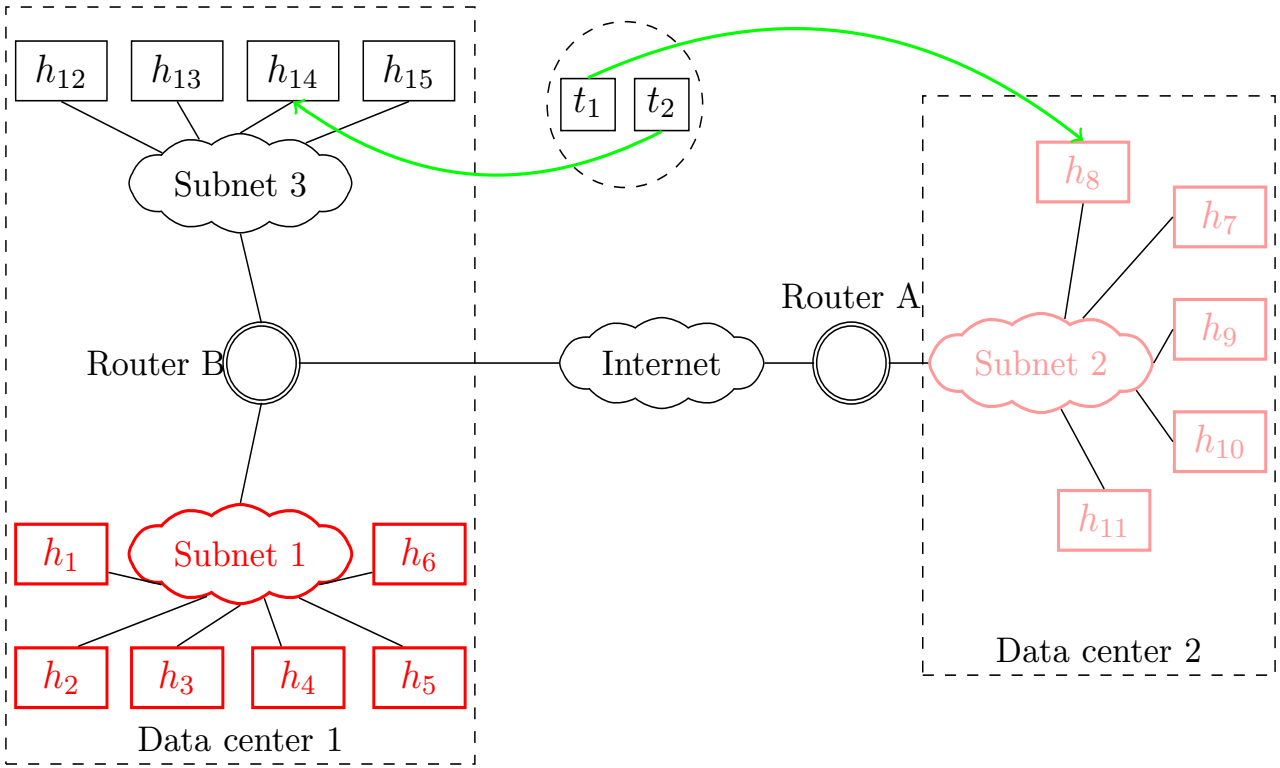


System Overview – Cloud Infrastructure



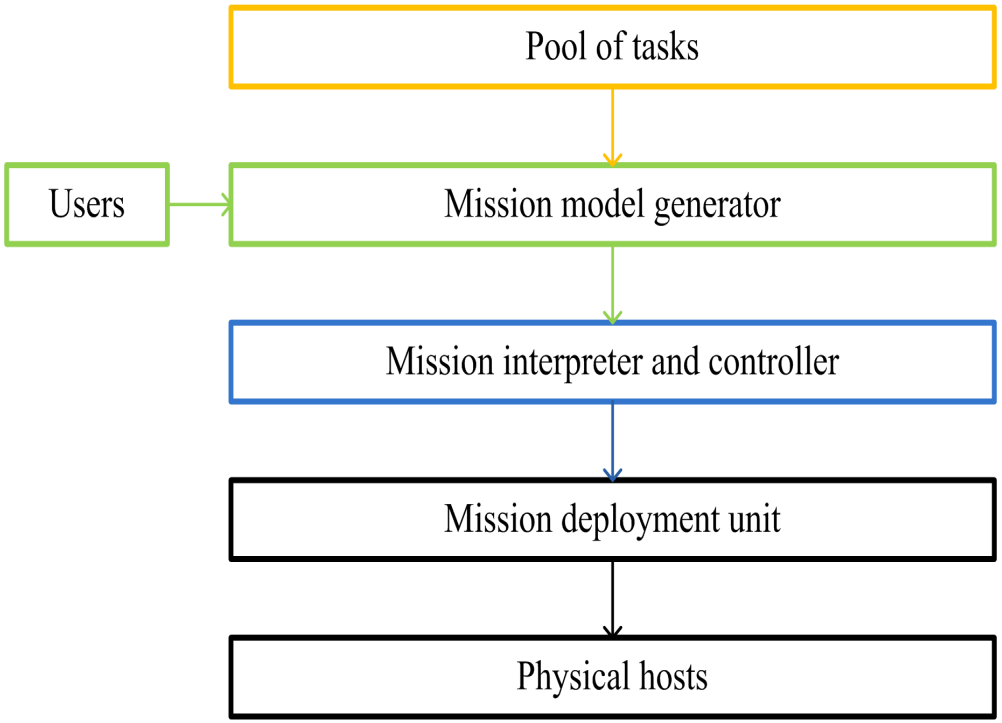
- Hosts may be vulnerable to various cyber-attacks
  (e.g., Subnet 1: compromised, Subnet 2: vulnerable, Subnet 3: highly secure)

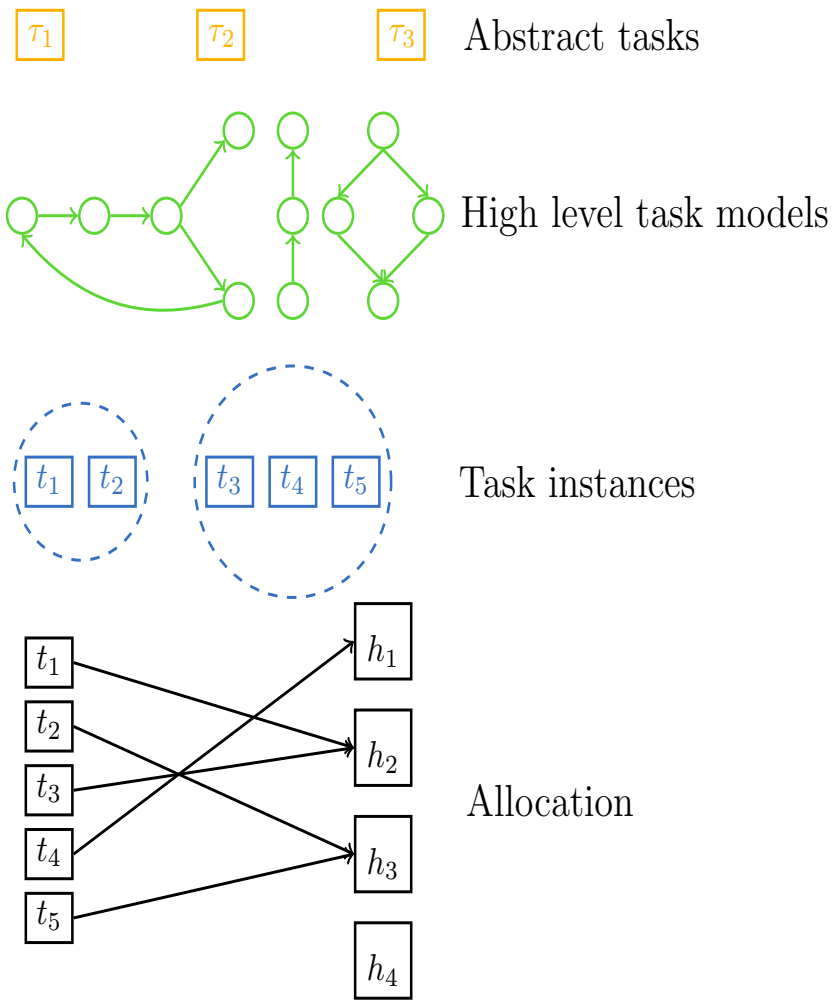
System Overview – Cloud Infrastructure



- Tasks may have vulnerability tolerance capability
  (e.g., Task 1 can handle buffer overflow attacks using memory management mechanisms)

Mission Deployment Framework Overview

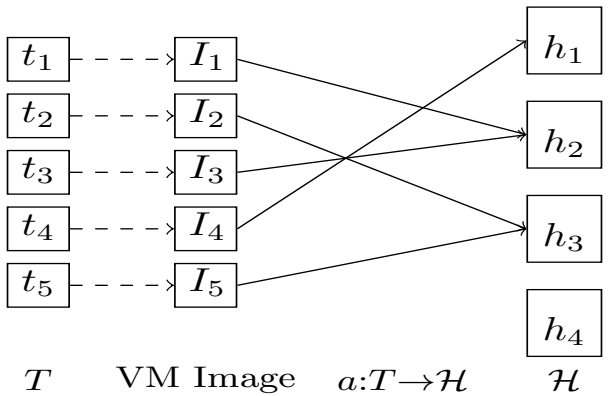




Mission Deployment

Static Mission Deployment

- Each host $h \in \mathscr{H}$ is associated with a vulnerability value V_h
- $tol(t)$ provides an estimate of the maximum level of vulnerability the task can be exposed to
- Task allocation problem with two sub-problems
  - Map each task to an appropriate VM image in the repository
  - Allocate VMs on suitable physical hosts in the Cloud



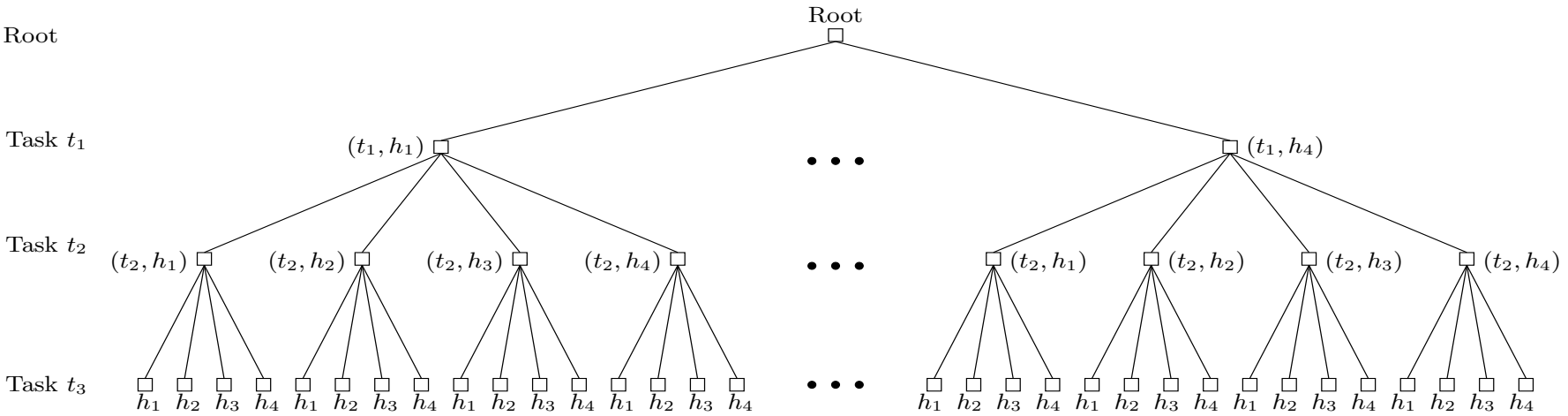
Selecting VM Images

- Challenge: Develop techniques to assess security of VM images at run-time and an automated security-driven search scheme to deploy mission tasks
- VM images encapsulate the entire software stack and determine the initial state of running VM instances
- Most Cloud IaaS require users to manually select VM images; in public Cloud services, VM images have critical vulnerabilities
- Objective: Select VM images that satisfy both functional requirements and security policy of mission tasks

Allocating VMs on Cloud Infrastructure (1)

- Challenge: Develop approximation algorithms to find suboptimal allocation solution in a time-efficient manner
- Objective is to minimize exposure of mission tasks to the vulnerabilities in the Cloud infrastructure
- Satisfy additional dependability constraints
  (e.g., host's capacity and task's vulnerability tolerance constraint)

Allocating VMs on Cloud Infrastructure (2)



- Possible solution: Use A^*-based state-space search approach
- State is a possible choice for allocating a task on a host (t_i, h_j)
- Root state is the initial state where no task is allocated
- Operation generates child states for a given state s
- Goal state is a state in which all the tasks have been allocated (leaf)
- Solution path is the path from root state to any goal state

Allocating VMs on Cloud Infrastructure (3)

- Objective is to find the solution path with minimum vulnerability value
- Cost function is the vulnerability measure of complete allocation $fvul(s) = gvul(s) + hvul(s)$
- $gvul(s)$ is the total minimum vulnerability due to task allocation from the root state to the current state s
- $hvul(s)$ is the lower-bound vulnerability estimate of the allocation from the current state to any goal state
  - $hvul(s)$ is computed using an admissible heuristic
  - Improves search performance while not compromising optimality

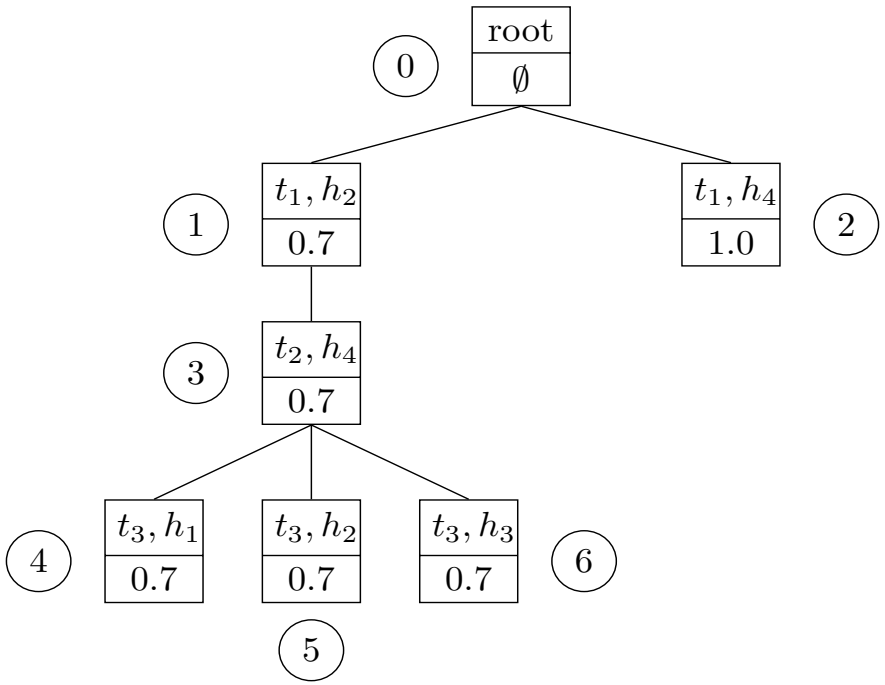
Allocating VMs on Cloud Infrastructure (4)

- Traversal scheme expands the states and generates the solution path starting from the root state
- The state with minimum $fvul(s)$ value is considered and its successors are generated (unless s corresponds to the goal state)
- For each successor s^* of s
  - Vulnerability cost $gvul$ is calculated
  - If s^* is already visited, and if its real cost is less than that of the current successor, it is dropped, and $hvul(s^*)$ value is estimated
  - The $fvul(s^*)=gvul(s^*)+hvul(s^*)$ value is determined and stored
- The parent state s is marked as visited

Example - Allocating VMs on Cloud Infrastructure (5)

Network		Application	
Host	Residual CPU capacity, Vulnerability level	Task	CPU Requirement, Vulnerability tolerance
h_1	0.5, 0.3	t_1	0.4, 0.2
h_2	0.7, 0.1	t_2	0.4, 0.2
h_3	0.3, 0.2	t_3	0.3, 0.4
h_4	0.5, 0.2		

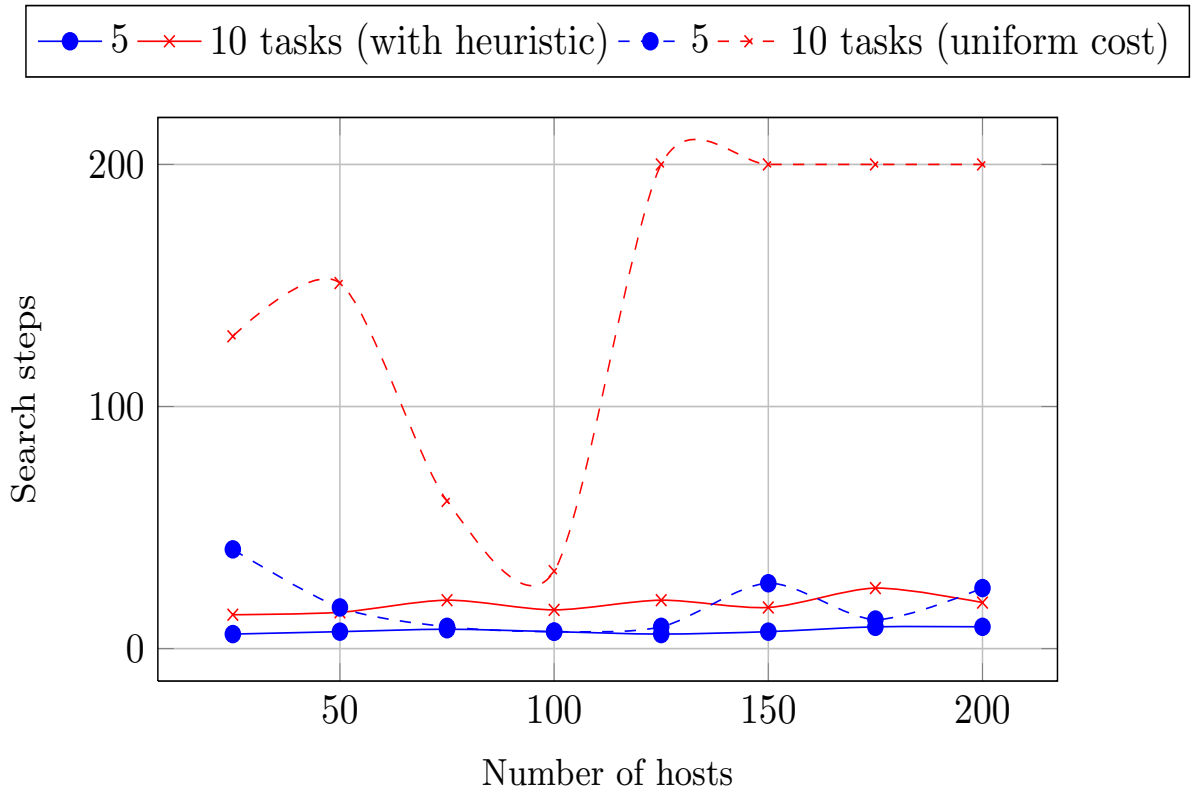
$\Delta V_{t_i,h_j}$	h_1	h_2	h_3	h_4
t_1	0.3	0.1	0.1	0.2
t_2	0.1	0.2	0.1	0
t_3	0	0.2	0.1	0.1



State-space tree nodes expansion sequence

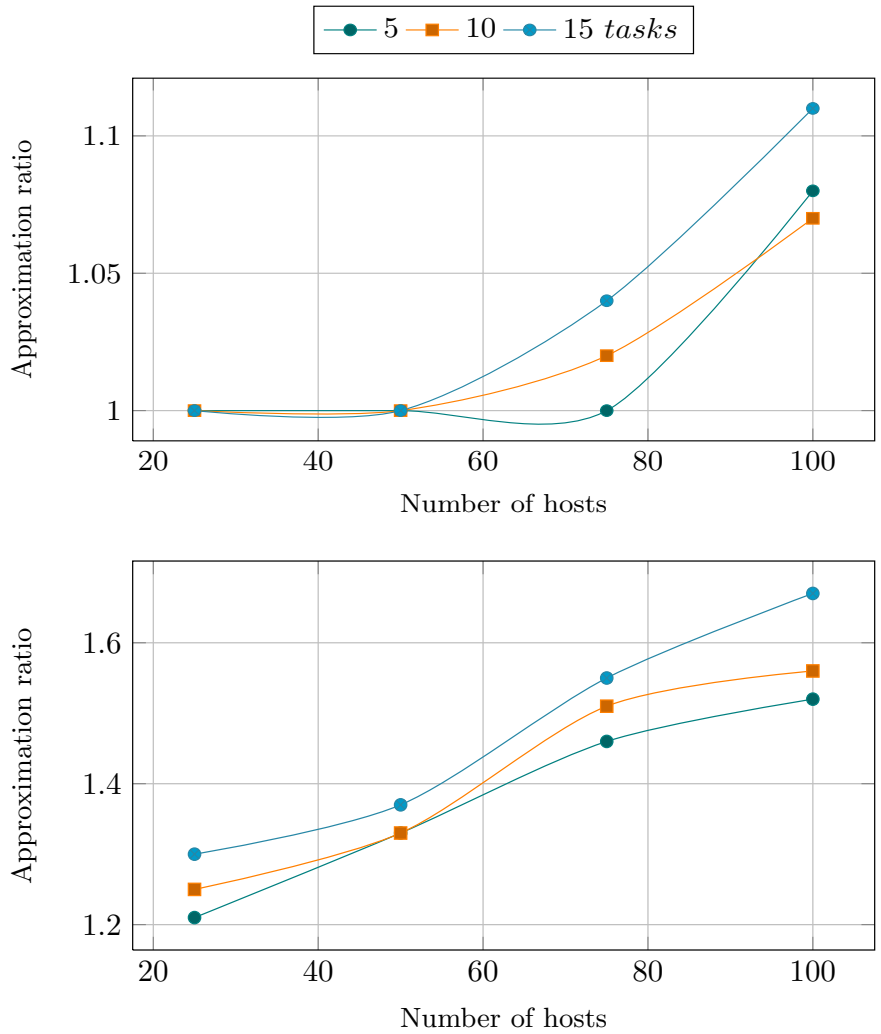
Allocating VMs on Cloud Infrastructure (6)

Number of search steps with and without the estimation heuristic



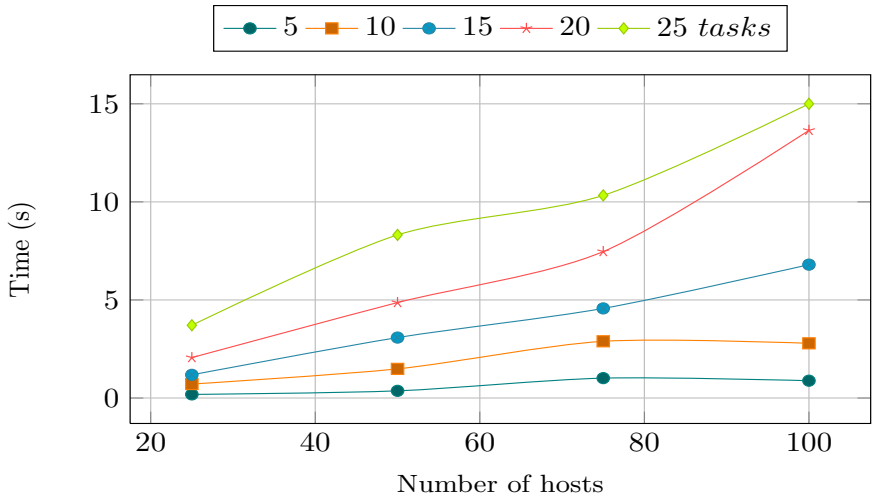
Allocating VMs on Cloud Infrastructure (7)

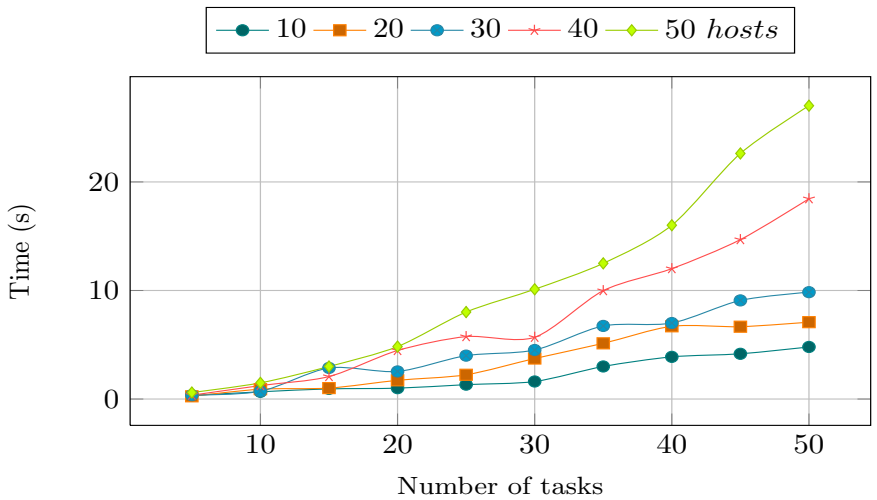
Quality of the solution wrt the (i) mission and (ii) globally across the Cloud infrastructure



Allocating VMs on Cloud Infrastructure (8)

Scalability of the proposed approach



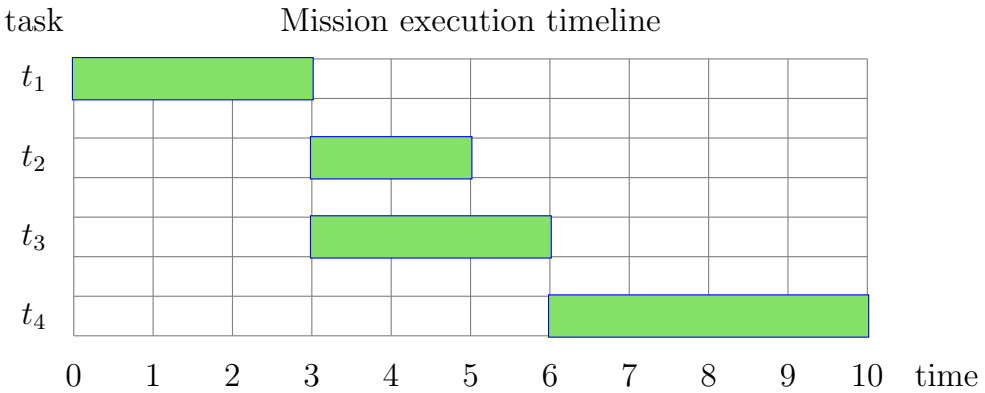


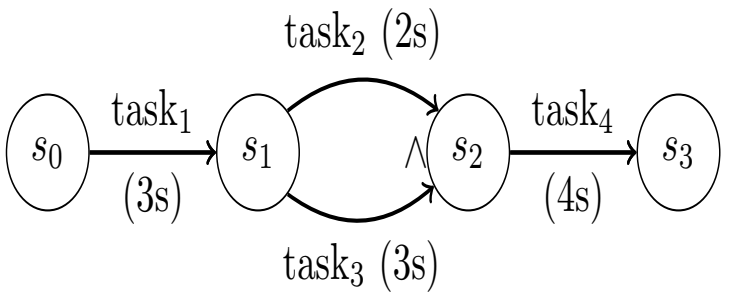
Dynamic Mission Deployment (1)

- Each task is associated with temporal constraints
  (e.g., a task may only run after another task)
- Critical missions must complete within a certain amount of time
- Possible solution: Complex task scheduling solution that takes into account the capability of the VM while computing the solution

Dynamic Mission Deployment (2)

- Challenge: Schedule mission tasks on the hosts
  1. To minimize their exposure to the vulnerabilities in the network
  2. To ensure their deadlines are met





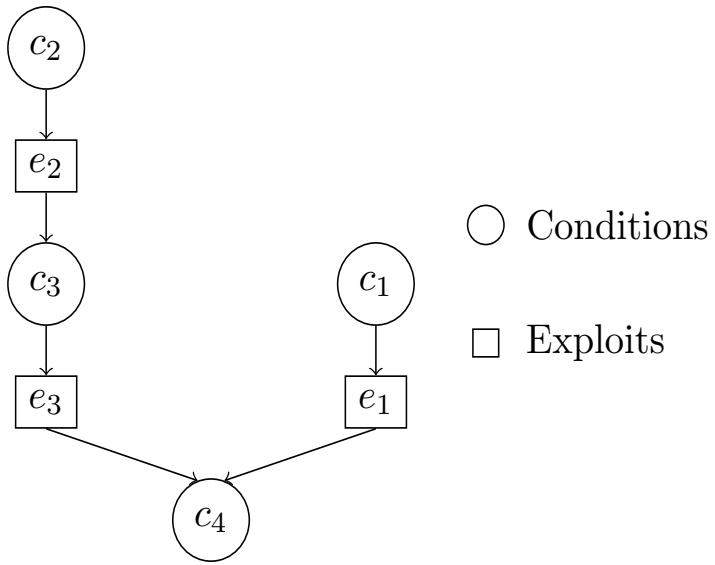
- Critical tasks (e.g., task t_3) must be placed on highly reliable host
- Adopt scheduling schemes such as greedy heuristics, genetic algorithms, tabu search, A* to solve the scheduling problem

Incremental Vulnerability Analysis (1)

- Challenge: Develop a scalable scheme to estimate the increase $\Delta V_{t_i,h_j}$ in vulnerability level of host h_j due to deployment of task t_i
- An allocation introduces new services (tasks, VMs) on a host and increases its vulnerability by ΔV
- Perform “what-if” analysis during mission deployment

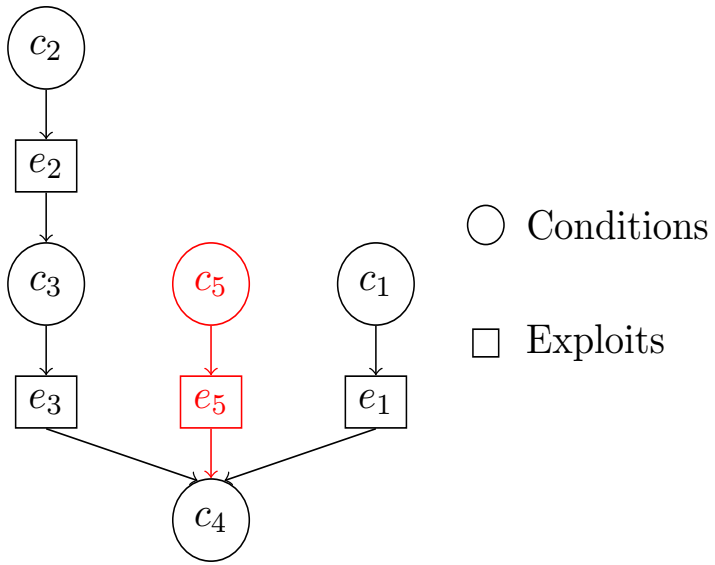
Incremental Vulnerability Analysis (2)

- Possible solution: Instead of recomputing the vulnerability "from-scratch", apply an "incremental" algorithm on attack graph
  1. Event-based approach
  2. Identify the changed parts of the graph and calculate the changed vulnerability
  3. Combine the results with already computed results



Incremental Vulnerability Analysis (2)

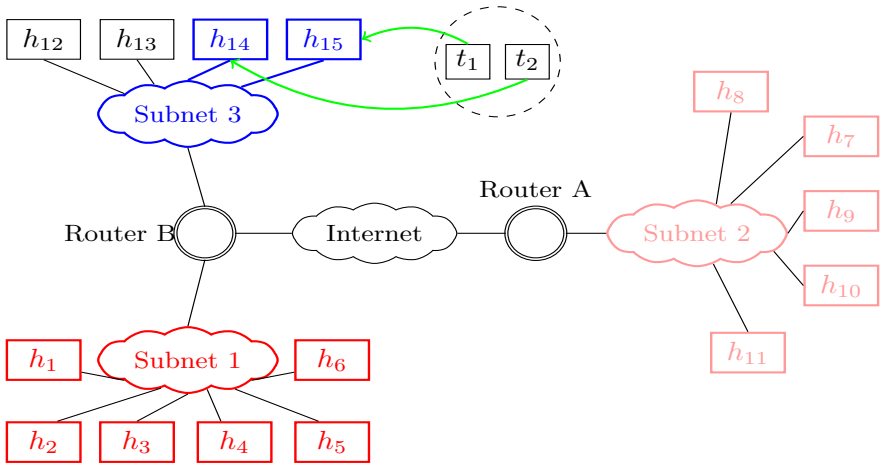
- Possible solution: Instead of recomputing the vulnerability "from-scratch", apply an "incremental" algorithm on attack graph
  1. Event-based approach
  2. Identify the changed parts of the graph and calculate the changed vulnerability
  3. Combine the results with already computed results



Mission Protection

Static Network Hardening

- Challenge: Given a mission is deployed in the Cloud, protect the resources used by the mission tasks
- In the static version, all the hosts and network links are protected for the entire duration of mission execution



- Possible solution: Build on top of previous work for network hardening

Dynamic Mission Protection

- Challenge: At any point in time, find a cost-optimal time-varying strategy to harden the resources not yet used by the mission
- Dynamic protection minimizes the disruption that hardening strategy causes to legitimate users
- Possible solution: Efficient technique that analyzes huge streams of security threats at real-time
- For example, use of attack graphs to track where the attacker is going (the penetration path)

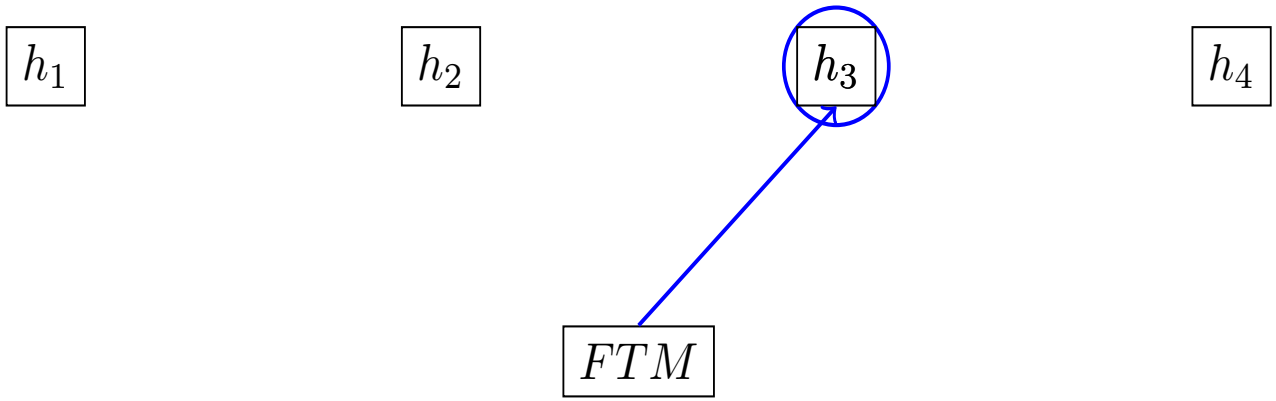
Dependability in the Mission Interpreter

Dependability Support for Mission Interpreter

- Realize the notion of Dependability as a Service
- Dependability schemes based on the virtualization technology (e.g., checkpointing virtual machine instances)
  - + introduce dependability in a transparent manner
  - + offers high level of generality
  - + Possible to change dependability properties based on business needs
- Construct dependability mechanisms at runtime
  - Mission centric Service Level Agreement (SLA)

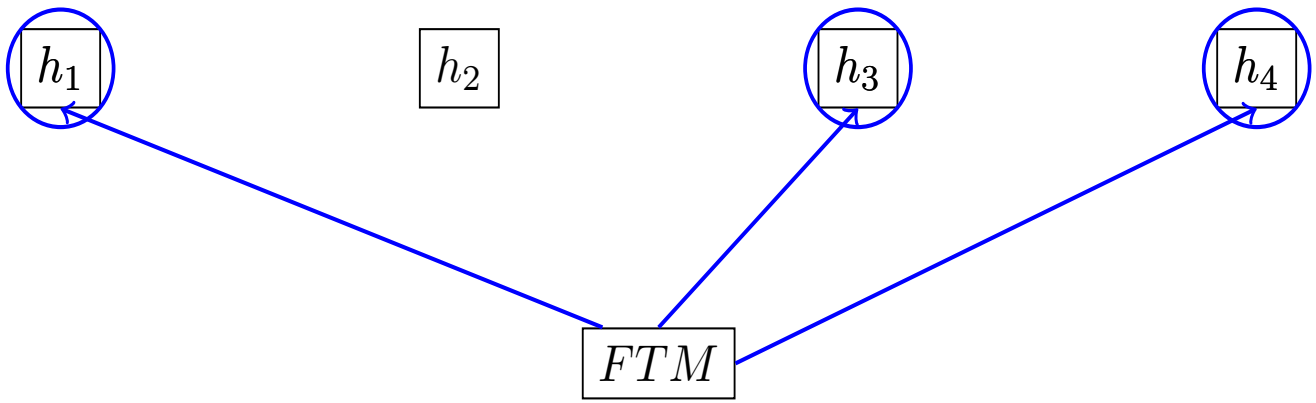
Dependability as a Service

- Build and deliver the service by orchestrating a set of micro-protocols
  - Realize dependability techniques as independent, stand-alone, configurable modules (web services)
  - Operate at the level of virtual machine instances
- VM instance replication technique



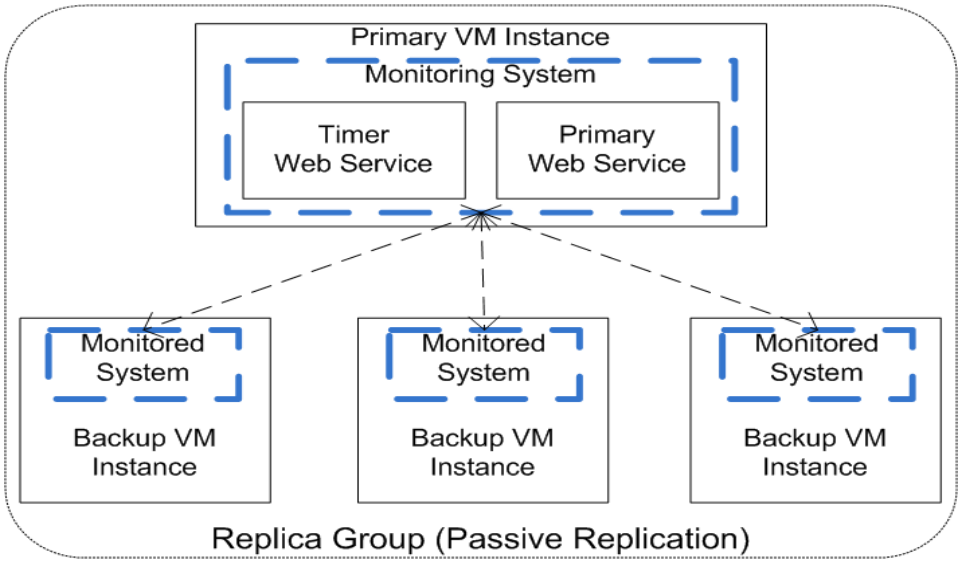
Dependability as a Service

- Build and deliver the service by orchestrating a set of micro-protocols
  - Realize dependability techniques as independent, stand-alone, configurable modules (web services)
  - Operate at the level of virtual machine instances
- VM instance replication technique



Dependability as a Service

- Build and deliver the service by orchestrating a set of micro-protocols
  - Realize dependability techniques as independent, stand-alone, configurable modules (web services)
  - Operate at the level of virtual machine instances
- Failure detection using hearbeat test

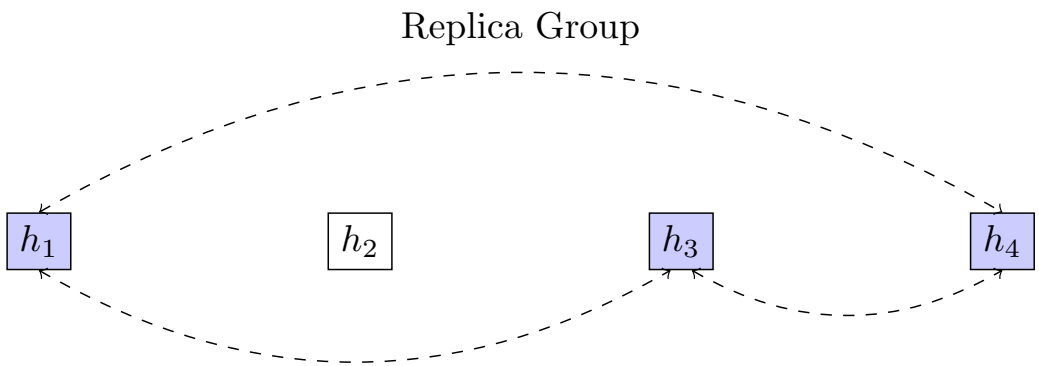


Example of a comprehensive dependability solution

- A fault tolerance policy:
  ft_sol[
  `invoke`:ft_unit(*VM-instances replication*)
  `invoke`:ft_unit(*failure detection*)
  `do{`
  `execute`(*failure detection ft_unit*)
  `}while`(*no failures*)
  `if`(*failure detected*)
  `invoke`:ft_unit(*masking mechanism*)
  `invoke`:ft_unit(*recovery mechanism*)
  ]

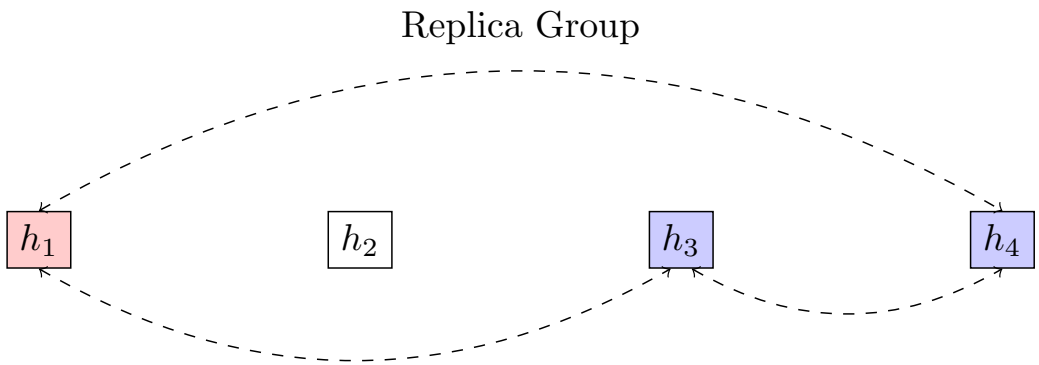
Example of a comprehensive dependability solution

- A fault tolerance policy:
  ft_sol[
  `invoke`:ft_unit(*VM-instances replication*)
  `invoke`:ft_unit(*failure detection*)
  `do{`
  `execute`(*failure detection ft_unit*)
  `}while`(*no failures*)
  `if`(*failure detected*)
  `invoke`:ft_unit(*masking mechanism*)
  `invoke`:ft_unit(*recovery mechanism*)
  ]



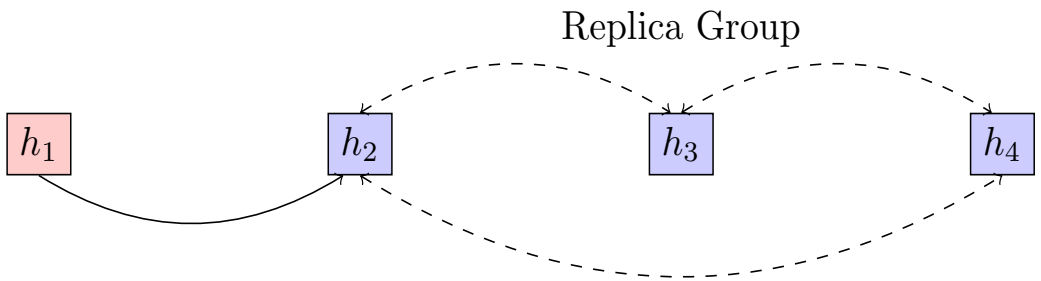
Example of a comprehensive dependability solution

- A fault tolerance policy:
  ft_sol[
  `invoke`:ft_unit(*VM-instances replication*)
  `invoke`:ft_unit(*failure detection*)
  `do`{
  `execute`(*failure detection ft_unit*)
  }`while`(*no failures*)
  `if`(*failure detected*)
  `invoke`:ft_unit(*masking mechanism*)
  `invoke`:ft_unit(*recovery mechanism*)
  ]



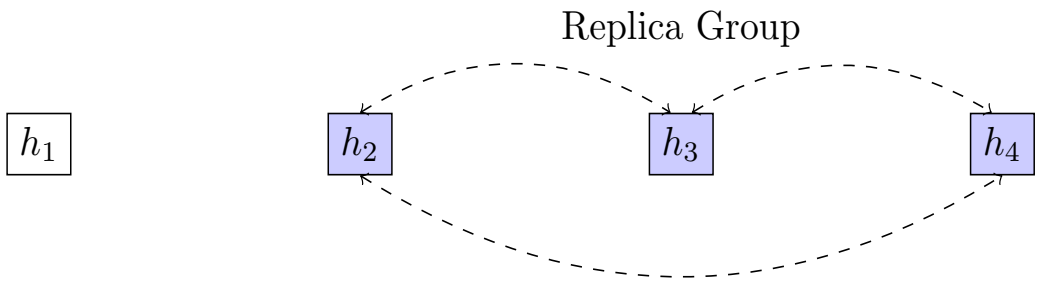
Example of a comprehensive dependability solution

- A fault tolerance policy:
  ft_sol[
  `invoke`:ft_unit(*VM-instances replication*)
  `invoke`:ft_unit(*failure detection*)
  `do{`
  `execute`(*failure detection ft_unit*)
  `}while`(*no failures*)
  `if`(*failure detected*)
  `invoke`:ft_unit(*masking mechanism*)
  `invoke`:ft_unit(*recovery mechanism*)
  ]



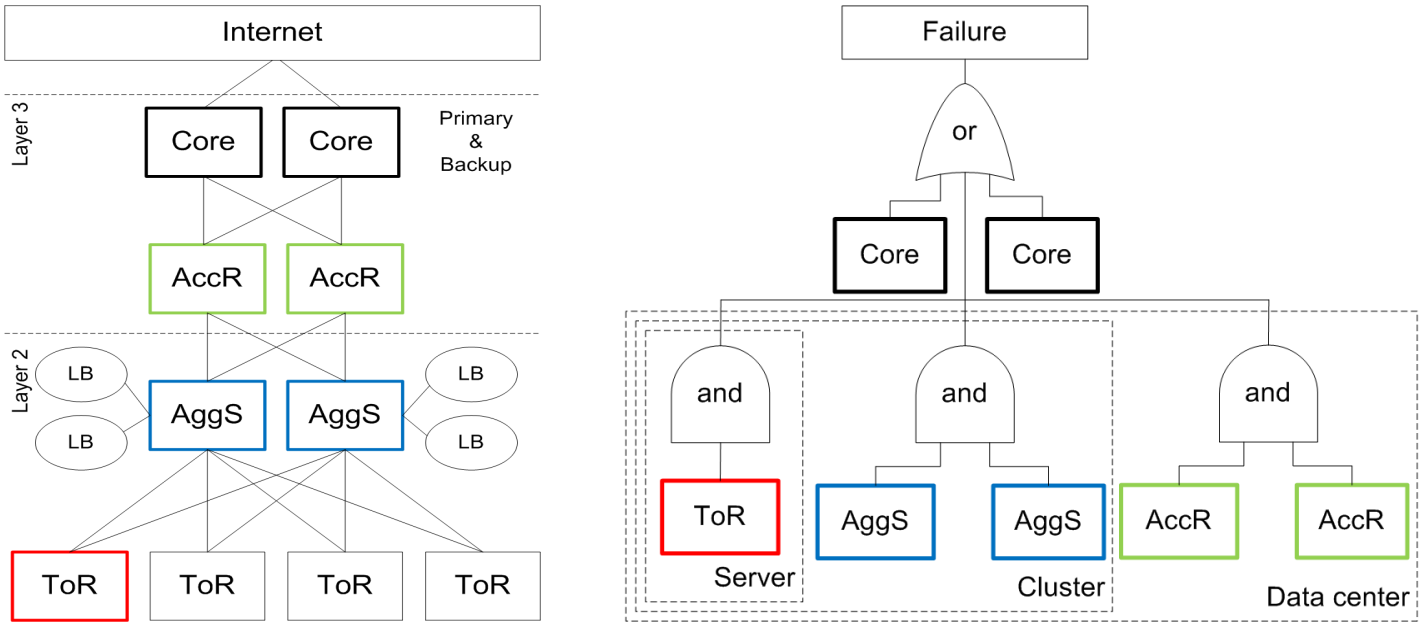
Example of a comprehensive dependability solution

- A fault tolerance policy:
  ft_sol[
  `invoke`:ft_unit(*VM-instances replication*)
  `invoke`:ft_unit(*failure detection*)
  `do{`
  `execute`(*failure detection ft_unit*)
  `}while`(*no failures*)
  `if`(*failure detected*)
  `invoke`:ft_unit(*masking mechanism*)
  `invoke`:ft_unit(*recovery mechanism*)
  ]



Configuration of a Dependability Solution (1)

- Based on the affect of failures on mission's tasks
- Using Fault trees and Markov chains

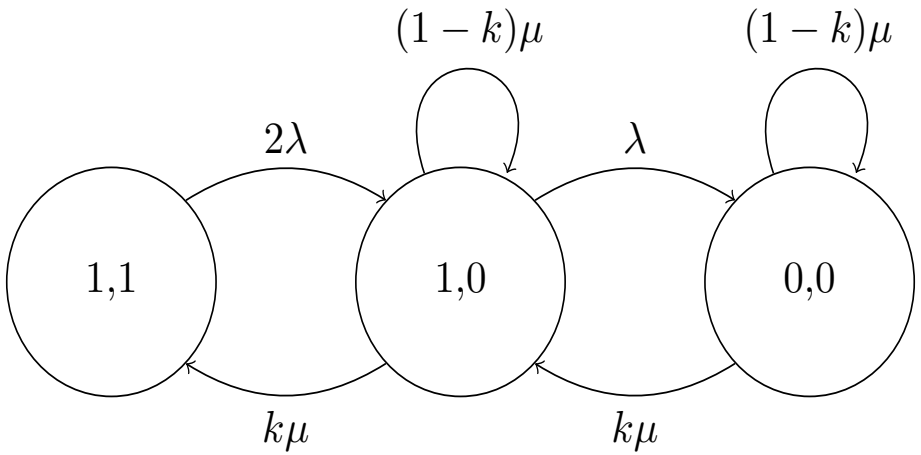


ToR–Top of Rack Switch AggS–Aggregate Switch

AccR–Access Router LB–Load Balancer

Configuration of a Dependability Solution (2)

- Analyze the properties of typical dependability mechanisms
- For example, semi-active replication
- Primary, Backup (λ – failure rate, μ – recovery rate, k – constant)



Identify possible deployment levels

- Based on data published by Kim et al. in PRDC'09, Smith et al. in IBM Systems Journal'08, Undheim et al. in Grid'11
- Availability values for each replication mechanism at different deployment levels

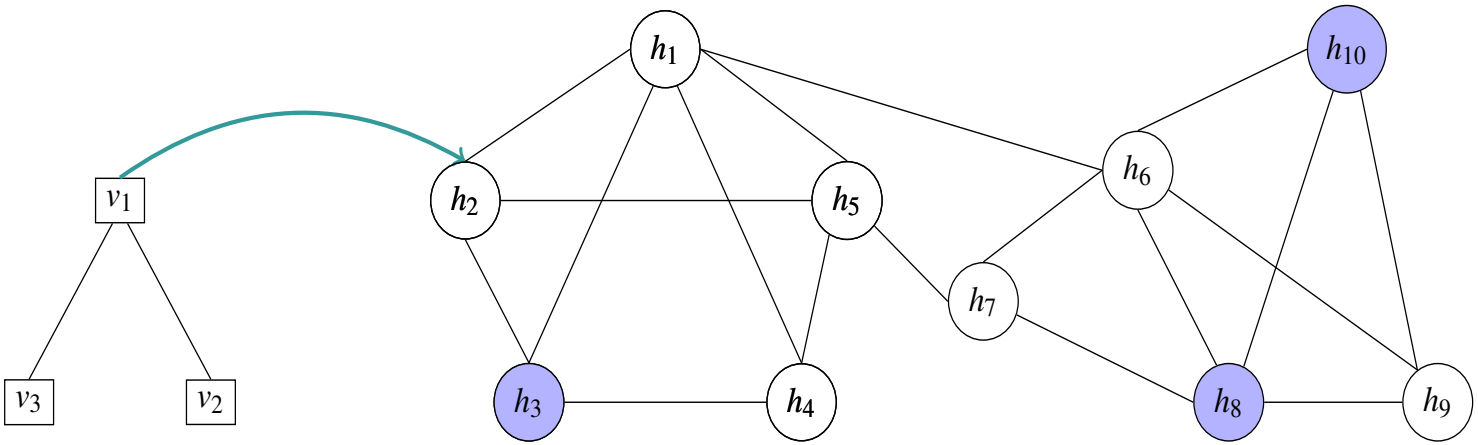
	Same Cluster	Same Data center, diff. clusters	Diff. Data centers
Semi-Active	0.9871	0.9913	0.9985
Semi-Passive	0.9826	0.9840	0.9912
Passive	0.9542	0.9723	0.9766

Replica Placement Constraints

- Location and performance requirements of replicas can be specified using constraints
  - Global constraints – Resource Capacity
  - Infrastructure oriented constraints – Forbid, Count
  - Application oriented constraints – Restrict, Distribute, Latency

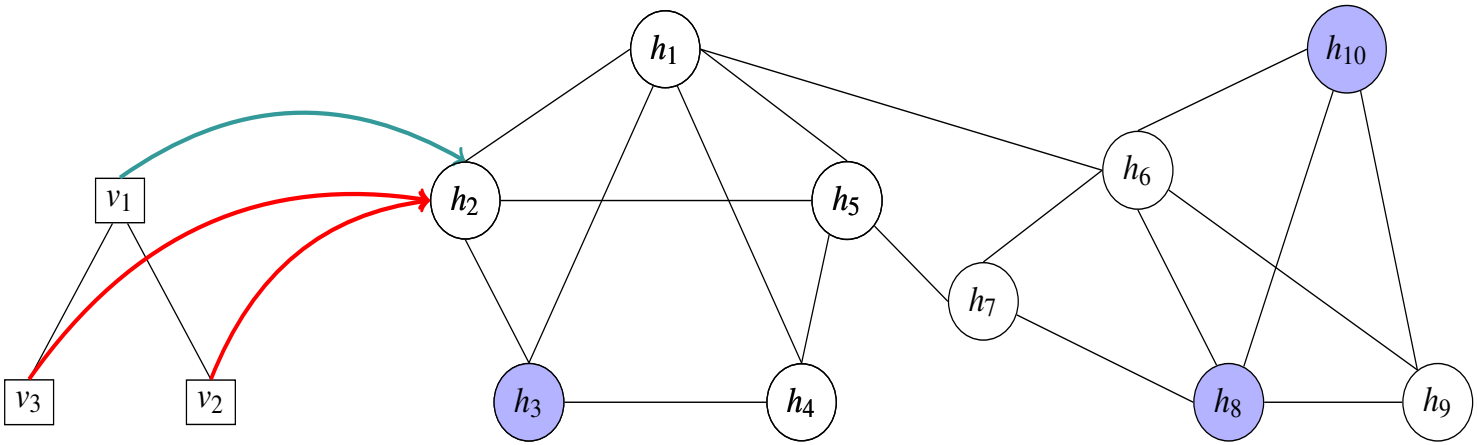
Example - Distribute Constraint

- To avoid single points of failure among replicated tasks
- Two tasks are never located on the same physical host



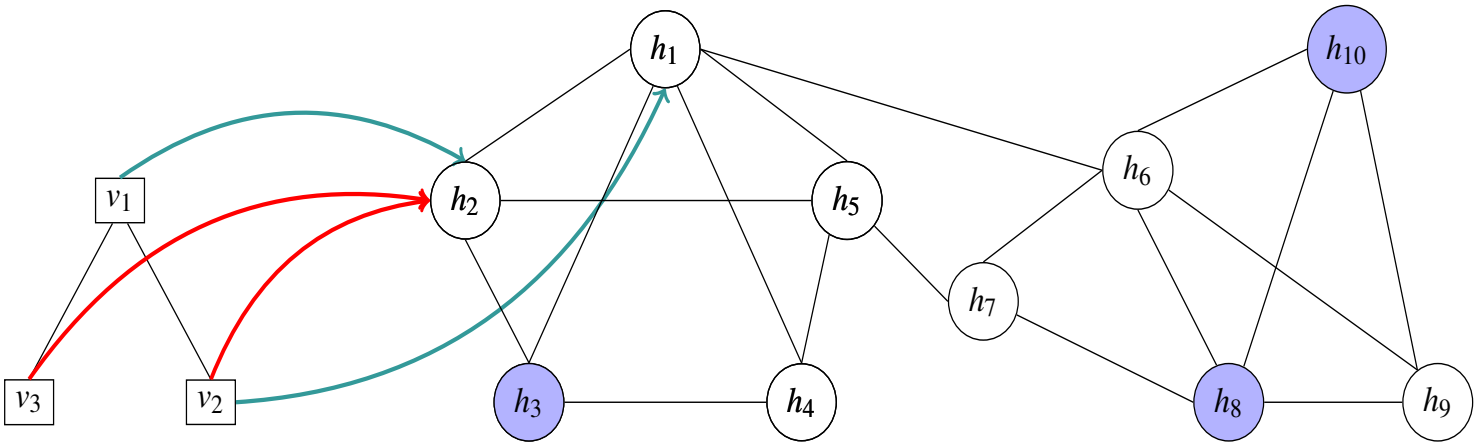
Example - Distribute Constraint

- To avoid single points of failure among replicated tasks
- Two tasks are never located on the same physical host



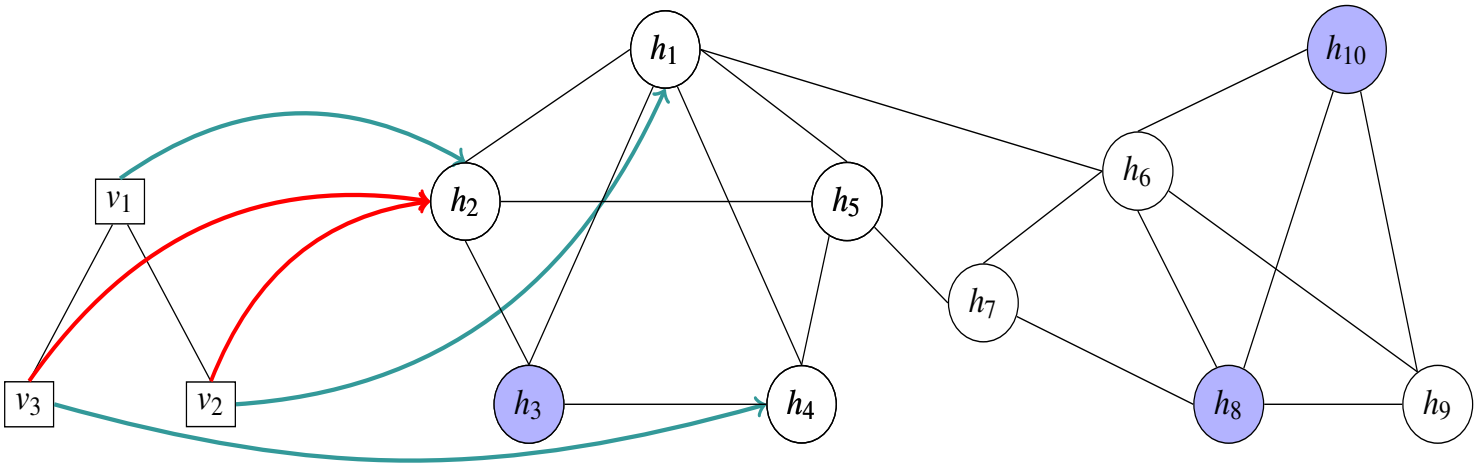
Example - Distribute Constraint

- To avoid single points of failure among replicated tasks
- Two tasks are never located on the same physical host



Example - Distribute Constraint

- To avoid single points of failure among replicated tasks
- Two tasks are never located on the same physical host



Conclusions

- Mission-centric techniques to improve the security and fault tolerance in Cloud computing
- Secure mission deployment techniques (allocation and scheduling)
- Static and dynamic mission protection by network hardening
- Provide complementary dependability support to the mission interpreter as a service

Acknowledgments

This work is supported by the Army Research Office under award number W911NF-12-1-0448, and by the Office of Naval Research under award number N00014-12-1-0461

Publications

Chapters in Books

- R. Jhawar, V. Piuri, "Fault Tolerance and Resilience in Cloud Computing Environments" in Computer and Information Security Handbook, 2nd Edition, J. Vacca (ed.), Morgan Kaufmann, 2013 (to appear)

International Journals Articles

- R. Jhawar, V. Piuri, M. Santambrogio, "Fault Tolerance Management in Cloud Computing: A System-Level Perspective," Systems Journal, IEEE, vol.PP, no.99

- C. A. Ardagna, R. Jhawar, V. Piuri, "Dependability Certification of Services: A Model-Based Approach", (under submission)

International Conferences and Workshops

- M. Albanese, S. Jajodia, R. Jhawar, V. Piuri, "Secure Mission Deployment in Vulnerable Networks", (under submission)

Publications

- C.A. Ardagna, E. Damiani, R. Jhawar, and V. Piuri, "A Model-Based Approach to Reliability Certification of Services," in Proc. of the 6th IEEE Int'l Conference on Digital Ecosystem Technologies - Complex Environment Engineering, Campione d'Italia, Italy, June, 2012

- R. Jhawar, and V. Piuri, "Fault Tolerance Management in IaaS Clouds," in Proc. of the 1st IEEE-AESS Conference in Europe about Space and Satellite Telecommunications, Rome, Italy, October 2-5, 2012

- R. Jhawar, V. Piuri, and P. Samarati, "Supporting Security Requirements for Resource Management in Cloud Computing," in Proc. of the 15th IEEE Int'l Conference on Computational Science and Engineering, Paphos, Cyprus, December 5-7, 2012

- R. Jhawar, V. Piuri, and M. Santambrogio, "A Comprehensive Conceptual System-Level Approach to Fault Tolerance in Cloud Computing," in 2012 IEEE Int'l Systems Conference, Vancouver, BC, Canada, March 19-22, 2012