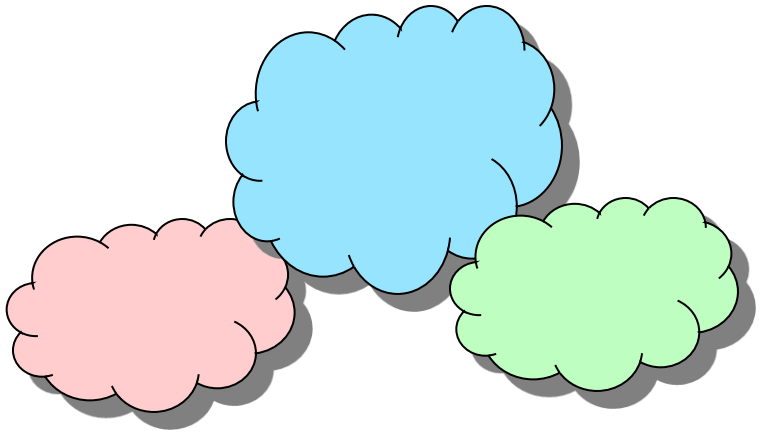


Towards Programmable Infrastructures: the Steps made by Cloud Computing and their Technical Support

Dana Petcu, West University of Timisoara, Romania

Content

- Programmable Infrastructures
- Programmable Services of the Cloud
- Automatic Clouds
- Particular solutions for Automatic Clouds
- Conclusions

Programmable Infrastructures?

- Programmatic access
  to the devices connected to the Internet
- Identified until recently with programmable networks
- Should involve:
  - network switches (Cloud networking),
  - simple gadgets or instruments (Internet-of-Things)
  - data center resources (Cluster, Grid, Cloud computing)
- Problem:
  - Manually intervention is still required in several processes involving e-infras settings

Grids and Clouds

- **Steps made by Grids:**
  - Globus - marriage with Web services
  - SAGA – Simple API for Grid Applications
    - job handling and monitoring, file transfer and management, distributed orchestration mechanisms.
    - Python and C++
    - Uniform Access-layer to DCI (EGI, XSEDE, DATAONE, UK NGS, NAREGI/RENEKI) and Clouds (recent)
  - gEclipse
    - an integrated, Grid enabled workbench tool for Grid appl users, developers and operators based on the Eclipse platform

- **Steps made by Clouds:**
  - (programmatic) elasticity in term of resources
  - uniform treatment of infrastructure, software, networking as (programmable) services

Status of programming services for Clouds

- Software-as-a-service
  - hide completely the e-infrastructure from the user
- Infrastructure-as-a-service
  - still requires a manual intervention of the application deployers to set the execution environments,
- Platform-as-a-service
  - allows to reach a certain level of programmability.
  - elasticity promised as main characteristic of the Cloud computing is even not supported as feature by all PaaS
  - proprietary tools and APIs lead to a vendor lock-in problem hardly accepted by the users of PaaS

Requirements for programmability

1. To establish an abstract model of the resources that is sufficiently general
   - to catch the characteristics of a large variety of resources
   - to be able to be instantiated as unique resource representative by using the model parameters
2. A proper programming paradigm should be used to express the actions applied to these models.
3. Proper tools should support the resource models and programming paradigm.

Two perspectives of the programmability

1. **The application developer**
   - interested to control programmatically the resources that are used for a particular applications
     - basic requirements need to be fulfilled, like
       - one point of access
       - immediate reaction in case of a resource fault
       - the control of the number of resources that are used
2. **The infrastructure provider**
   - interested to reach a certain level of automatization by programmatic
     - self-management,
     - self-tuning,
     - self-configuration,
     - self-diagnosis, and
     - self-healing of the resource provisioning system.

mOSAIC's proposal for the appl developer

- Abstraction of the Cloud services that:
  - Ensures vendor agnosticity
  - Use the common denominator of several similar services
  - Implemented in the form of 'Connectors' and 'Drivers'
- Few points of entry:
  - From Eclipse when developing
  - Using the web interfaces when deploying and controling
  - Assisted by a multi-agent system, semantic engine, cloud ontology and SLA mechanism to find the proper Cloud (i.e. multiple Clouds through a broker)
- The platform ensures:
  - Built-in fault tolerance mechanism
  - Web interface for controling of the processes and resources that are consumed

Automatic Clouds: for the providers

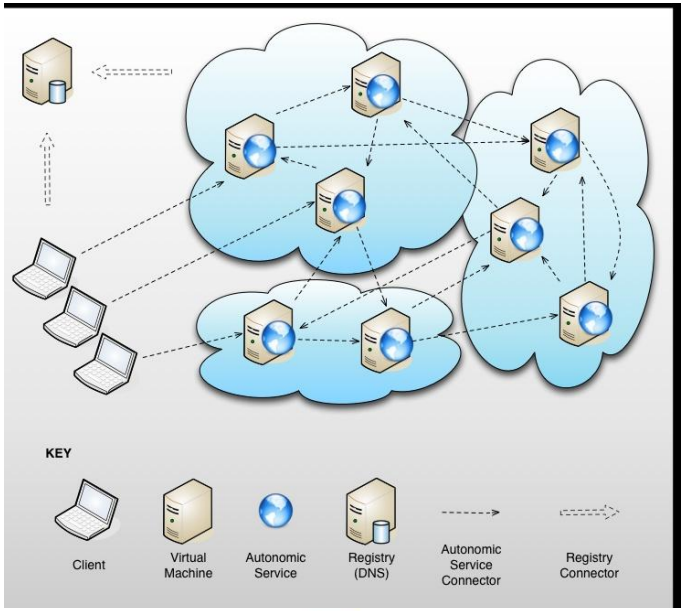
- Represent the highest target of a programmable Cloud
  - Introducing automatic computing techniques in Clouds expected to reduce of the human intervention at the Cloud provider sites
- Particular suited for when rapid elasticity is requested
  - for adaptation to a variable number of requests
  - or to ensure the high level of reliability despite the potential massive failures.
- Existing proof-of-the-concept
  - are based on known methods from AI, like multi-agents systems, genetic algorithms, neural networks, multiobjective optimization heuristics, semantic engines etc
- mOSAIC PaaS
  - intends to be a deployable middleware for Cloud service providers
  - includes incipient form of support for Automatic Cloud

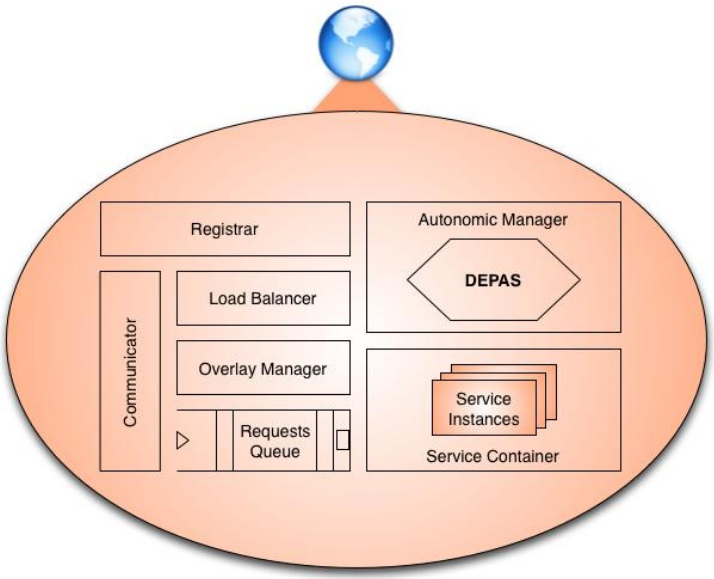
Research issues related to Automated Clouds in AMICAS supporting the mOSAIC extension

- **Auto-scaler**
  - N.M. Calcavecchia, B.A.Caprarescu, E. Di Nitto, D. J. Dubois, D. Petcu, DEPAS: *A Decentralized Probabilistic Algorithm for Auto-Scaling*, Computing, Springer, doi: 10.1007/s00607-012-0198-8, June 2012, available at http://arxiv.org/abs/1202.2509
  - B. A. Caprarescu, D. Petcu. *Decentralized Probabilistic Auto-Scaling for Heterogeneous Systems*, ADAPTIVE 2012 – July 28, available at http://arxiv.org/abs/1203.3885
- **Scheduler**
  - M. Frincu, *Scheduling Highly Available Applications on Cloud Environments*, Future Generation Computer Systems, May 2012, doi:10.1016/j.future.2012.05.017

Auto-scaler (PhD stud. B. Caprarescu)

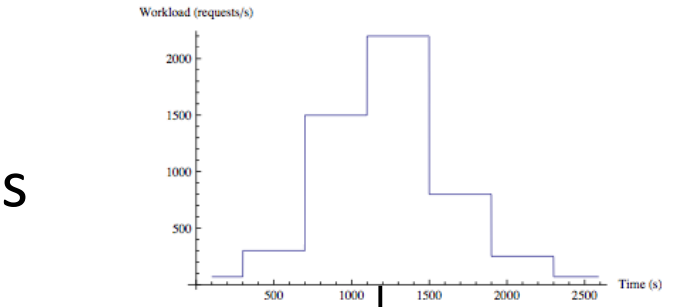
- Problem:
  - Most existing auto-scaling solutions are **centralized**
- Solution:
  - based on a P2P architecture
  - one autonomic service is deployed on each VM
- DEPAS algorithm:
  - each VM probabilistically decides to add new nodes or remove itself.
  - probability is computed based on an estimation of the average system load
  - the average system load is approximated by each node with the average load of itself and its neighbors

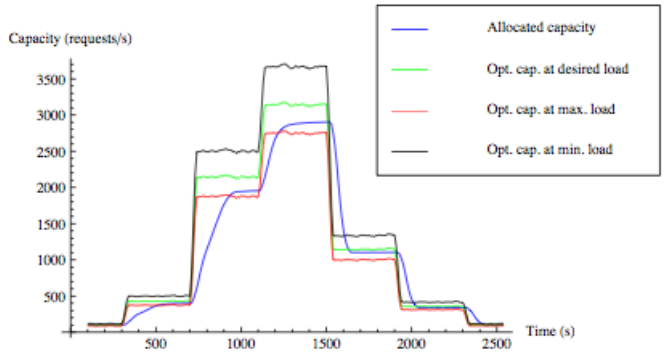


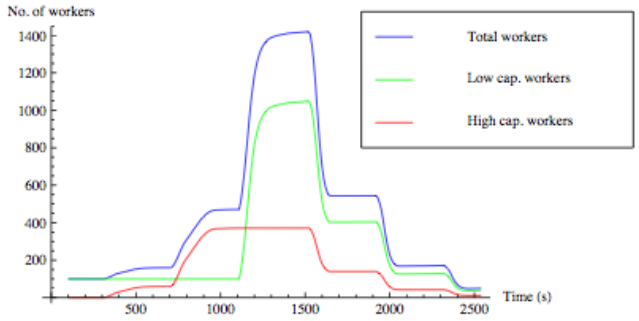


Auto-scaler (PhD stud. B. Caprarescu)

- Simulation results: ADAPTIVE procs
- On Amazon EC2: COMPUTING j.
- Test conclusions:
  - After a period of adaption, DEPAS allocates a right capacity (between the optimum capacity at max load and optimum capacity at min load)
  - The delay in adaptation is cased by the relatively high duration of load monitoring timeframe and cycle duration
  - The benefit comes in the high stability (no oscillations) which is impressing for a decentralized algorithm





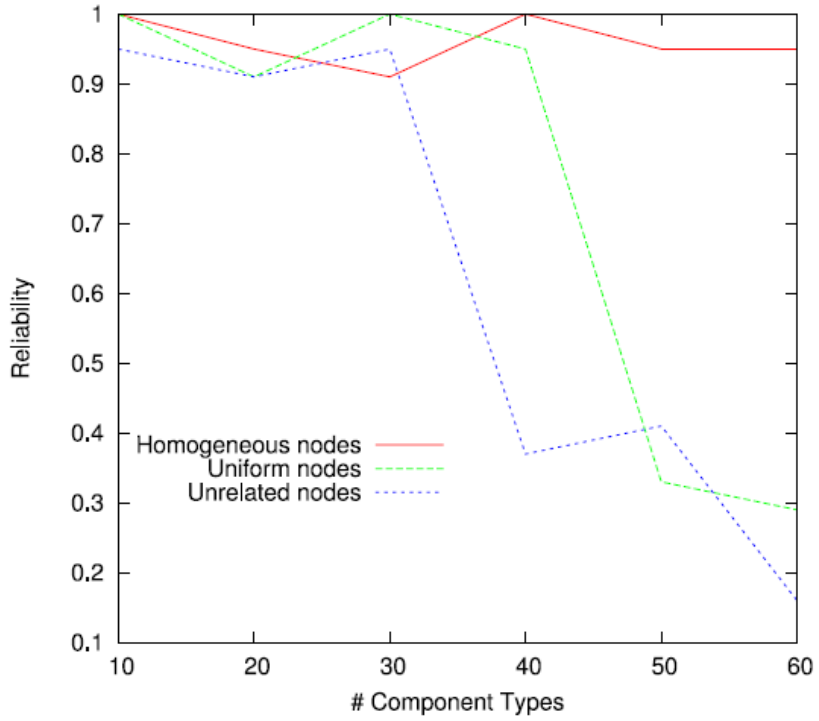


Scheduler (PostDoc Marc Frincu)

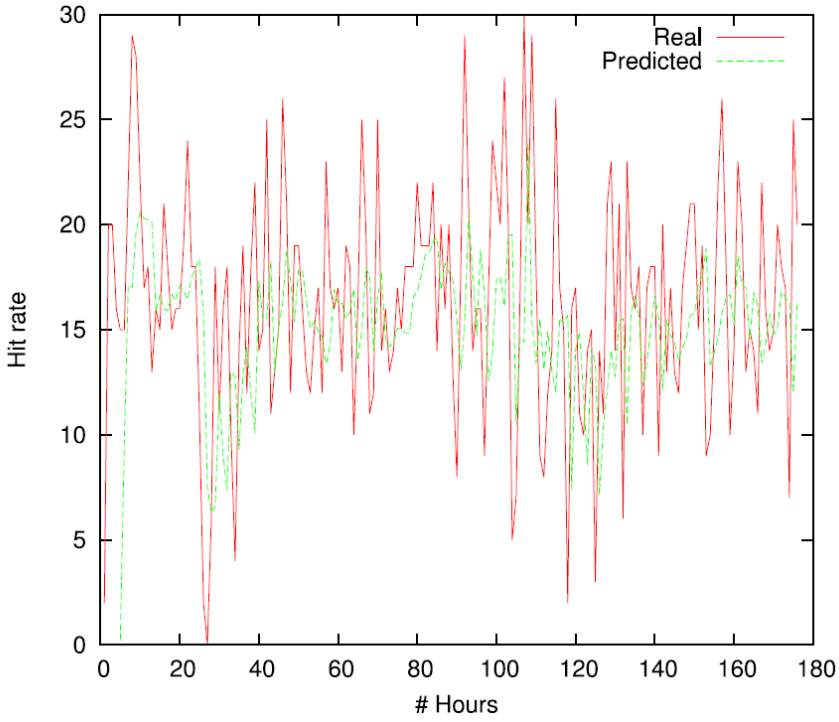
- Problem:
  - component-based appls can encounter failures of components
  - a scaled application can span its components on several nodes
  - finding the optimal no. component types needed on nodes so that every type is present on every allocated node
  - cost restrictions and threshold for no. nodes
- Application to:
  - Highly available Web 2.0 applications
- Novelty:
  - Most of the approaches schedule VMs not components
- Scheduling algorithms:
  - One that produces an optimal solution in case when the load of every component is known
  - One that produces a sub-optimal solution and relies on a GA to allocate components in case the component load is unknown

Scheduler (PostDoc Marc Frincu)

- Test goals:
  - Ability to achieve high availability measured through reliability indicator
  - Heterogeneity of the load on every node; expect load close to max of resource capacity



(a) Reliability.



Predicted vs. real traffic using neural networks

Conclusions

- Programming infrastructure
  - Is far from being well-supported, even in Clouds
  - Network, storage and computing are seen as pay-as-you-go services but still not integrated and collaborating sufficiently
- Automatic Cloud
  - Highest target of programmability that is emerging
  - Requires complex solutions involving SE and AI techniques
- mOSAIC
  - Has target until now the appl developer perspective
  - In one of its extension, AMICAS, first steps were made to support automatization, i.e. scheduler & auto-scaler