Cooperative Dynamic Scheduling of Virtual Machines in Distributed Systems

Flavien Quesnel and Adrien Lèbre

ASCOLA Research Group, Ecole des Mines de Nantes/INRIA/LINA, Nantes, France
firstname.lastname@mines-nantes.fr

Abstract. Cloud Computing aims at outsourcing data and applications hosting and at charging clients on a per-usage basis. These data and applications may be packaged in virtual machines (VM), which are themselves hosted by nodes, i.e. physical machines.

Consequently, several frameworks have been designed to manage VMs on pools of nodes. Unfortunately, most of them do not efficiently address a common objective of cloud providers: maximizing system utilization while ensuring the quality of service (QoS). The main reason is that these frameworks schedule VMs in a static way and/or have a centralized design.

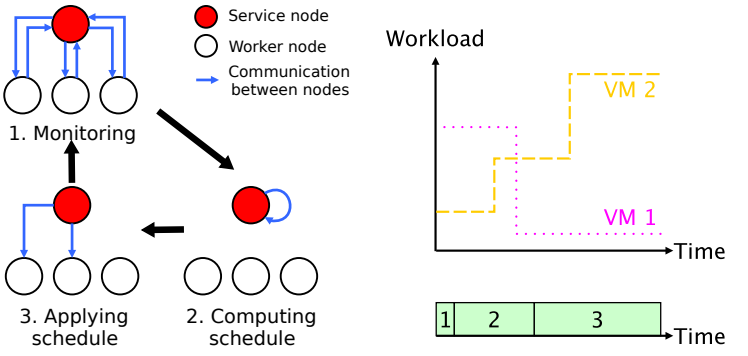
In this article, we introduce a framework that enables to schedule VMs cooperatively and dynamically in distributed systems. We evaluated our prototype through simulations, to compare our approach with the centralized one. Preliminary results showed that our scheduler was more reactive. As future work, we plan to investigate further the scalability of our framework, and to improve reactivity and fault-tolerance aspects.

1 Introduction

Scheduling jobs has been a major concern in distributed computer systems. Traditional approaches rely on batch schedulers [2] or on distributed operating systems (OS) [7]. Although batch schedulers are the most deployed solutions, they may lead to a suboptimal use of resources. They usually schedule processes statically – each process is assigned to a given node and stays on it until its termination – according to user requests for resource reservations, that may be overestimated. On the contrary, preemption mechanisms were developed for distributed OSes to make them schedule processes dynamically, in line with their effective resource requirements. However, these mechanisms were hard to implement due to the problem of residual dependencies [1].

Using system virtual machines (VM) [14], instead of processes, allows to perform dynamic scheduling of jobs while avoiding the issue of residual dependencies [4,12]. However, some virtual infrastructure managers (VIM) still schedule VMs in a static way [6,10]; it conflicts with a common objective of virtual infrastructure providers: maximizing system utilization while ensuring the quality of service (QoS). Other VIMs implement dynamic VM scheduling [5,8,15], which enables a finer management of resources and resource overcommitment. However,

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part II, LNCS 7156, pp. 457–466, 2012.



(a) Scheduling steps

(b) Workload fluctuations during scheduling

Fig. 1. Scheduling in a centralized architecture

they often rely on a centralized design, which prevents them to scale and to be reactive. Scheduling is indeed an NP-hard problem, the time needed to solve it grows exponentially with the number of nodes and VMs considered. Besides, it takes time to apply a new schedule, because manipulating VMs is costly [4]. During the computation and the application of a schedule (cf. Fig. 1(a)), centralized managers do not enforce the QoS anymore, and thus cannot react quickly to QoS violations. Moreover, the schedule may be outdated when it is eventually applied if the workloads have changed (cf. Fig. 1(b)). Finally, centralization can lead to fault-tolerance issues: VMs may not be managed anymore if the master node crashes, as it is a single point of failure (SPOF). Considering all the limitations of centralized solutions, more decentralized ones should be investigated. Indeed, scheduling takes less time if the work is distributed among several nodes, and the failure of a node does not stop the scheduling anymore.

Several proposals have been made precisely to distribute dynamic VM management [3,13,17]. However, the resulting prototypes are still partially centralized. Firstly, at least one node has access to a global view of the system. Secondly, several VIMs consider all nodes for scheduling, which limits scalability. Thirdly, several VIMs still rely on service nodes, that are potential SPOFs.

In this paper, we introduce a VIM that enables to schedule and manage VMs cooperatively and dynamically in distributed systems. We designed it to be non-predictive and event-driven, to work with partial views of the system, and to require no SPOF. We made these choices for the VIM to be reactive, scalable and fault-tolerant. In our proposal, when a node cannot guarantee the QoS for its hosted VMs or when it is under-utilized, it starts an iterative scheduling procedure (ISP) by querying its neighbor to find a better placement. If the request cannot be satisfied by the neighbor, it is forwarded to the following one until the ISP succeeds. This approach allows each ISP to consider a minimum number of nodes, thus decreasing the scheduling time, without requiring a central point. In addition, several ISPs can occur independently at the same moment throughout the infrastructure, which significantly improves the reactivity of the system. It

should be noted that nodes are reserved for exclusive use by a single ISP, to prevent conflicts that can occur if several ISPs do concurrent operations on the same nodes or VMs. In other words, scheduling is performed on partitions of the system, that are created dynamically. Moreover, communication between nodes is done through a fault-tolerant overlay network, which relies on distributed hash table (DHT) mechanisms to mitigate the impact of a node crash [9]. We evaluated our prototype by means of simulations, to compare our approach with the centralized one. Preliminary results were encouraging and showed that our scheduler was reactive even if it had to manage several nodes and VMs.

The remainder of this article is structured as follows. Section 2 presents related work. Section 3 gives an overview of our proposal, while Sect. 4 details its implementation and Sect. 5 compares it to a centralized proposal [5]. Finally, Sect. 6 discusses perspectives and Sect. 7 concludes this article.

2 Related Work

This section presents some work that aim at distributing resource management, especially those related to the dynamic scheduling of VMs. Contrary to previous solutions that performed scheduling periodically, recent proposals tend to rely on an event-based approach: scheduling is started only if an event occurs in the system, for example if a node is overloaded.

In the DAVAM project [16], VMs are dynamically distributed among managers. When one VM has not enough resources, its manager tries to relocate it by considering all resources of the system (the manager builds this global view by communicating with its neighbors).

Another proposal [13] relies on peer-to-peer networks. It is very similar to the centralized approaches, except that there is no service node, so that it is more fault-tolerant. When an event occurs on a node, this node collects monitoring information on all nodes, finds which nodes can help it to fix the problem, and performs appropriate migrations.

A third proposition [17] relies on the use of a service node that collects monitoring information on all worker nodes. When an event occurs on a worker node, this node retrieves information from the service node, computes a new schedule and performs appropriate migrations. This approach does not consider fault-tolerance issues.

Snooze [3] has a hierarchical design: nodes are dynamically distributed among managers, a super manager oversees managers and has a global view of the system. When an event occurs, it is processed by a manager that considers all nodes it is in charge of. Snooze design is close to the Hasthi [11] one; the main difference is that Snooze targets virtualized systems and single system images, while Hasthi is presented to be system agnostic.

3 Proposal Overview

In this section, we describe the theoretical foundations of our proposal. After giving its main characteristics, we explain shortly how it works.

3.1 Main Characteristics

Reactivity, scalability and fault-tolerance are desired properties to make a VIM with a better QoS management.

Keeping that in mind, we made the VIM follow an event-based approach. In this context, scheduling is started only when it is required, on the reception of events, leading to better reactivity. This contrasts with more traditional solutions where scheduling is started periodically. This also differs from a predictive approach, where new schedules are computed in advance to anticipate workload fluctuations; this kind of approach requires knowledge on workload profiles, which is not always possible.

An event may be generated each time a virtualized job (vjob) [4] is submitted or terminates, when a node is overloaded or underloaded, or when a system administrator wants to put a node into maintenance mode.

Besides relying on events, our VIM is comparable to peer-to-peer systems.

There is no service node, all nodes are equal. Each node can (i) be used to submit vjobs, (ii) generate events and (iii) try to solve events generated by other nodes.

A node monitors only its local resources. However, it can get access on-demand to a partial view of the system by communicating with its neighbors by means of an overlay network similar to those used to implement distributed hash tables. To facilitate understanding, we consider that the communication path is a ring. Accessing a partial view of the system improves scalability (computing and applying a schedule is faster) while the DHT mechanisms enhance fault-tolerance (the nodes can continue to communicate transparently even if several of them crash).

3.2 The Iterative Scheduling Procedure

When a node N_i retrieves its local monitoring information and detects a problem (e.g. it is overloaded), it starts a new iterative scheduling procedure by generating an event, reserving itself for the duration of this ISP, and sending the event to its neighbor, node N_{i+1} (cf. Fig. 2).

Node N_{i+1} reserves itself, updates node reservations and retrieves monitoring information on all nodes reserved for this ISP, i.e. on nodes N_i and N_{i+1}. It then computes a new schedule. If it fails, it forwards the event to its neighbor, node N_{i+2}.

Node N_{i+2} performs the same operations as node N_{i+1}. If the computation of the new schedule succeeds, node N_{i+2} applies it (e.g. by performing appropriate VM migrations) and finally cancels the reservations, so that nodes N_i, N_{i+1} and N_{i+2} are free to take part in another ISP.

Considering that a given node can take part only in one of these iterative scheduling procedures at a time, several ISPs can occur simultaneously and independently throughout the infrastructure, thus improving reactivity.

Note that if a node receives an event while it is reserved, it just forwards it to its neighbor.

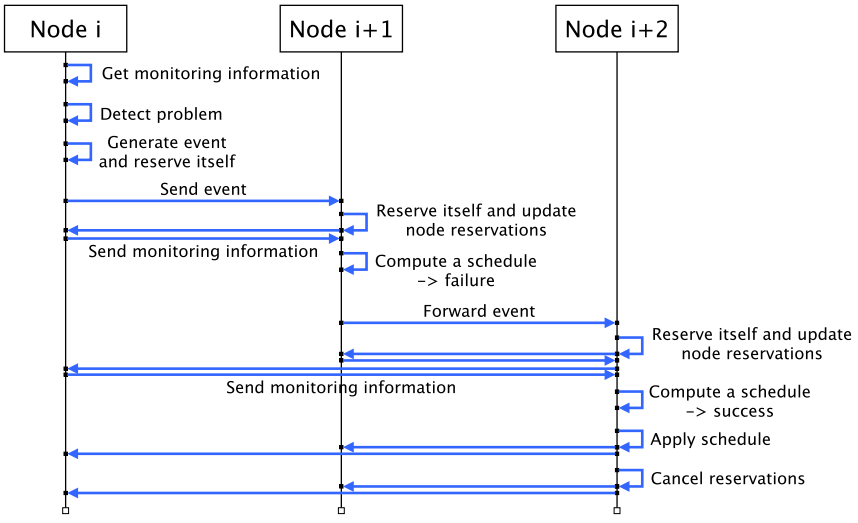


Fig. 2. Iterative scheduling procedure

4 Implementation

4.1 Current State

We implemented our proposal in Java. The prototype can currently process 'overloaded node' and 'underloaded node' events; these events are defined by means of CPU and memory thresholds by the system administrator. Moreover, the overlay network is a simple ring (cf. Fig. 3) without any fault-tolerance mechanism, i.e. it cannot recover from a node crash. Furthermore, the prototype manipulates virtual VMs, i.e. Java objects.

4.2 Node Agent

The VIM is composed of node agents (NA).

There is one NA on every node, each NA being made of a knowledge base, a resource monitor, a client, a server and a scheduler (cf. Fig. 3).

The knowledge base contains various types of information. Some information is available permanently: monitoring information about the local node (resources consumed and VMs hosted), a stub to contact the neighbor, and a list of events generated by the node. Other information is accessible only during an iterative scheduling procedure: monitoring information about the nodes reserved (if a scheduler is running on the node) and a stub to contact the scheduler that tries to solve the event.

The resource monitor retrieves node monitoring information periodically and updates the knowledge base accordingly. If it detects a problem (e.g. the node is overloaded), it starts a new ISP by generating an event, reserving the node for this ISP and sending the event to the neighbor by means of a client.

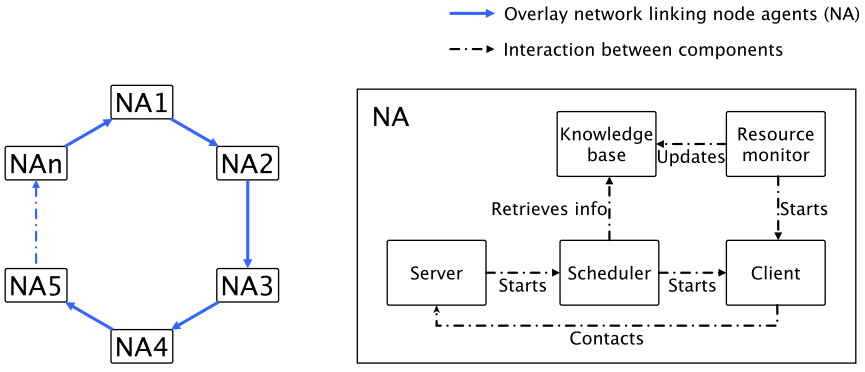


Fig. 3. Implementation overview

A client is instantiated on-demand to send a request or a message to a server.

The server processes requests and messages from other nodes. In particular, it launches a scheduler when it receives an event.

The scheduler first retrieves monitoring information from the nodes taking part in an ISP. It then tries to solve the corresponding event by computing a new schedule and applying it, if possible. If the schedule is applied successfully, the scheduler finally cancels node reservations. The prototype is designed so that any dynamic VM scheduler may be used to compute and apply a new schedule. Currently, the prototype relies on Entropy [5], with consolidation as the default scheduling policy.

5 Experiments

We compared our approach with the Entropy [5] one by means of simulation. Basically, the simulator injected a random CPU workload into each virtual VM and waited until the VIM solves all 'overloaded node' issues. Comparison criteria included the average time to solve an event, the time elapsed since the load injection until all 'overloaded node' issues are solved, and the cost of the schedule to apply. This cost is related to the kind of actions to perform on VMs (e.g. migrations) and to the amount of memory allocated to the VMs that are manipulated [5].

The experiments were done on a HP Proliant DL165 G7 with 2 CPUs (AMD Opteron 6164 HE, 12 cores, 1.7 GHz) and 48 GB of RAM. The software stack was composed of Debian 6/Squeeze, Sun Java VM 6 and Entropy 1.1.1. The simulated nodes had 2 CPUs (2 GHz) and 4 GB of RAM. The simulated VMs had 1 virtual CPU (2 GHz) and 1 GB of RAM. The virtual CPU load could take only one of the following values (in percentage): 0, 20, 40, 60, 80, 100. Entropy has timeouts to prevent it to spend too much time computing a new schedule; these timeouts were set to twice the number of nodes considered (in seconds). Our VIM considers that a node is overloaded if the VMs hosted try to consume

more than 100% of CPU or RAM; it is underloaded if less than 20% of CPU and less than 50% of RAM are used.

As we can see on Table 1, our VIM is more reactive, i.e. it quickly solved individual events, especially the 'overloaded node' ones. This can be explained by the fact that our VIM generally considers a few number of nodes, compared to Entropy. This leads to a smaller cost for applying schedules.

Table 1. Experimental results

		128 VM / 64 nodes		256 VM / 128 nodes	
		DVMS	Entropy	DVMS	Entropy
Iteration length (s) (time between two iterations)	Avg	83	198	114	475
	Std dev	41	56	82	37
	Max	232	240	427	489
Time to solve an event (s)	Avg	12	N/A	12	N/A
	Std dev	18	N/A	19	N/A
	Max	149	N/A	299	N/A
Time to solve an overloaded node event (s)	Avg	6	N/A	6	N/A
	Std dev	12	N/A	12	N/A
	Max	52	N/A	48	N/A
Number of nodes considered (partition size)	Avg	8	64	10	128
	Std dev	8	0	14	0
	Max	60	64	115	128
Maximum cost for applying the schedule (arbitrary unit)	Avg	7134	24405	8479	39977
	Std dev	2690	12798	2756	20689
	Max	13312	49152	18432	87040
Percentage of nodes hosting VMs (%)	Avg	55	53	54	53
	Std dev	2	2	2	2
	Max	58	58	59	59

(Distributed VM Scheduler vs Entropy Centralized approach)
Avg: average values, Std dev: standard deviation, Max: worst case

In details, the first row shows the iteration length that corresponds to the required time to solve all events occurring during one iteration. The second row gives the time to solve one event. That is the time between the event appearance and its resolution. The third row focuses on overloaded events. These events refer to QoS violations and must be solved as quickly as possible. For these two rows, we do not mention the values of the centralized approach since it relies on a periodic scheme: Entropy monitors the configuration at the beginning of the iteration, analyzes the configuration and applies the schedule at the end. The fourth row shows the size of each partition: i.e. the number of nodes considered for a scheduling. As we can see on the fifth row, the smaller the partition is, the cheaper is the reconfiguration cost. However, it is worth nothing that the values for the Entropy approach, as previously, consider the total cost for the whole iteration whereas the cost of the reconfiguration related to one event is considered for the DVMS approach. As a consequence, the sum of each reconfiguration in

the DVMS approach can be higher than the cost corresponding of the Entropy one. However, since we are trying to solve each event as soon as possible, we are not interested by the global cost but by the cost for one event. Finally, the last row presents the consolidation rate, which is the percentage of nodes hosting at least one VM. We can see that, despite the fact that our approach is more reactive, it does not impact negatively the consolidation rate.

6 Future Work

Several ways should be explored to improve the prototype, with regard to event management, fault-tolerance and network topology.

Event Management. Event management could be enhanced by merging iterative solving procedures, rethinking event definition and implementing other kinds of events.

Using ISPs can result in deadlocks, as they rely on dynamic partitions of the system. A deadlock occurs when each node belongs to a partition and all partitions need to grow, i.e. each ISP needs more nodes to solve the corresponding event. Deadlocks can be resolved by merging ISPs, which implies to merge the related events and partitions. A basic algorithm was implemented to do that, but it will not be detailed in this article due to space limitations.

ISP merging can also be used to combine complementary events (e.g. an 'overloaded node' event with an 'underloaded node' one) to make ISPs converge faster, thus increasing reactivity.

'Overloaded node' and 'underloaded node' events are currently defined by means of CPU and memory thresholds. It may not be always relevant. For example, if a load balancing policy is used while the global load is low, many nodes will send 'underloaded node' events that cannot be solved. Refining event definition by taking the neighbors' load into account may be a solution.

Other kinds of events should be implemented, like those related to vjob submissions or terminations, or to a node that is put into maintenance mode. Moreover, it may be interesting to take other resources than CPU and memory into account, like network bandwidth.

Fault-Tolerance. Currently, the VIM is not fault-tolerant: if a node crashes, it breaks the overlay network. This can be fixed with mechanisms used in DHTs [9].

Network Topology. The current prototype does not take the network topology into account. However, the knowledge of network bandwidth between each pair of nodes could lead to faster migrations in a heterogeneous system.

7 Conclusion

In this article, we proposed a new approach to schedule VMs dynamically and cooperatively in distributed systems, keeping in mind the following objective: maximizing system utilization while ensuring the quality of service.

We presented the current state of implementation of a prototype and we evaluated it by means of simulations, to compare our approach with the centralized one. Preliminary results were encouraging and showed that our solution was more reactive and scalable.

On-going work has focused on performing larger-scale simulations and on evaluating the prototype with real VMs. Future work will be done with regard to event management, fault-tolerance and network topology. This work fits into a broader project that seeks to implement a framework for managing VMs in distributed systems the same way an OS manages processes on a local machine.

Acknowledgments. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

1. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: NSDI 2005: Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation, NSDI 2005, pp. 273–286. USENIX Association, Berkeley (2005)
2. Etsion, Y., Tsafrir, D.: A Short Survey of Commercial Cluster Batch Schedulers. Tech. rep., The Hebrew University of Jerusalem, Jerusalem, Israel (May 2005)
3. Feller, E., Rilling, L., Morin, C., Lottiaux, R., Leprince, D.: Snooze: A Scalable, Fault-Tolerant and Distributed Consolidation Manager for Large-Scale Clusters. Tech. rep., INRIA Rennes, Rennes, France (September 2010)
4. Hermenier, F., Lebre, A., Menaud, J.M.: Cluster-Wide Context Switch of Virtualized Jobs. In: VTDC 2010: Proceedings of the 4th International Workshop on Virtualization Technologies in Distributed Computing. ACM, New York (2010)
5. Hermenier, F., Lorca, X., Menaud, J.M., Muller, G., Lawall, J.: Entropy: a consolidation manager for clusters. In: Hosking, A.L., Bacon, D.F., Krieger, O. (eds.) VEE 2009: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 41–50. ACM, New York (2009)
6. Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., Good, J.: On the Use of Cloud Computing for Scientific Workflows. In: ESCIENCE 2008: Proceedings of the 2008 Fourth IEEE International Conference on eScience, pp. 640–645. IEEE Computer Society, Washington, DC (2008)
7. Lottiaux, R., Gallard, P., Vallee, G., Morin, C., Boissinot, B.: OpenMosix, OpenSSI and Kerrighed: a comparative study. In: CCGRID 2005: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid, vol. 2, pp. 1016–1023. IEEE Computer Society, Washington, DC (2005)
8. Lowe, S.: Introducing VMware vSphere 4, 1st edn. Wiley Publishing Inc., Indianapolis (2009)
9. Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z.: Peer-to-Peer Computing. Tech. rep., HP Laboratories, Palo Alto, CA, USA (July 2003)

10. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: The Eucalyptus Open-Source Cloud-Computing System. In: Cappello, F., Wang, C.L., Buyya, R. (eds.) CCGRID 2009: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 124–131. IEEE Computer Society, Washington, DC (2009)
11. Perera, S., Gannon, D.: Enforcing User-Defined Management Logic in Large Scale Systems. In: Services 2009: Proceedings of the 2009 Congress on Services - I, pp. 243–250. IEEE Computer Society, Washington, DC (2009)
12. Quesnel, F., Lebre, A.: Operating Systems and Virtualization Frameworks: From Local to Distributed Similarities. In: Cotronis, Y., Danelutto, M., Papadopoulos, G.A. (eds.) PDP 2011: Proceedings of the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, pp. 495–502. IEEE Computer Society, Los Alamitos (2011)
13. Rouzaud-Cornabas, J.: A Distributed and Collaborative Dynamic Load Balancer for Virtual Machine. In: Guarracino, M.R., Vivien, F., Träff, J.L., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Di Martino, B., Alexander, M. (eds.) Euro-Par-Workshop 2010. LNCS, vol. 6586, pp. 641–648. Springer, Heidelberg (2011)
14. Smith, J.E., Nair, R.: The Architecture of Virtual Machines. Computer 38(5), 32–38 (2005)
15. Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I.: Virtual Infrastructure Management in Private and Hybrid Clouds. IEEE Internet Computing 13(5), 14–22 (2009)
16. Xu, J., Zhao, M., Fortes, J.A.B.: Cooperative Autonomic Management in Dynamic Distributed Systems. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 756–770. Springer, Heidelberg (2009)
17. Yazir, Y.O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., Coady, Y.: Dynamic Resource Allocation in Computing Clouds Using Distributed Multiple Criteria Decision Analysis. In: Cloud 2010: IEEE 3rd International Conference on Cloud Computing, pp. 91–98. IEEE Computer Society, Los Alamitos (2010)