Resource Allocation for IaaS Clouds
Scalable Algorithm for Application Placement

Rerngvit Yanggratoke Fetahi Wuhib

October 11, 2012

Background

Cloud Computing is a technology whereby applications are delivered as services over the Internet [2]. A common service provided by cloud computing is Infrastructure-as-a-Service (IaaS). In an IaaS cloud, the type of service provided is virtual hardware resource that can be used to run virtual machines and that is accessible over the Internet. There are a number of public IaaS cloud providers [1, 8]. There are also a number of open source cloud management software that allow creating private IaaS clouds [6, 7]. Often, the term cloud computing is used to refer to the service as well as the hardware and system software that is used to provide the service.

Resource allocation is an important function in managing clouds. In the context of IaaS clouds, the function includes identifying a specific physical machine to run a specific virtual machine and how much resource of the physical machine to allocate to the virtual machine. This problem is hard when certain *management objectives* should be met. The resource demand of virtual machines changes over time. This means that, the resource allocation in the cloud may need to be recomputed, in order to meet the management objectives. For economic reasons (cf. [2]), cloud services are provided by very large datacenters and for a larger number of users. This means that the resource allocation function has to be scalable to 100,000 physical machines and millions of users.

In this project, you will use a simple model for the resource allocation process in an IaaS cloud, and you will develop a gossip-based heuristic protocol for resource allocation. (Gossip-based algorithms will be introduced during the first lecture for this project.)

1 Introduction

The goal of this project is to address the problem of resource allocation in a large datacenter providing an IaaS service. The datacenter consists of a set of servers (i.e., physical machines) hosting a set of virtual machines (VM). Since VMs are typically used to run applications, we also refer to a VM as an 'application' and the resource allocation process as 'application placement'. We assume that each server has a fixed amount of CPU capacity and each application has a time-dependent CPU demand. In this setting, the problem is to find a placement of applications onto the servers that satisfies the performance objective of the provider (see figure 1). For this project, we consider a combination of the following two objectives: *load balancing* across the servers and *minimizing energy consumption* of the servers. Out of consideration for scalability, you develop distributed solutions based on gossip protocols.

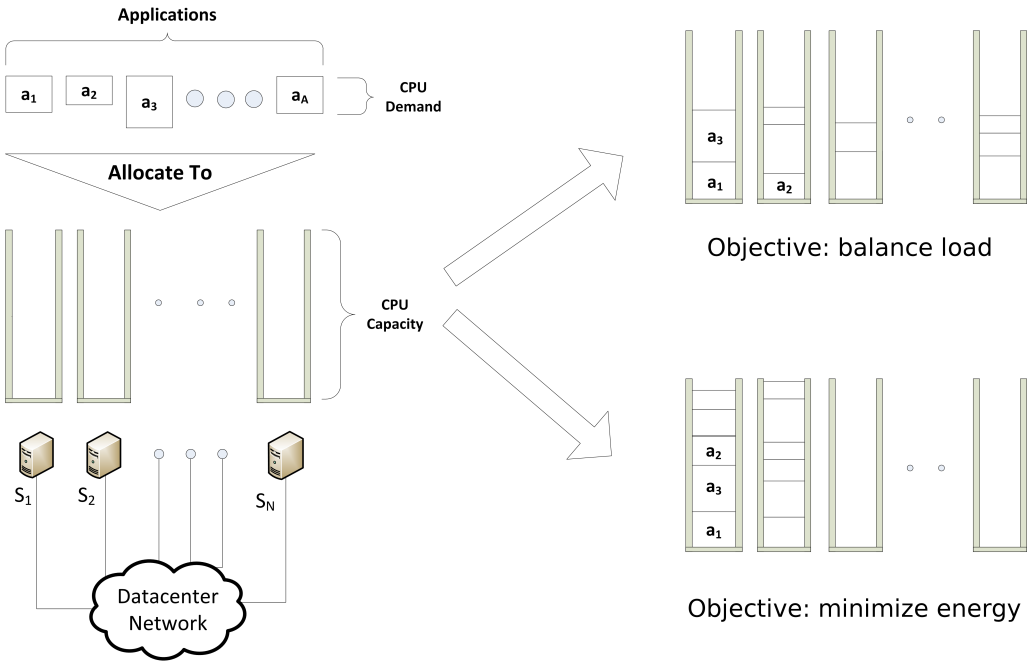

Figure 1: Distributed application placement in a datacenter environment

This document is organized as follows. Section 2 presents the formal model of the application placement process. Section 3 introduces the performance objectives considered in this project. Section 4 describes the architecture for which you develop application placement algorithm. Section 5 explains how to evaluate your algorithms, section 6 lists down the specific tasks you need to complete for this project and section 7 discusses the project deliverables.

2 A Model for Application Placement

- We model the datacenter as a set of servers N that connected by a network and communicating by message passing. The datacenter runs a set of applications A. All servers in N have the same CPU capacity Ω.
- Each application $a \in A$ has a time-dependent CPU resource demand which we denote by $\omega_a(t)$.
- The *placement* of the applications in the entire cloud is defined in terms of the placement on each server. We use $A_n(t)$ represents is the set of applications placed to run on server n at time t. In a valid placement, every application $a \in A$ is placed on one and only one server.

3 Performance Objectives for the Application Placement Process

For the problem of application placement in a datacenter setting, we consider combining two performance objectives: *load balancing* and *minimization of energy consumption*.

3.1 The Load Balancing Objective

For the model described in section 2, the problem of application placement with the objective of *load balancing* can be formulated as follows:

- Given the demand of applications $\omega_{a_1}(t), \omega_{a_2}(t), \ldots, \omega_{a_A}(t)$ at time t, the problem is to find a placement set $A_n(t)$ for each server n such that variance of CPU utilization over all servers, defined as $\omega^n = \frac{1}{\Omega}\sum_{a \in A_n} \omega_a$, is minimal.
- Whenever the demand of the applications significantly changes, a new placement is computed in order to keep the variance of the CPU demand minimized. Typically, there are many possible placements that minimize the variance of the total CPU demand. In such a case, we choose the placement that minimizes the *cost of reconfiguration*, which we define as the total number of new applications started as a result of the new placement.

This performance objective is the most common one in today's cloud environment as it allows the provider to handle sudden demand spikes better.

3.2 Energy Efficiency Objective

For the model described in section 2, the problem of application placement with the objective of energy efficiency can be formulated as follows:

- Given the demand of applications $\omega_{a_1}(t), \omega_{a_2}(t), \ldots, \omega_{a_A}(t)$ at time t, the problem is to find a placement set $A_n(t)$ for every server n such that (1) servers are not overloaded, i.e., $\sum_{a \in A_n} \omega_a(t) < \Omega$, and (2) the number of servers running applications is minimal.
- Similar to subsection 2, we choose a placement that minimizes the cost of reconfiguration.

The process of using as few servers as possible to run a given workload is called 'server consolidation'. By powering down servers that are not running applications, the datacenter can minimize the energy consumption of its servers [11].

3.3 Combining the Two Objectives

The goal in this project is to develop a scalable algorithm for application placement with the objective that:

- load balancing is enforced when the average server utilization is above a threshold τ.
- energy efficiency objective is enforced otherwise.

4 Distributed Application Placement

4.1 System Architecture

We use a simple *distributed* architecture for the resource management system of an IaaS cloud. In this architecture, every server of the datacenter runs a *Resource Manager* component responsible for computing the placement of applications on that server. Resource Manager has two subcomponents: the *Application Placement Manager* and the *Overlay Manager*. The Overlay Manager is responsible for providing a peer sampling service, a service that allows one server to randomly select other servers in the cloud. The Placement Manager takes as input the existing placement of the server and computes a new placement for the server through interacting with the Placement Managers of other servers in the datacenter. The placement, computed in a distributed way, should satisfy the performance objectives set by the cloud provider.

4.2 A Template for Distributed Application Placement Algorithms

For this project, you develop a gossip-based solution for the problem of application placement. Your solution should be developed by extending the template provided in algorithm 1. The algorithm is adapted from the gossip-based resource management algorithms presented in [10, 11]. The template partially implements the distributed application placement algorithm that runs in the Application Placement Manager component

explained in previous section. As can be seen from algorithm 1, the algorithm relies only on local information on each node and the messages it receives from its peers. The algorithm follows the push-pull interaction model [4, 5]. In this model, two nodes exchange and update states based on the state of the other node during an execution of a protocol round. This scheme can be implemented using a threads whereby each node runs an active and a passive thread. Periodically, the active thread sends a message to a random peer and waits for a response. The passive thread on the peer receives the message and replies with its local information. We assume that there is a synchronization mechanism that prevents simultaneous access to protocol states between the active and passive threads.

4.3 *updatePlacement*($A_{n'}$)

Algorithm 1 Template for the distributed Application Placement Algorithm. Code for server n

initialization:

```
1: Read A_n
2: Start the passive and active threads
```

active thread:

```
1: for r = 1 to r_max do
2:    n' ← choose uniformly at random from neighbors of n;
3:    if n' is not active then
4:       continue;
5:    end if
6:    send(n', A_n);
7:    A_n' = receive(n');
8:    updatePlacement(A_n');
9:    sleepUntilNextCycle();
10: end for
11: Start or stop applications according to A_n.
```

passive thread:

```
1: while true do
2:    A_n' = receive(n');
3:    send(n', A_n);
4:    updatePlacement(A_n');
5: end while
```

updatePlacement($A_{n'}$)

```
1: ############
2: Your solution here!
3: ############
```

updatePlacement($A_{n'}$) is the core function in this template and it implements a way of updating the node's local state after an interaction with a node n'. The updating of states should be such that, if one looks at the placement on all servers, it evolves into a placement that achieves the performance objective of the datacenter. This function is shared by both active and passive threads for making a placement decision and it basically moves applications from one server to another. Figure 2 shows an example of what could happen during an execution of *updateplacement*() on two nodes n and j for some specific objective.

4.4 Execution Model

Generally, gossip algorithms are round-based iterative algorithms. For properly designed algorithms, the execution of a round of the gossip protocol (on all nodes) improves the (global) state of the algorithm. In

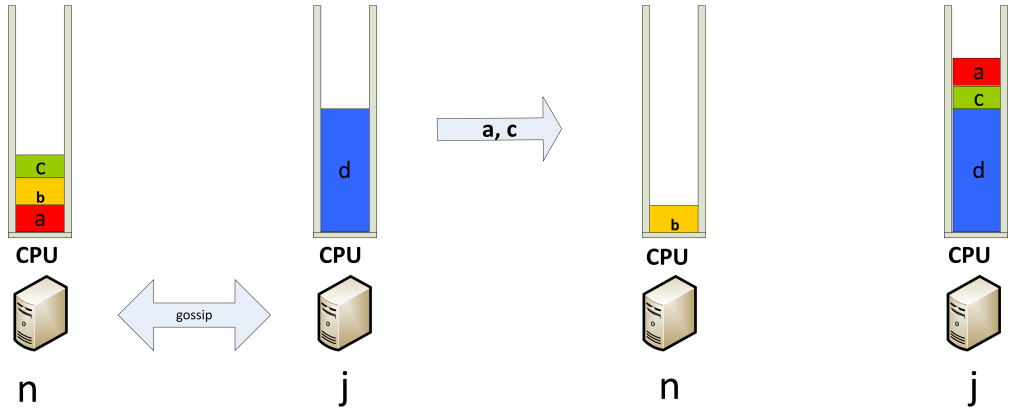

Figure 2: A possible execution of *updatePlacement*() on two interacting nodes n and j. Boxes labeld a to d are applications, with their height corresponding to their CPU demand.

other words, for some appropriate measure of distance between the current state of the algorithm and the goal state of the protocol, each (global) execution of a round of the gossip algorithm reduces this distance. In other words, the state of the gossip protocol *converges* to the desired state with increasing number of executed rounds. In general, the distance never becomes zero. This means that one has to stop the gossip protocol after executing a specific number of rounds. For this project we denote this number by r_{max}.

Figure 3 shows a diagram with the execution model for the protocol. The demand of the applications significantly changes at discrete points in time. When this occurs, the application placement algorithm reads in the current placement and the new demands of all applications, and it proceeds to execute r_{max} rounds. At the end of the r_{max} rounds, the computed configurations are implemented and the resource allocation process waits until the application demands change significantly. We call the period between two application demand changes an *epoch*.

4.5 Peer Sampling Service

In general, gossip protocols depend on the ability to pick a node uniformly at random from the set of all nodes. This ability can be approximated by running a peer sampling service. A protocol that implements such a service is CYCLON[9] which you will implement in this project.

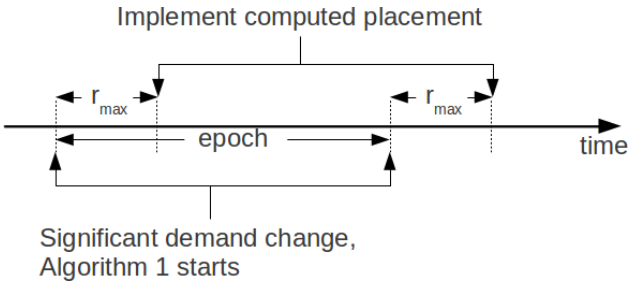

Figure 3: The execution model for the resource allocation protocol.

5 Evaluation through Simulation

After developing your protocol for resource allocation, you evaluate its performance through simulations. The simulation platform that will be used is called PeerSim [4]. All simulations will be implemented using the *cycle driven* model which assumes that communication and processing delays can be neglected.

5.1 Simulation Parameters

- Number of servers : 10,000
- CPU capacity: 100 units
- Peer sampling service: CYCLON [9]
- CYCLON cache size: 20
- CYCLON shuffle length: 10
- $r_{max} = 30$

5.2 Performance Metrics

Use the following metrics to quantify the performance of the placement protocol:

- fraction of *Overloaded Servers* (S): a server n is overloaded if $\sum_{a \in A_n} \omega_a > \Omega$.
- *Coefficient of variation of server CPU utilization* (V): CPU utilization is defined for server n as $\frac{1}{\Omega} \sum_{a \in A_n} \omega_a$ while the coefficient of variation is defined as the ratio of the standard deviation to the average.
- *Cost of reconfiguration (C)*: fraction of applications started on a new server.
- *Resource Consumption (R)*: fraction of active servers.

5.3 Simulation Scenarios

The simulator package available together with the description of this project contains code for initialization and generation of application demand. The demand generation process is parameterized by a value f: the ratio of the expected application demand to the server capacity Ω.

Three sets of simulation scenarios are considered in this project. These are overload scenarios, underload scenarios, and changing load scenarios. For overload scenarios, the parameters (shown in table 1) are chosen such that the total demand is 140% of the total capacity. For underload scenarios, the parameters (shown in table 2) are selected that the number of applications is constant while the expected application demand is varied, resulting in total demand ranging from ∼6% to 100%.

No.	f	Number of applications
1	1.4/32	320,000
2	1.4/16	160,000
3	1.4/8	80,000
4	1.4/4	40,000
5	1.4/2	20,000

Table 1: Simulation parameters for overload scenario

No.	f	Number of applications
1	1.0/64	40,000
2	1.0/32	40,000
3	1.0/16	40,000
4	1.0/8	40,000
5	1.0/4	40,000

Table 2: Simulation parameters for underload scenario

For changing load scenario, the number of applications is set to 40000 while the total demand increases linearly with epochs, from 10% to 150% over 100 epochs.

6 Project Tasks

Your project will be evaluated based on how well you complete the following tasks. To guide you through the development process, you will be provided with a zip file containing all code templates. Your code should be implemented in the package "peersim.EP2400.resourcealloc.tasks". You are free to add additional classes to the package, but you are not allowed to change the other packages of the simulator.

6.1 Task 1: Implementing CYCLON and Performance Observers

1. Implement CYCLON [9] in the code template is provided to you (file name "CYCLON.java").
2. Implement Performance Observers to measure the four performance metrics discussed in section 5.2 using the template file "PerformanceObserver.java". Your code should write out the test results in two file in the folder "sim-results": "cycles.csv" should contain the metrics after execution of each cycle with the format N_{cycle}, V, S, R, where N_{cycle} is the cycle number,"epochs.csv" should contain the metrics after execution of each epoch with format N_{epoch}, V, S, R, C.

6.2 Task 2: Distributed Application Placement

Develop a solution for the distributed application placement problem described earlier. Use the template provided to you in section 4.2. $updatePlacement(A_{n'})$ is the function where you put your solution. Your algorithm and implementation should have polynomial computational complexity (i.e., you should avoid using exhaustive search algorithms). We recommend that you start with two different implementations of $updatePlacement(A_{n'})$, one for each performance objective, and then try to figure out how to combine them together.

1. Design an algorithm for the function $updatePlacement(A_{n'})$ that achieves the objective discussed in section 3.3. Start by writing the pseudo-code of your algorithm. Your algorithm must use only local information available to each server, for example, applications allocated to that server and its neighbors. Do not use global information, such as $f, |N|, |A|$.
2. Implement your algorithm in "DistributedResourceAllocation.java".
3. Evaluate the performance of your algorithm for the five overload scenarios (use the configuration file "task2-fixed-load.cfg") and measure the V and C parameters. To help in visualizing the results, produce two plots: a plot of V in function of the round number r for the first r_{max} cycles (a graph with 5 curves, one for each scenario), a plot of $\overline{V}$ and $\overline{C}$ in function of f (one graph with two curves). $\overline{V}$ and $\overline{C}$ are the values of V and C after running r_{max} rounds averaged over 50 epochs. For the second plot, include the 95% confidence intervals for your data points.
4. Continue with the underload scenarios. Use the configuration file "task2-fixed-load.cfg". From the simulation results, produce the following plots in the same manner as the above task: R in function of r for the first r_{max} cycles, $\overline{R}$, $\overline{S}$, and $\overline{C}$ in function of f.
5. Finally, run simulations for the changing load scenario. Use the configuration file "task2-changing-load.cfg". From the simulation results, produce a plot of V, R, C, S in function of epoch, for the first 100 epochs. Plot all results on the same graph.

6.3 Task 3: Analysis

Answer the following questions based on your simulation results. Relate your arguments with your plots whenever appropriate.

- Does your protocol achieve the performance objective(s)? Support your answer with the plots produced for the underload and overload cases.

- Explain what you see in the plots of the changing load scenario.
- How and why does f affect the performance of your protocol? Discuss its effect on all the performance metrics considered.
- If you were not restricted to the given template, how would you further reduce the cost of reconfiguration? (This is an open question which may require you to explore new parameters, algorithms, protocol modifications, etc.).

6.4 Task 4: Distributed Application Placement with Additional Constraint (optional)

This task is optional. This means that you do not need to complete it to pass the project. However, you need to complete it if you want to get the highest grade. It is not required for PhD students.

1. Extend your algorithm produced in section 6.2 to consider memory demand. Let's denote the memory demand of application a by γ_a. The goal is to further support the constraint that the total memory demand of applications on a server n does not exceed its memory capacity Γ_n. We consider the simplified case where $\Gamma_n = \Gamma, \forall n \in N$ and $\gamma_a = \gamma, \forall a \in A$. Write the pseudocode of your algorithm.
2. Implement your algorithm in the module "AdvancedDistributedResourceAllocation.java".
3. Evaluate the performance of your algorithm in the same way as section 6.2. Use $\Gamma = 8$GB and $\gamma = 1$GB. Use the configuration files "task4-fixed-load.cfg" and "task4-changing-load.cfg".
4. Compare and contrast the results you obtained in this task with those in section 6.2.

6.5 Task for Ph.D students

This task is mandatory for Ph.D. students. Other students are also welcome to do it, but it does not give any additional points.

Datacenters providing cloud services consume a very large amount of electric power. Some studies suggest that the cost of electricity per server over its lifetime is often larger than the cost of the server itself [3]. This means that saving on energy consumption of servers can potentially translate into significant cost savings. With this in mind, a lot of research has been done on reducing the power consumption of servers in a cloud. Below are some of these works:

1. A. J. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient resource management for cloud computing environments," in *Proceedings of the International Conference on Green Computing*. Chicago, IL: IEEE, Aug 2010.
2. C.-T. Yang, K.-C. Wang, H.-Y. Cheng, C.-T. Kuo, and W. Chu, "Green power management with dynamic resource allocation for cloud virtual machines," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, sept. 2011, pp. 726 –733.
3. A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, may 2010, pp. 826 –831.
4. Z. Gong and X. Gu, "PAC: Pattern-driven Application Consolidation for efficient cloud computing," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, aug. 2010, pp. 24 –33.
5. R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Network Operations and Management Symposium (NOMS), 2010 IEEE*, april 2010, pp. 479 –486.

6. V. Raj and R. Shriram, "Power aware provisioning in cloud computing environment," in *Computer, Communication and Electrical Technology (ICCCET), 2011 International Conference on*, march 2011, pp. 6 –11.

7. S. Dutta and A. Verma, "Service deactivation aware placement and defragmentation in enterprise clouds," in *Network and Service Management (CNSM), 2011 7th International Conference on*, oct. 2011, pp. 1 –9.

8. D. Borgetto, M. Maurer, G. Da-Costa, J.-M. Pierson, and I. Brandic, "Energy-efficient and sla-aware management of iaas clouds," in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, ser. e-Energy '12. New York, NY, USA: ACM, 2012, pp. 25:1–25:10.

Your task for this project it to choose four papers that deal with energy efficiency of datacenters (possibly outside of the above list) and produce a 4-page review that discusses the various research challenges, the approaches taken and the solutions proposed by the literature. A comparative analysis of the papers should be included. (This means that you should select the papers such that such an analysis is possible.)

7 Project Deliverables

Submit a project report and the source code of your implementations for grading. Your report should contain the plots, pseudocode and answers to the questions of the tasks you have undertaken. Your report should be sent as a PDF file (max 10 pages).

Submit the sourcecode of your project (including configuration files, but excluding data files) in a single compressed file. Your source code should be well commented. It is crucial that your implementation follows the guidelines for making your simulation results reproducible. This means that all random number generators functions should use the default PeerSim random seed 1234567890.

7.1 Interview

After the project is submitted, you will step by for an interview. Schedule for the interview will be made available after the project is started.

References

[1] Amazon. `https://aws.amazon.com/ec2`. Last Accessed: September, 2011.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing, Feb 2009.

[3] C. L. Belady. In the data center, power and cooling costs more than the it equipment it supports. *Electronics Cooling*, (1), 2007.

[4] M. Jelasity, A. Montresor, and O. Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In *In Engineering Self-Organising Systems, G. Di Marzo Serugendo*, volume 2977, pages 265–282, 2004. `http://www.inf.u-szeged.hu/~jelasity/cikkek/esoa03.pdf`.

[5] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23:219–252, August 2005. `http://www.inf.u-szeged.hu/~jelasity/cikkek/tocs04.pdf`.

[6] OpenNebula Project Leads. `http://opennebula.org/`.

[7] Openstack. `http://www.openstack.org/`. Last Accessed: September, 2011.

[8] Rackspace. http://www.rackspace.com/. Last Accessed: September, 2011.

[9] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005. http://www.few.vu.nl/~spyros/papers/Cyclon.pdf.

[10] F. Wuhib, R. Stadler, and M. Spreitzer. Gossip-based resource management for cloud environments. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 1 –8, oct. 2010.

[11] R. Yanggratoke, F. Wuhib, and R. Stadler. Gossip-based resource allocation for green computing in large clouds. In *Network and Service Management (CNSM), 2011 7th International Conference on*, pages 1 –9, oct. 2011.