Cluster Comput (2017) 20:909–924
DOI 10.1007/s10586-016-0700-8

An approach for scaling cloud resource management

Dan C. Marinescu[1] · **Ashkan Paya**[1] · **John P. Morrison**[2] · **Stephen Olariu**[3]

Received: 30 May 2016 / Revised: 15 August 2016 / Accepted: 18 November 2016 / Published online: 27 January 2017

Abstract Given its current development trajectory, the complexity of cloud computing ecosystems are evolving to where traditional resource management strategies will struggle to remain fit for purpose. These strategies have to cope with ever-increasing numbers of heterogeneous resources, a proliferation of new services, and a growing user-base with diverse and specialized requirements. This growth not only significantly increases the number of parameters needed to make good decisions, it increases the time needed to take these decisions. Consequently, traditional resource management systems are increasingly prone to poor decisions making. Devolving resources management decisions to the local environment of that resource can dramatically increase the speed of decisions making; moreover, the cost of gathering global information can thus be eliminated; saving communication costs. Experimental data, provided in this paper, illustrate that extant cloud deployments can be used as effective vehicles for devolved decision making. This finding strengthens the case for the proposed paradigm shift, since it does not require a change to the architecture of existing cloud systems. This shift would result in systems in which resources decide for themselves how best they can be used. This paper takes this idea to its logical conclusion and proposes a system for supporting self-managing resources in cloud environments. It introduces the concept of coalitions, consisting of collaborating resources, formed for the purpose of service delivery. It suggests the utility of restricting the interactions between the end-user and the cloud service provider to a well-defined services interface. It shows how clouds can be considered functionally, as engines for delivering an appropriate set of resources in response to service requests. And finally, since modern applications are increasingly constructed from sophisticated workflows of complex components, it shows how combinatorial auctions can be used to effectively deliver packages of resources to support those workflows.

Keywords Computer clouds · Self-organization · Over-provisioning · Coalition formation · Combinatorial auctions

✉ Ashkan Paya
apaya@cs.ucf.edu

Dan C. Marinescu
dcm@cs.ucf.edu

John P. Morrison
j.morrison@cs.ucc.ie

Stephen Olariu
solariu@odu.edu

[1] Department of Computer Science, University of Central Florida, Orlando, FL, USA

[2] Computer Science Department, University College Cork, Cork, Ireland

[3] Computer Science Department, Old Dominion University, Norfolk, VA, USA

1 Introduction and motivation

The ready availability of large numbers of powerful, and increasingly heterogeneous, resources being made available by cloud service providers (the provider) is making possible the deployment of large, data and compute intensive, applications. In many cases, these, quite often legacy, applications are monolithic in construction and require bespoke execution environments. Consequently, it can be challenging to deploy them in the cloud without acquiring infrastructure as a service (IaaS) and employing specialized engineering knowledge. In this cloud usage model, the provider has no control over the effective utilization of resources, nor has cloud customers

(the customer) an incentive to engage in costly customization to increase resource efficiency when, regardless of the efficiency achieved, they are paying for the entire resource.

Composing cloud services from work flows of large, possibly legacy, applications will most likely be the trend as support for emerging Big Data applications require sophisticated, multi-phase data processing. Being essentially independent, the required resources for the applications run in each of these phases may differ greatly in number and type, and hence the problems of cost and efficiency could be significantly exacerbated.

Clearly, an approach is needed to allow the sophistication of the cloud, as a compute service, to evolve in an efficient and cost-effective manner. The approach proposed here, begins by assuming the existence of a clear and distinct services interface between the customer and the provider. It assumes that this service can be expressed as a work flow in which nodes represent extant applications and edges distinguish the phases of the service where particular applications are active. The assumption of this services interface drastically alters the current cloud usage model in that it shifts the burden of resource discovery, provisioning, and deployment from the customer to the provider. This shift greatly reduces the cost to, and the level of expertise needed by, the customer while simultaneously giving the provider full control over and affords opportunities for the efficient use of, the cloud resources.

The downside of this shift for the provider is the huge increase in resource management complexity that it precipitates, as the provider has to essentially take on all the tasks that heretofore were undertaken by each customer individually.

In this paper, a resource management approach is proposed to address this complexity. In an attempt to minimize the overhead of resource management, it suggests constraints on cloud system organization to further the goal of making decisions in a distributed manner, based only on local information. It embraces market-based mechanisms as a vehicle for implementing cloud resource management policies and it considers how far this approach can be extended in the direction of cloud self-management. Market-based mechanisms have proven ability to respond well in environments where dynamic resource management is needed including the auctioning of airport takeoff and landing slots, allocating wireless spectrum licenses, and industrial procurement.

A realistic model of the physical organization of the cloud data center, the warehouse scale computer (WSC), such as the one used by Google and described in [1] is used as the basis of our study. The entire cloud infrastructure is assumed to consist of several WSCs containing several hundreds of thousands of servers interconnected by a hierarchy of networks. These servers are housed in racks that are connected Gigabit Ethernet switches. Each switch has two to eight uplinks which go to the higher level switches in the network hierarchy. A number of racks are connected to a *cell* and a typical WSC consists of tens of cells. The latency increases when messages cross multiple layers of the hierarchy and the bandwidth decreases.

Section 2 outlines some existing and future challenges for cloud resource management. The simulation experiments detailed in Sect. 3, show that in the face of rising complexity, such as that postulated here, a market-based bidding mechanism for resource allocation significantly outperforms a centralized resource manager. Keeping all decision information local necessitates the inclusion of functional components to support self-management. Section 5 examines some of these components, coalition formation, combinatorial auctions and a reservation system. Section 4 considers cloud self-management and cloud coalitions and in Sect. 6 we analyze the challenges posed by trying to realize a self-organizing cloud infrastructure.

2 Cloud resource management policies and mechanisms

The polices for cloud resource management are expected to support: (i) admission control; (ii) capacity allocation; (iii) load balancing; (iv) energy optimization, and (v) quality of service (QoS) [21]. Several mechanisms can be used to implement these policies including: control theory, machine learning, utility-based, and market-oriented mechanisms.

Several factors add to the challenges posed by cloud resource management discussed in Sect. 1:

2.1 An increasingly more heterogeneous cloud infrastructure

Servers with different configurations of multi-core processors, attached co-processors (GPUs, FPGAs and MICs) are already, or are expected to become elements of the cloud computing landscape. Amazon Web Services (AWS) already support G2-type instances with GPU co-processors.

2.2 Over-provisioning

Cloud elasticity is based on over-provisioning, assembling pools of resources far larger than required to satisfy the average needs. Elasticity is beneficial, it allows cloud users to increase or decrease their resource consumption based on their need, but it comes at a high cost. Over-provisioning demands high initial investments and results in low server utilization. The average cloud server utilization is in the 18–30% range [1]. Over-provisioning is not economically sustainable [2], it is critical to increase server utilization and the efficiency of the cloud data centers.

2.3 High energy consumption and a large carbon footprint of cloud data centers

Low server utilization implies that the cloud power consumption is far larger than it should be. The power consumption of cloud servers is not proportional with the load, even when idle they use a significant fraction of the power consumed at the maximum load. Computers are not energy proportional systems [1] thus, power consumption of clouds based on over-provisioning is excessive and has a negative ecological impact [3]. A 2010 survey [4] reports that idle or under-utilized servers contribute 11 million tones of unnecessary CO_2 emissions each year and that the total yearly cost of the idle servers is $19 billion.

2.4 The need for cloud interoperability

The cloud computing landscape is fragmented, providers support different cloud delivery models: Amazon IaaS, microsoft platform as a service (PaaS), Google mostly software as a service (SaaS), and so on. Choice is always good but vendor lock-in is an inherent danger. To create an organization which could seamlessly support cloud interoperability and allow multiple cloud delivery models to coexist poses additional challenges.

2.5 New and more demanding cloud applications

The spectrum of cloud services and cloud applications is widening. For example, recently the AWS added some ten new services, including Lambda, Glacier, Redshift, Elastic Cache, and DynamoDB. Several types of EC2 (elastic cloud computing) profiles, M3—balanced, C3—compute optimized, R3—memory optimized, I2 and HS1—storage optimized were also identified in the last months. The spectrum of EC2 instance types is also broadening; each instance type provides different sets of computer resources measured by vCPUs (a hyper-thread of an Intel Xeon core for M3, C3, R3, HS1, G2, and I2).
These challenges, and others, highlight the need to rethink resource management in the light of the expected complexity that will come about as the cloud evolves to services interface as postulated in this paper.

3 Hierarchical control versus market mechanisms

Market mechanisms have distinct advantages over the other mechanisms discussed in Sect. 2. They require neither a system model nor information about the global system state. In a large system, operating in a highly dynamic environment, this is a significant advantage as the information about the global state can be, at best, obsolete.

Can the advantages of market mechanisms be quantified? Answers to this question are reported in [5,6] and [7]. As the cloud computing infrastructure is hierarchically organized it makes sense to compare market mechanisms with hierarchical control. An experiment to estimate the advantage of market mechanisms compared with hierarchical control are discussed in this section.

A realistic cloud infrastructure consisting of several WSCs was simulated and this, in itself, was challenging, in line with Barroso's prediction in [1] "...they (the WSCs) are a new class of large-scale machines driven by a new and rapidly evolving set of workloads. Their size alone makes them difficult to experiment with, or to simulate efficiently."

The scale of the system does not allow a detailed simulation and the results reported are more qualitative than quantitative. For example, the communication complexity is reported as the number of messages at different levels of the network hierarchy, rather than the communication time. Determining the communication time would require more details than we could simulate.

A simulation experiment the simulation experiments are conducted on the Amazon cloud using *c3.8xlarge*[1] EC2 instances. It is challenging to simulate systems with 4–8 WSCs efficiently, the execution time for each one of the simulation experiments reported in this section is about 24 h and each simulation requires 5–6 days wall clock time.

It is important to understand how the scale and the load of the system, as well as, several other parameters of the resource management system affect the ability of the cloud infrastructure to respond to service requests. An important measure of the effectiveness of a resource management system is the communication complexity expressed by the number of messages at each level of an interconnection infrastructure.

The communication latency increases and the bandwidth of the interconnection infrastructure decreases from the rack to the cell and then to the WSC level. We expect the communication complexity of a hierarchical resource management system to be dominated by monitoring and the effort for locating a server capable to process a service request.

The simulation model assumes a time-slotted system. A service request is characterized by three parameters:

(1) Service type.
(2) Service duration—expressed as a number of time slots.
(3) Service intensity—expressed as the number of vCPUs.

The system size, the system load, the service time, the total number of service types supported by the system, and the number of service types supported by a server affect the

[1] Compute-optimized instance with 32 vCPU and 60 GiB memory.

system performance. From the broad set of system performance metrics the following are the most relevant:

- The number of messages exchanged at different levels for mapping the service requests. These numbers reflect the overhead of the request processing process.
- The ability of the system to balance the load measured as the coefficient of variation (the variance versus the average) of the system load per allocation slot (AS).
- The rejection ratio (RR), the ratio of service requests rejected because no server able to match the service type, the service intensity, and the service duration demanded by the clients could be found.

The system configuration is derived from the data in [1] and the parameters of the simulation model have been chosen as realistic as possible. The experiments were conducted with two system configurations, 4 and 8 WSCs. A WSC has the following configuration: 24 cells, 100 racks per cell, 40 servers in each rack, and 4 processors per server. Thus, a WSC has 88, 000 servers and 352, 000 processors. The system is homogeneous, all servers have the same capacity 100 vCPUs.

The simulation environment is flexible. A configuration file describes the system infrastructure, the network speed, the server load, and the parameters of the model. For example, the system configuration for the high initial load case is:

```
% System configuration
 serverNum = 40;
 cpuNum = 4;
 rackNum = 100;
 cellNum = 25;
 WSCsNum = 4;
 servers_capacity =100;
% Network speeds and load parameters
 interRackSpeed = 1;
 intraRackSpeed = 10;
 MIN_LOAD = 80;
 MAX_LOAD = 85;
% Model parameters
 NUMBER_OF_TYPES = 100;
 vCPU_MAX_REQUES = 800;
 vCPU_MIN_REQUEST =10;
 vCPU_PER_SERVER = 10;
 MAX_SERVICE_TIME = 10;
 MONITORING_PERIOD = 10;
 SIMULATION_DURATION = 200;
 SERVER_TYPES = 5;
 REQUEST_TYPES = 5;
 RACK_CAP =  serverNum *
     servers_capacity;
 CLUS_CAP= rackNum * RACK_CAP;
 WSC_CAP= clusterNum * CLUS_CAP;
 SYSTEM_CAP= WSCsNum * WSC_CAP;
```

The amount of resources in a service request has a broad range, between 10 and 800 vCPUs, while a single server can provide 10 vCPUs. The spectrum of service types offered is quite large, initially 500 types and then reduced to 100.

The time is slotted and a batch of service requests with a random distribution of the service time, type, and intensity arrive in each slot. The individual service requests are randomly assigned to one of the WSCs. Practical considerations regarding simulation costs and time to get the results have limited the duration of simulation to 200 ASs.

Several simulation experiments with different system parameters are presented. In the first experiment the attributes of service requests are uniformly distributed and the ranges are: $(1 - 100)$, $(1 - 10)$, and $(10 - 800)$ for service type, service time, and service intensity, respectively. A server supports 5 different service types randomly selected from a total of 500 possible service types. The monitoring interval is ten ASs; for later experiments it will increase to 20 and then to 50 ASs. In the next experiments the effect of changing the parameters of the system model are investigated:

(1) Doubling the number of WSCs from 4 to 8; this gives an indication of the scalability of the model.
(2) Increasing the average system load from about 20% to about 80% gives an indication about the robustness of the policy and its ability to perform well under stress.
(3) Reducing the number of requested service types from 500 to 100; one of the aims is to support a very broad range of services so the impact of the service diversity is important.
(4) Reducing the number of types of services offered by each server from 5 to 2; the more types of services the more flexible the server configuration should be.
(5) Changing the distribution of the service time from $(1 - 10)$ to $(1 - 20)$ time slots. The larger the range of the service time the broader the range of applications able to use the cloud infrastructure.
(6) Increasing the monitoring interval for hierarchical control from 20 to 50 time slots; the monitoring interval is expected to have an effect on the quality of information used by load balancers.

Hierarchical control in each time slot incoming service requests are randomly assigned to one of the WSCs. Each WSC periodically collects data from the cells, which in turn collect data from racks, which collect data from individual servers.

The communication complexity for this monitoring process increases linearly with the size of the system. The more frequent the monitoring at each level, the more accu-

Table 1 Hierarchical control—the simulation results for a system configuration with 4 WSCs

WSCs	Initial/final load (%)	Initial/final γ	RR (%)	# service requests	WSC msg/req	Cell msg/req	Rack msg/req
4	22.50/19.78	0.007/0.057	2.2	14,335,992	0.98	3.18	271.92
	78.50/82.38	0.004/0.183	7.1	57,231,592	1.01	10.16	973.15
8	22.50/19.26	0.006/0.049	1.9	31,505,482	0.98	3.18	271.92
	78.50/81.98	0.005/0.213	8.7	94,921,663	1.01	11.36	1071.75

Shown are: the average initial and final system load for the low and high load; the initial and final coefficient of variation γ of the load; the RR; and the average number of messages for monitoring and control per service request at WSC, cell, and rack level

Table 2 Hierarchical control

WSCs	Initial/final load (%)	Initial/final γ	RR (%)	# of service requests	WSC msg/req	Cell msg/req	Rack msg/req
4	22.50/21.15	0.003/0.051	1.9	16,932,473	1.00	3.53	337.34
	82.50/67.18	0.003/0.109	7.2	42,034,225	1.00	11.15	1,097.00
8	22.50/22.13	0.008/0.055	5.4	38,949,889	1.00	4.22	470.35
	82.50/81.63	0.006/0.155	4.2	84,914,877	1.00	10.72	1,038.96
4	22.50/21.15	0.003/0.051	1.7	17,341,885	0.99	3.22	276.34
	82.50/74.27	0.006/0.059	14.6	52,206,014	1.00	12.12	1255.40
8	22.50/16.27	0.006/0.035	1.3	37,750,971	0.99	3.18	268.27
	82.50/74.55	0.007/0.081	2.9	99,686,943	1.00	10.77	1,036.64

Top half the number of service types is reduced from 500 to 100, *bottom half* the number of service types offered by a server is reduced from 5 to 2. All other parameters are identical to the ones for the experiment with results in Table 1

rate the information is, but the larger the volume of data and the interference with the "productive communication", communication initiated by running applications. The communication bandwidth at each level is limited and when the system load increases the communication latency is likely to increase significantly, as many applications typically exchange large volumes of data.

The simulation model assumes that load balancers at each level monitor the system they control. When a request is assigned to a WSC, the load balancer directs it to the cell with the lowest reported load and the process repeats itself at the cell level. The cell load balancer directs the request to the rack with the lowest reported load, which in turn directs it to server with the lowest reported load.

If the server rejects the request, the rack load balancer redirects the request to the server with the next lower load. If the rack cannot satisfy the request, it informs the cell load balancer which in turn redirects the request to the rack with the next lowest reported average load. The request is rejected if none of the cells of the WSC are able to find a server able to satisfy the type, duration, and intensity of the service request.

The simulation is conducted for two average initial system loads: low, around 20% and high, around 80% of the system's capacity. The total number of service requests for 4 WSCs and for low and high initial system load are around $(12 - 17) \times 10^6$ and $(42 - 57) \times 10^6$, respectively. In each case we show: (1) the number of WSCs; (2) the average initial and final system load for low and high load; (3) the initial and final coefficient of variation γ of the load; (4) the RR; the number of messages for monitoring and control per service request at (5) WSC level; (6) cell level; and (7) rack level.

Simulation results for hierarchical control the results of the first simulation experiment, Table 1, show that the RR, the coefficient of the variation of the final load, and the average number of messages required to map a service request to a server are more than three fold larger in the case of higher load; indeed, $7.1/2.2 = 3.22, 0.183/0.057 = 3.22$, and $984/276 = 3.2$, respectively. At higher load more requests are rejected, load balancing is less effective, and the overhead for mapping a request is considerably higher. The increase in the number of messages means a substantial increase in the communication costs and also a longer waiting time before a request enters the service.

Doubling the size of the system does not affect the statistics for the same average system load. For example, when the initial average load is 22.50% the average number of messages exchanged per service request is the same at the three levels of the hierarchy for both system configurations. The RR varies little, 2.2 versus 1.9 and 7.1 versus 8.7% for 4 and 8 WSCs, respectively.

Table 2 (Top half) presents the results after reducing the total number of service request types from 500 to 100. A reduction of the RR and of the number of messages at high load for the larger configuration of 8 WSCs compared to the

Table 3 Hierarchical control

WSCs	Initial/final load (%)	Initial/final γ	RR (%)	# of service requests	WSC msg/req	Cell msg/req	Rack msg/req
4	22.50/22.41	0.005/0.047	0.20	12,352,852	1.00	3.13	261.11
	82.50/80.28	0.003/0.063	2.10	43,332,119	1.00	3.41	1108.12
8	22.50/22.77	0.005/0.083	1.30	25,723,112	1.00	3.11	236.30
	82.50/79.90	0.005/0.134	4.10	88,224,546	1.00	10.63	1029.56
4	22.50/21.07	0.003/0.033	1.00	12,335,103	0.99	3.21	270.07
	82.50/83.46	0.007/0.080	1.80	51,324,147	1.01	10.87	1040.63
8	22.50/19.16	0.005/0.030	1.30	29,246,155	1.00	3.37	304.88
	82.50/84.12	0.002/0.041	2.30	93,316,503	1.00	3.66	1005.87

Top half the service time is uniformly distributed in $(1-20)$ instead of $(1-10)$ allocation slots, *bottom half* the monitoring interval is increased from 10 to 50 reservation slots. All other parameters identical to the ones for the experiment with results in Table 1

case in Table 1 is noticeable. Also, the RR decreases from 7.4 to 4.2% for configurations with 4 and 8 WSCs, respectively.

When the number of service types offered by a server is reduced from 5 to 2 and the system configuration changes from 4 to 8 WSCs the rejection rate decreases, see Table 2 (bottom half). The reduction from 14.6 to 2.9 can be attributed to the fact that an incoming service request is randomly assigned to one of the WSCs; the larger the number of WSCs the less likely is for the request to be rejected. The number of messages at the rack level is considerably larger for the smaller system configuration at high load, 1255 versus 973 in the first case presented in Table 1.

Next, the monitoring interval is set to 20 ASs and the service time is uniformly distributed in the range 1–20 ASs. The results in Table 3 (top half) show that the only noticeable effect is the reduction of the rejection rate.

In the following experiment the monitoring interval is extended from 10 to 50 ASs. The service time is uniformly distributed in the range 1–10 ASs; even when the monitoring interval was 10 ASs, this interval is longer than the average service time thus, the information available to load balancers at different levels is obsolete.

The results in Table 3 (bottom half) show that increasing the monitoring interval to 50 slots has little effect for the 4 WSC configuration at low load, but it reduces substantially the RR and increases the number of messages at high load. For the 8 WSC configuration increasing the monitoring interval reduces the RR at both low and high load, while the number of messages changes only slightly.

Figure 1 (top) show the time series of the average system load for the low and the high initial load, respectively for the case in Table 1 when the monitoring interval is 20 time slots and the service time is uniformly distributed in the 1–20 slots range and there are 8 WSCs. The system workload has significant variations from slot to slot; for example, at high load the range of the average system load is from 58 to 85% of the system capacity. Figure 1 (bottom) show the initial and the final load distribution for the 8 WSCs; the imbalance among WSCs at the end of the simulation is in the range of 1–2%.

The results of the five simulation experiments are consistent, they typically show that at high load the number of messages, thus the overhead for request mapping increases three to four fold, at both cell and rack level and for both system configurations, 4 and 8 WSCs.

Simulation of a market model in this resource management model all servers of a WSC bid for service. A bid consists of the service type(s) offered and the available capacity of the bidder.

The overhead is considerably lower than that of the hierarchical control; there is no monitoring and the information maintained by each WSC consists only of the set of unsatisfied bids at any given time. The servers are autonomous and act individually; there is no collaboration among them, while self-organization and self-management require agents to collaborate with each other.

At the beginning of an AS servers with available capacity above a threshold τ place bids. The bids are then collected by each WSC. A bid is *persistent,* if not successful in the current AS it remains in effect until a match with a service request is found. This strategy to reduce the communication costs is justified because successful bidding is the only way a server can increase its workload.

One of the objectives of the investigation is the effectiveness of the bidding mechanism for lightly and heavily loaded system, around 20 and 80% average system load, respectively. The thresholds for the two cases are different, $\tau = 30\%$ for the former and $\tau = 15\%$ for the latter. The choice for the lightly loaded case is motivated by the desire to minimize the number of messages; a large value of τ, e.g., 40% would lower the RR but increase the number of messages. Increasing the threshold, e.g., using a value $\tau = 20\%$, would increase dramatically the rejection rate in case of heavily loaded system; indeed, few servers would

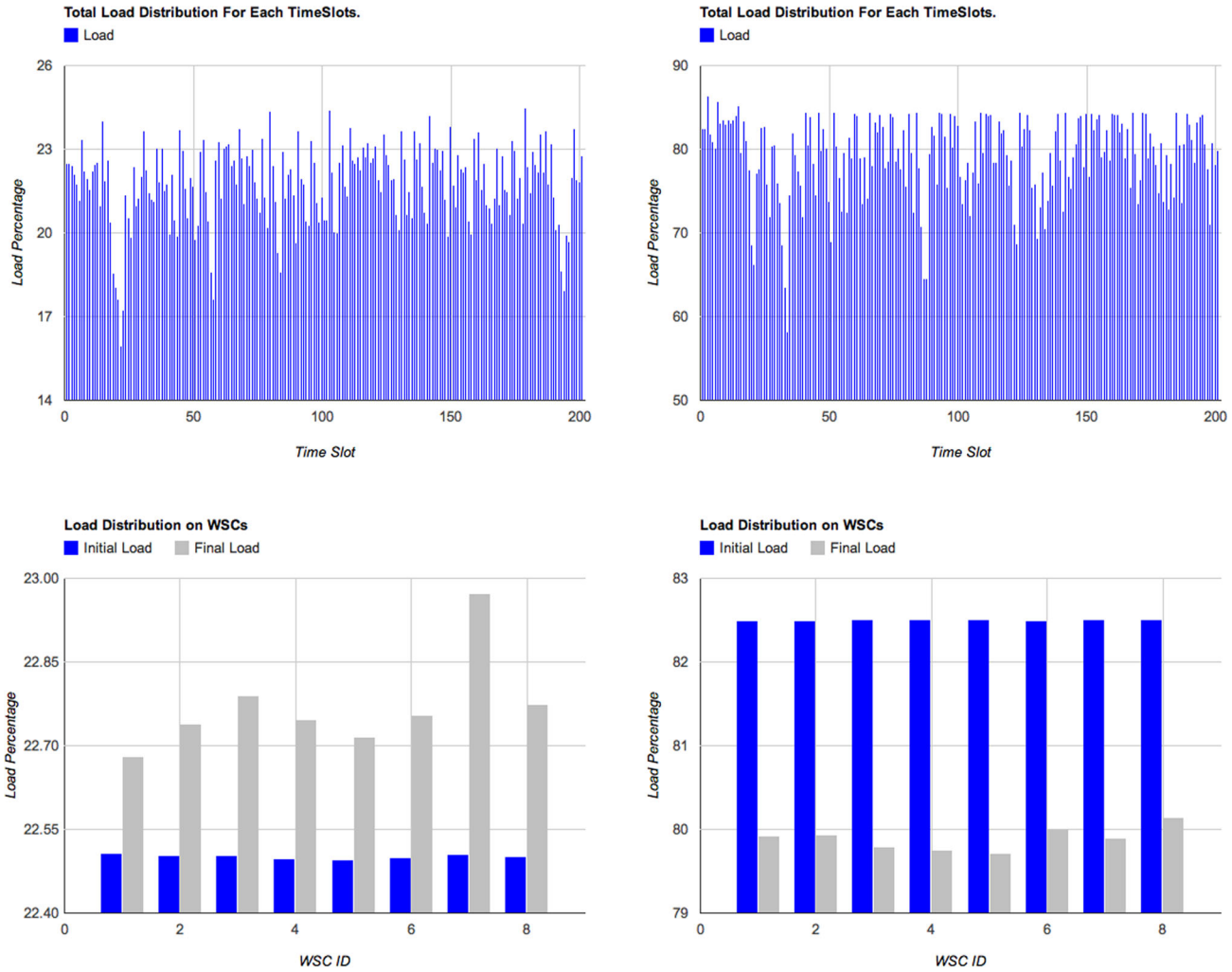


Fig. 1 Hierarchical control for a cloud with 8 WSCs. The monitoring interval is 20 ASs and the service time is uniformly distributed in the range 1–20 ASs. The initial average system load: (*left*) 20%; (*right*) 80% of system capacity. *Top* time series of the average WSC load. *Bottom* initial and final average WSC load

have 20% available capacity when the average system load is 80%.

Simulation results for the market model the measurements reported for the hierarchic control are repeated under the same conditions as those for hierarchical control for a fair comparison; only bidding replaces monitoring and hierarchical control. The same performance indicators are used: communication complexity, the efficiency of load balancing, and the RR. The results are shown in Tables 4 and 5.

The simulation results show a significant reduction of the communication complexity, more than two orders of magnitude in case of the market-oriented mechanism. For example, at low average load the average number of messages per reservation request at the rack level is 0.987, Table 4, versus 271.92 for 4 and 8 WSCs, Table 1.

At high average load the same values are: 4.155 versus 973.14 for the 4 WSC case and 5.761 versus 1071.75 for the 8 WSC case. A second observation is that when the average load is 20% of the system capacity the communication complexity is constant, 0.987, for both configurations, 4 and 8 WSCs, regardless of the choices of simulation parameters. At high average load, the same value is confined to a small range, 4.078 to 6.734.

The organization is scalable, the results for 4 and for 8 WSCs differ only slightly. This is expected because of the distributed scheme where each WSC acts independently, it receives an equal fraction of the incoming service requests and matches them to the bids placed by the servers it controls.

The average RR at low average load decreases, see Tables 4 and 5. On the other hand, the rejection rate increases when the range of the service time increases from the $1-10$ to $1-20$, see Table 5 (bottom). This effect is most likely due to the fact that requests with a large service time arriving

Table 4 Market model

WSCs	Initial/final load (%)	Initial/final γ	RR (%)	# service requests	WSC msg/req	Cell msg/req	Rack msg/req
4	22.50/23.76	0.007/0.067	.22	15,235,231	0.002	0.011	0.987
	82.50/80.32	0.004/0.115	5.44	63,774,913	0.003	0.042	4.155
8	22.50/22.47	0.006/0.033	.18	30,840,890	0.002	0.011	0.987
	82.50/81.30	0.005/0.154	7.23	89,314,886	0.003	0.054	5.761

Simulation results for a system configuration with 4 WSCs. Shown are the initial and final system load for the low and high load, the initial and final coefficient of variation γ of the load, the RR, and the average number of messages for monitoring and control per service request at WSC, cell, and rack level

Table 5 Market model

WSCs	Initial/final load (%)	Initial/final γ	RR (%)	# of service requests	WSC msg/req	Cell msg/req	Rack msg/req
4	22.50/22.3	0.004/0.050	.18	15,442,372	0.002	0.011	0.987
	82.50/79.88	0.004/0.098	6.01	56,704,224	0.002	0.059	5.968
8	22.50/23.0	0.007/0.049	.3	31,091,427	0.002	0.011	0.987
	82.50/80.91	0.009/0.127	5.81	85,322,714	0.003	0.0.51	5.845
4	22.50/20.94	0.007/0.056	.1	15,295,245	0.002	0.011	0.987
	82.50/77.83	0.008/0.133	10.1	49,711,936	0.003	0.063	6.734
8	22.50/22.33	0.007/0.063	.02	31,089,191	0.002	0.011	0.987
	82.50/78.18	0.008/0.142	3.61	71,873,449	0.002	0.059	6.098
4	22.50/23.31	0.002/0.064	2.27	13,445,186	0.001	0.011	0.988
	82.50/84.05	0.007/0.101	3.75	57,047,343	0.002	0.042	6.329
8	22.50/18.93	0.007/0.038	2.94	28,677,012	0.001	0.011	0.988
	82.50/85.13	0.008/0.072	4.38	88,342,122	0.002	0.029	4.078

Top the number of service types is reduced from 500 to 100, *center* the number of service types offered by a server is reduced from 5 to 2, *bottom* the service time is uniformly distributed in the range (1–20) instead of (1–10) ASs. All other parameters are identical to the ones for the experiment with results in Table 4

during later slots do not have time to complete during the 200 allocation slots covered by the simulation.

At high average system load the average RR is only slightly better for market-based versus hierarchical control. Lastly, the market-based mechanism performs slightly better than hierarchical control in terms of slot-by-slot load balancing, the coefficient of variation of the system load per slot is $\gamma \leq 1.115$.

The number of different service types offered by the cloud does and the number of services supported by individual servers, do not seem to affect the performance of the system see Table 5 (top) and (center).

Figure 2 (top) show time series of the average system load for the low and the high initial load, respectively. The actual system workload has relatively small variations from slot to slot; for example, at high load the range of the average system load ranges from 77 to 82% of the system capacity. Figure 2 (bottom) show the initial and the final load distribution; the imbalance among the eight WSCs at the end of the simulation is in the $21 - 23\%$ range at low load and in the $80 - 80.1\%$ range at high load.

The results show that market-based policy performs well at high system load and this is extremely important. The average server utilization based on existing cloud resource management policies reported in the literature is rather low. A policy that allows servers to operate effectively under heavy load is highly desirable.

The results of the simulation experiments discussed in this section confirm our intuition that monitoring required by a hierarchical resource management adds a significant overhead for resource management in a large-scale system and cannot provide accurate information about the state of system resources. We can only draw qualitative conclusions from the simulation experiments, the performance of the market mechanisms is significantly better for critical performance metrics than the results of hierarchical control and this effect is noticeable for experiments with different sets of parameter models.

4 Cloud coalitions

Coalition formation supports a more effective use of large-scale system resources, as well as a convenient means for accessing these resources [8]. There is little surprise that the

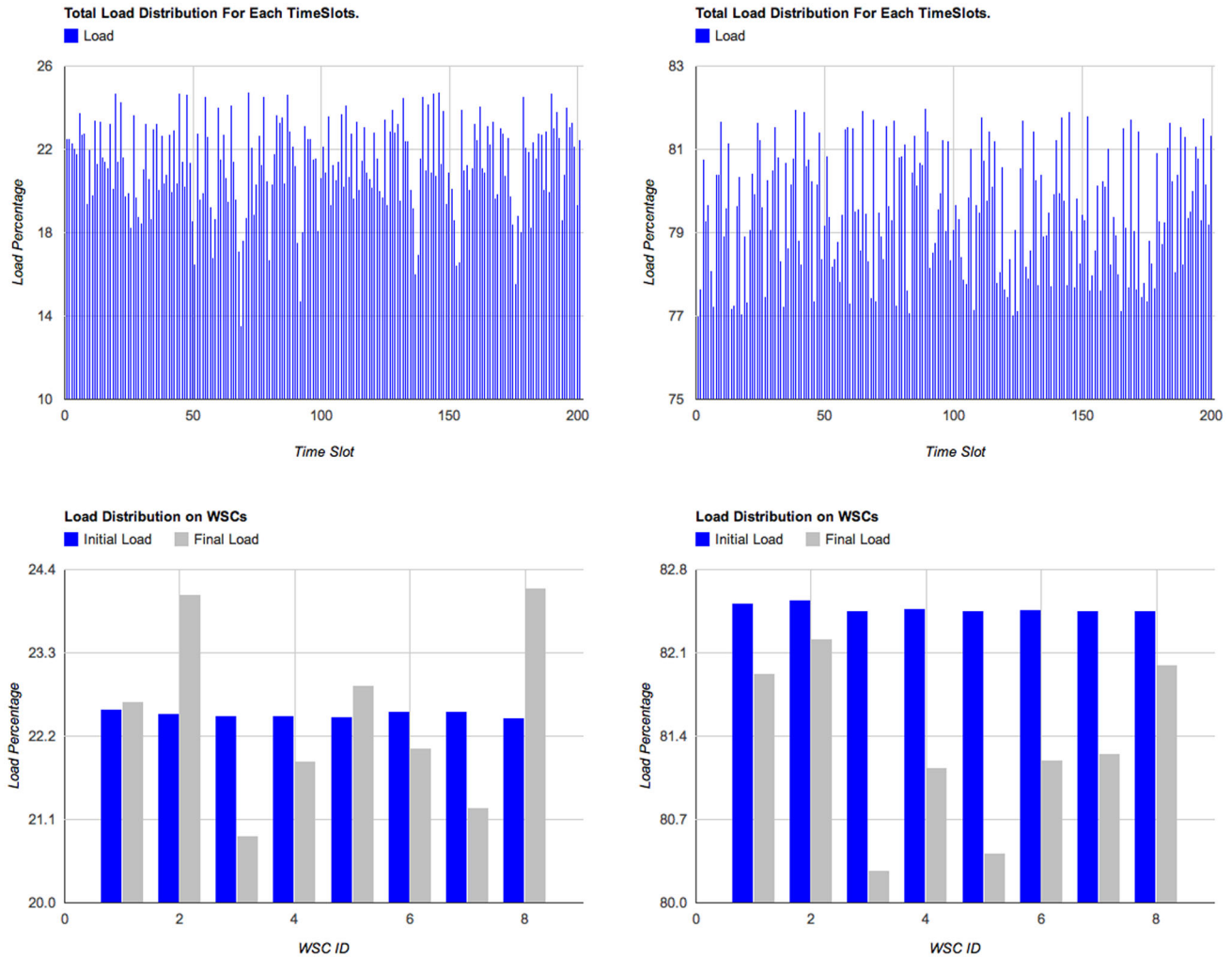


Fig. 2 Market model. A cloud with 8 WSCs, the monitoring interval is 20 ASs and the service time is uniformly distributed in the range 1–20 ASs. The initial average system load is: (*left*) 20%; (*right*) 80% of system capacity. *Top* time series of the average load. *Bottom* initial and final average load

interest in coalition formation migrated in recent years from computational grids to cloud resource management.

A stochastic linear programing game model for coalition formation is presented in [9]; the authors analyze the stability of the coalition formation among cloud service providers (CSP) and show that resource and revenue sharing are deeply intertwined. An optimal VM provisioning algorithm ensuring profit maximization for CSPs is introduced in [10].

A combinatorial coalition formation problem is described in [11]. That paper assumes that a seller has a price schedule for each item. The larger the quantity requested, the lower is the price a buyer has to pay for each item; thus, buyers can take advantage of price discounts by forming coalitions. A similar assumption is adopted by the authors of [16] who investigate systems where the negotiations among deliberate agents are not feasible due to the scale of the system.

Two types of cloud coalitions are reported in the literature:

(1) Coalitions of CSPs for the formation of cloud federations. A *cloud federation* is an infrastructure allowing a group of CSPs to share resources; the goal is to balance the load and improve system reliability.
(2) Coalitions of the servers of a data center. The goal is to assemble a pool of resources larger than the ones available in a single server.

The vast majority of on-going research in this area is focused on game-theoretic aspects of coalition formation for cloud federations [9,10,12]. Cloud federations require a set of standards and that aspect of the present cloud computing landscape is still evolving. The adoption of inter-operability standards supporting cloud federations seems a distant possibility, in spite of the efforts of the cloud computing community coordinated by the National Institute of Standards (NIST).

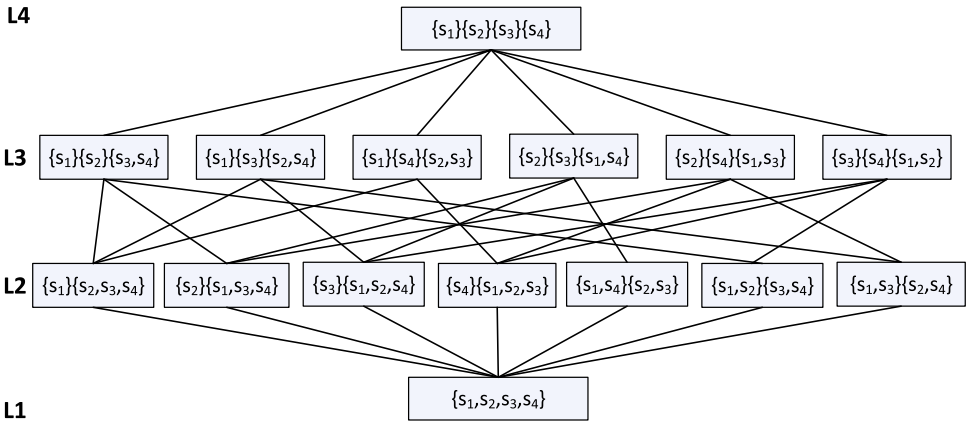


Fig. 3 A lattice with four levels $L1$, $L2$, $L3$ and $L4$ showing the coalition structures for a set of 4 servers, s_1, s_2, s_3 and s_4. The number of coalitions in a coalition structure at level L_k is equal to k. In a homogeneous system the identity of the servers does not matter and there is only one coalition structure at each level

The second coalition type has received little attention in the past. This is likely to change due to the emerging interest in *Big Data* cloud applications, which require more resources than a single server can provide. To address this deficit, this paper looks considers sets of identically configured servers, able to communicate effectively among themselves, forming coalitions with sufficient resources for data- and computationally-intensive problems. For the rest of this paper, coalition formation refers to this type only.

Coalition formation to support Big Data applications is considered in [5]. Coalition formation is modeled as a cooperative game where the goal of all agents is to maximize the rewards for the entire set of agents. A set of R servers $\{s_1, s_2, \ldots, s_R\}$, located in the same rack is considered. In this case, a *coalition* $\mathbb{C}_i$ is a non-empty subset of R.

Figure 3 shows a lattice representation of the coalition structures for a set of four servers s_1, s_2, s_3 and s_4. This lattice has four levels, $L1$, $L2$, $L3$ and $L4$ containing the coalition structures with 1, 2, 3 and 4 coalitions, respectively. In general, the level k of a lattice contains all coalition structures with k coalitions; the number of coalitions structures at level k for a population of N agents is given by the sterling number of second kind:

$$\mathcal{S}(N,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^N. \quad (1)$$

In the case illustrated in Fig. 3, $N = 4$ and the number of coalition structures at levels $L1 - L4$ are 1, 7, 6, 1, respectively.[2] The total number of coalition structures with N agents is called the Bell number

$$\mathcal{B}(N) = \sum_{k=0}^{N}\mathcal{S}(N,k) = \sum_{k=0}^{N}\frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^N. \quad (2)$$

The number of coalition structures increases exponentially with the number of agents. For example, for $N = 40$, a typical number of servers in a rack, the number of coalition structures is close to 10^{35}[3] and number of coalitions is close to 10^{10}.

Searching for the optimal coalition structure $\mathbb{C}$ is computationally challenging due to the size of the search space. The first step for determining the optimal coalition structure is to assign a value v reflecting the utility of each coalition. The second step is the actual coalition formation.

By considering a rack to be homogeneous, all servers have an identical configuration. This realistic assumption considerably simplifies the complexity of the search for an optimal coalition structure, as the servers are indistinguishable from one another.

An algorithm to find optimal coalition structures in cooperative games by searching through a lattice like the one in Fig. 3 was introduced by [13]. A more refined algorithm is described in [14]; in this algorithm the coalition structures are grouped according to the so-called *configurations* reflecting the size of the coalitions.

5 Rack-level coalition formation and combinatorial auctions

Since the infrastructure of a WSC is hierarchically structured, communication latency is lower among servers in the same rack. Coalition formation within a rack is therefore optimal from that perspective. Moreover, servers within a rack can be arranged to be homogeneous, while allowing from heterogeneity among racks. Thus, forming rack-level coalitions can also be guaranteed to be homogeneous.

Task-oriented coalition formation is often $\mathcal{NP}$ hard [15]. When all agents have the same ability to perform a single task, the problem is similar to the set partitioning problem,

[2] For $N = 5$ and $N = 6$ the stirling numbers of the second kind are respectively 1, 15, 25, 10, 1 and 1, 31, 90, 65, 15, 1.

[3] $\mathcal{S}(40, 14) = 3.58598722556218034914285 54E + 34$

while in the case of agents able to perform multiple tasks the problem resembles the set covering problem [16].

Here reservation system is designed to find resource coalitions to undertake a specific service request. The system, proposed here, has two stages; coalitions of servers are formed periodically during the first stage and, in the second stage, these coalitions participate in combinatorial auctions designed to identify a collection of coalitions capable of undertaking a work flow of services.

System organization a set of N servers $\{s_1, s_2, \ldots, s_N\}$, located in the same rack are considered. A *coalition* $\mathbb{C}_i$ is a non-empty subset of N. A *coalition structure* is a set of m coalitions $\mathbb{S} = \{\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_m\}$ satisfying the following conditions

$$\bigcup_{i=1}^{m} \mid \mathbb{C}_i \mid = N \text{ and } i \neq j \Rightarrow \mathbb{C}_i \bigcap \mathbb{C}_j = \emptyset. \quad (3)$$

At least two basic mechanisms for coalition formation are possible. The first one, will be referred to as *just-in time coalition formation* . It consists of several steps: first, service requests are examined by the WSC to determine the type of servers and the size of the coalitions needed, secondly, servers and the coalition sizes matching these requirements are identified. Finally, any unsatisfied requests have to be processed.

The second mechanism, *coalition formation based on past history,* integrates the two processes, coalition formation and combinatorial auctions, as stage one and stage two, respectively of the reservation system. The system now uses information from past auctions to determine the size of the server coalitions formed by racks with different types of servers, and then matches them to the current needs expressed by service requests. In the second stage combinatorial auctions the coalitions created during the first stage are included in successfully auctioned packages thus, the precise value of all coalitions structures can be determined. An important condition is that only *available servers*, servers with no commitments for the current slot, can participate in coalition formation and in the auction organized in that slot; call $N_a \leq N$ that number of available servers.

Coalition formation protocol an elected *rack leader* collects information about all *successful coalitions* - coalitions that have been included in packages auctioned successfully during a window of, w, successive past slots. The current rack-leader records an entry for the corresponding *partial coalition structure (PCS)* including n_k—the *coalition size*, m_k - the *multiplicity* of occurrence, the value $\bar{v}_k$ calculated as the average price over all auctions when a PCS, including a coalition of size n_k, was part of a *package* successfully auctioned during the past w ASs.

Call $\mathcal{L}$ the *PCL-list*. For a window of size w the list $\mathcal{L}$ is the list of all triplets $\mathcal{L}_k = [n_k, m_k, \bar{v}_k]$ ordered first by $1 \leq n_k \leq N_a$ then by m_k. The list includes only entries $\mathcal{L}_k$ with $\bar{v}_k > 0$. Given N_a a *coalition structure (CS)* $\mathbb{S}_k$ among the entries $\mathcal{L}_{k1}, \mathcal{L}_{k2}, \ldots\ldots, \mathcal{L}_{kn}$ is *feasible* if $\sum_j n_k \times m_k = N_a$. Then, the value of the coalition structure $\mathbb{S}_k$ is $v_k = \sum_j \bar{v}_j$. Note that the formation of coalitions can be forced to include all available servers. An example of a PCS list $\mathcal{L}$ follows

```
---------------------------------------
a [1,4,35]  \*4 PCS of 1-server {s}
b [1,15,682] \ *15 PCS of 1-server {s}
        .........
c [2,3,78] \*3 PCS of  2-servers {s,s}
        ........
d [3,2,502] \*2 PCS of 3-servers {s,s,s}
e [3,4,812] \*4 PCS of 3-servers {s,s,s}
        .........
f [16,1,751] \*1 PCS of 16-servers
  {s,..,s}
g [16,2,740] \*2 PCS of 16-servers
  {s,..,s}
        .........
---------------------------------------
```

In this example, some of the feasible coalitions structures when $N_a = 16$ are: $\mathbb{S}_g$ with $v_g = 751$; $\mathbb{S}_{a,b}$ with $v_{a,b} = 35 + 682 = 712$; $\mathbb{S}_{a,e}$ with $v_{a,e} = 35 + 812 = 837$; $\mathbb{S}_{a,c,d}$ with $v_{a,c,d} = 35 + 78 + 502 = 615$, and so on. Note that the value of a coalition reflects also the length of time the coalition was active in response to successful auction. It can be seen that a PCS of 15 coalitions of 1 server have been active for larger number of slots than a PCS of 4 coalitions of 1 server. The value attributed to a coalition of k servers is distributed equally among the servers; the value of a package of several coalitions auctioned successfully is divided among the coalitions based on the resource supplied by each one of them.

The coalition formation protocol proceeds as follows:

(1) Server s_i sends to the current rack leader:
   (a) A vector $([v_i^1, \beta_i^1], [v_i^2, \beta_i^2], \ldots....[v_i^N, \beta_i^N])$ with $v_i^k, 1 \leq k \leq N$ the total value due to the participation of s_i in successful coalitions, of k servers and β_i^k a bit vector with w components with $\beta_i^{k,j} = 1$ if s_i was included in a successful coalition of k servers in slot j of window w.
   (b) Availability, $a_i = 1$ if available, 0 otherwise.

(2) After receiving the information from all servers the current rack leader:
   (a) Determines $N_a = \sum_{i=1}^{N} a_i$.
   (b) Computes $m_k = \sum_{i=1}^{N_a} \sum_{j=1}^{w} \beta_i^{k,j}, 1 \leq k \leq N$.
   (c) Computes $\bar{v}_k = \sum v_i^k$
   (d) Computes the optimal coalition structure.

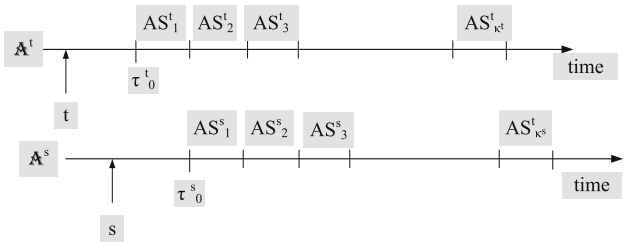


Fig. 4 Auctions $\mathbb{A}^t$ and $\mathbb{A}^s$ conducted at times t and s, respectively. τ_0^t and τ_0^s are the start of the first ASs, AS_1^t and AS_1^s of the two auctions. The number of slots auctioned in each case are κ^t and κ^s, respectively

(e) Assigns a server to coalition of size k a based on the values v_i^k.
(f) Chooses the best performer as the next coalition leader. The best performer is the one with the largest value $\sum_j v_i^j$.

Finding the optimal CS requires at most L operations with L the size of the PCL-list. The system starts with a predetermined coalition structure and coalition values.

Combinatorial auction protocol the protocol introduced in [17] targets primarily the *IaaS* cloud delivery model represented by AWS. Reservation systems are regularly used by CSPs. For example, AWS supports reservations as well as spot instances and offers a limited number of instance families, including M3 (general purpose), C3 (compute optimized), R3 (memory optimized), I2 (storage optimized), G2 (GPU) and so on. An instance is a package of system resources; for example, the `c3.8xlarge` instance provides 32 vCPU, 60 GiB of memory, and 2×320 GB of SSD storage.

An AS is a period of fixed duration, e.g., 1 h, that can be auctioned. An *auction*, $\mathbb{A}^t$, is organized at time t if there are pending reservation requests which require immediate attention. Figure 4 shows two consecutive auctions at times t and s; during the first slot of auction $\mathbb{A}^t$ new reservation requests are received and the AS AS_2^t is not fully covered; this slot becomes AS_1^s for $\mathbb{A}^s$.

A service $\mathcal{A}$ is described by a relatively small number of *attributes*, $\{a_1, a_2, \ldots, \}$. Each attribute a_i can take a number of distinct values, $v_i = \{v_{i,1}, v_{i,2}, \ldots\}$. The first attribute is the coalition size or equivalently the number of vCPS provided; other attributes could be the type of service and server architecture with two values "32-bit" and "64-bit;" another attribute could be "organization" with values "von Neumann" (vN), "data-flow" (DF), or "vN with graphics co-processor" (vN-GPU).

This protocol is inspired by the clock-proxy auction [18]. The clock-proxy auction has a clock phase, where the price discovery takes place, and a proxy phase, when bids for packages are entertained. In the original clock-proxy auction there is one seller and multiple buyers who bid for packages of goods.

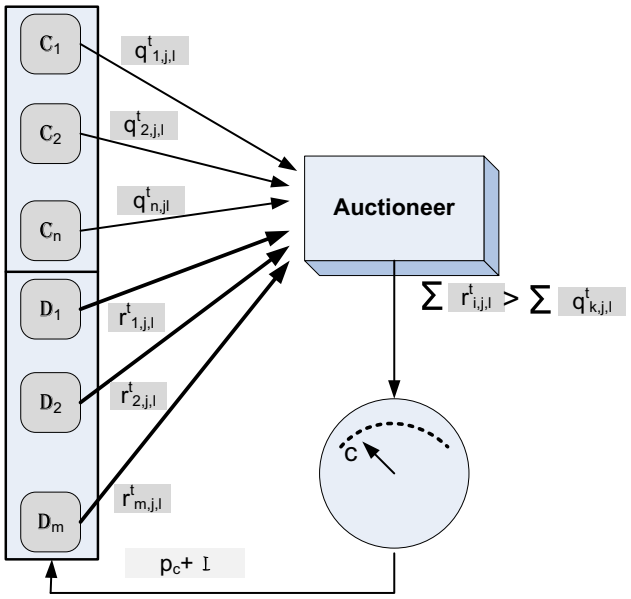


Fig. 5 The clock phase for service S_l^t and slot j. The starting price is p_l^0 given by Eq. 4. The clock advances and the price increases from p_c to $p_c + \mathcal{I}$ when the available capacity at that price given by Eq. 5 is exhausted; the demand is given by Eq. 5

The clock phase Figure 5 illustrates the basic idea of a clock phase: the auctioneer announces prices and the bidders indicate the quantities they wish to buy at the current price. When the demand for an item increases, so does its price until there is no excess demand; on the other hand, when the offering exceeds the demand, the price decreases.

During the clock phase of auction $\mathbb{A}^t$ the price discovery is done for each time slot and for each type of service; a clock runs for each one of the κ^t slots and for each one of the ν^t services. Next the clock phase for service S_l^t in slot j is described. Assume that there are n coalitions $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_n\}$ offering the service and m requests for reservations from clients $\mathbb{D} = \{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_m\}$. A clock auction starts at clock time $t = 0$ and at price per unit of service for S_l

$$p_l^0 = \min_{\mathcal{C}_k}\{p_{k,l}\} \tag{4}$$

Call $\mathcal{C}_0$ the available capacity at this price and $\mathcal{D}_0$ the demand for service S_l^t offered at price p_l^0 in slot j

$$\mathcal{C}_0 = \sum_{k=1}^{n} q_{k,j,l}^t \text{ and } \mathcal{D}_0 = \sum_{i=1}^{m} r_{i,j,l}^t. \tag{5}$$

If $\mathcal{C}_0 < \mathcal{D}_0$ the clock c advances and the next price per unit of service is

$$p_l^1 = p_l^0 + \mathcal{I} \tag{6}$$

with $\mathcal{I}$ the price increment decided at the beginning of auction. There is an ample discussion in the literature regarding the size of the price increment; if too small, the duration of the clock phase increases, if too large, it introduces incentives for gaming [18].

The process is repeated at the next clock value starting with the new price. The clock phase for service S_l^t and slot j terminates when there is no more demand.

The proxy phase in a traditional clock-proxy auction the bidders do not bid directly, they report the price to a proxy and the quantity of each item in the package they desire. The proxy then bids in an ascending package auction.

In the system presented here, the proxy phase of the auction consists of multiple rounds. The auction favors bids for long runs of consecutive slots when the service is provided by the same coalition. This strategy is designed to exploit temporal and spatial locality.

The auction starts with the longest runs and the lowest price per slot and proceeds with increasingly shorter runs and diminished incentives. Once a run of consecutive slots is the subject of a provisional winning bid, all shorter runs of slots for that particular service are removed from the coalition offerings.

During the first round only the longest run of consecutive slots for each one of the services offered by the participating coalitions is auctioned and only bidders that have committed to any of the slots of the run are allowed to bid. The price per slot for the entire run is the lowest price for any slot of the run the bidder has committed to during the clock phase of the auction. If there are multiple bids for service S_l^t the *provisional winner* is the one providing the largest revenue for the coalition offering the service.

If κ_l^t is the longest run of consecutive slots for service S_l^t auctioned in the first round then, in the second round, a shorter run of $\kappa_l^t - 1$ slots is auctioned. The price for the entire run equals the second lowest price for any slot of the run the bidder has committed to during the clock phase of the auction times the number of the time slots in the run.

The length of the consecutive slot runs auctioned decreases and the incentives diminish after each round. The preliminary rounds end with the auction of a single slot for each service. At the end of the preliminary round each bidder is required to offer the price for the slot committed to during the clock phase. During the final round the bidders reveal the packages they want to reserve; these packages include only the provisional winners from the preliminary slots. Once all provisional winning bids for services in a reservation request are known, the auctioneer chooses the package that best matches the consumer's needs and, at the same time maximizes the profit for the CSP. The *coalition* for a reservation request consists of the set of coalitions that provide the services in the winning package.

In [17] the results of a simulation of the combinatorial auction stage and discuss several metrics of success are reported. These include:

- The customer satisfaction index—percentage of reservation requests fully or partially satisfied in each AS given the total number of requests.
- The service mismatch index—percentage of services requested but not offered in each AS given the total number of services in that slot.
- The service success index—percentage of services used in each AS given all services offered in that slot.
- The capacity allocation index—percentage of the capacity offered but not auctioned in each AS given the capacity offered in that slot.
- The overbidding factor—percentage of slots with a provisional winner that have not been included in any package given all slots offered at the beginning of the auction.
- The temporal fragmentation index—percentage of services successfully auctioned in non-consecutive slots given all services successfully auctioned.
- The additional profit index—percentage of additional profit of coalitions involved in the auction (the difference of the actual price obtained at the auction and the price demanded by the coalition) relative to the price demanded by the coalition.

6 Challenges faced by practical implementation of cloud self-management

Practical application of self-management principles to computer clouds is extremely challenging as discussed in Sect. 2 and in the literature [19–23]. A powerful indication of the challenges posed by practical aspects of self-management is that none of the existing large-scale computing systems can be accurately labeled as self-managing.

Practical implementation of cloud self-management is challenging for several reasons:

6.1 The absence of a technically suitable definition of self-management

A definition that could hint to practical design principles for self-managing systems and quantitative evaluation of the results. Minsky [24] and Gell-Mann [25] have discussed the limitations of core concepts in complex system theory such as emergence and self-organization. The same applies to autonomic computing, there is no indication on how to implement any of the four principles and how to measure the effects of their implementation.

6.2 A quantitative characterization of complex systems and of self-management is extremely difficult

We can only assess the effectiveness of a particular self-management algorithm/protocol indirectly, based on some of the measures of system effectiveness, e.g., the savings in cost or energy consumption. We do not know how far from optimal a particular self-management algorithm is.

6.3 Computer clouds exhibit the essential aspects of complexity; it is inherently difficult to control complex systems

Complex systems: (a) are nonlinear; (b) operate far from equilibrium; (c) are intractable at the component level; (d) exhibit different patterns of behavior at different scales; (e) require a long history to draw conclusion about their properties; (f) exhibit complex forms of emergence; (g) are affected by phase transitions—for example, a faulty error recovery mechanism in case of a power failure took down Amazon's East Coast Region operations; and (h) scale well. In contrast, simple systems are linear, operate close to equilibrium, are tractable at component level, exhibit similar patterns of behavior at different levels, relevant properties can be inferred based on a short history, exhibit simple forms of emergence, are not affected by phase transitions, and do not scale well, see also Chap. 10 of [20].

6.4 Additional factors making even more challenging the application of self-management principles to large-scale computing and communication systems

(1) Abstractions of the system useful for a particular aspect of the design may have unwanted consequences at another level.
(2) Systems are entangled with their environment. The environment is man-made and the selection required by the evolution can either result in innovation, or generate unintended consequences, or both.
(3) Systems are expected to function simultaneously as individual systems as well as groups of systems (systems of systems) [26].
(4) Systems are both deployed and under development at the same time.

Several principles guide our decisions for cloud self-organization discussed in Sect. 5:

I. Take advantage of the properties of market-based mechanisms to ensure system scalability. Base the design on the principle of rational choice; assume that an autonomous server, will always choose the option that maximizes its utility. *Utility* is the measure of the value or the benefit of an action.
II. Devise mechanisms to support an effective reservation system. Reservations are ubiquitous for systems offering services to a large customer population, e.g., airline ticketing, chains of hotels, and so on. Existing clouds, e.g., the AWS, offer both reservations and "spot" instances, with spot access rates lower than those for reservations.
III. Base the design on coalition formation and combinatorial auctions for the reasons discussed in Sects. 1, 2 and 3. Design a system with feedback between the two processes.
IV. Design algorithms for coalition formation that exploit the architecture of the physical system. Take advantage of the rack homogeneity, the servers in one rack are identical in terms of architecture and system configuration, and of faster in-rack communication, the servers in one rack communicate with one another more effectively than with servers from different racks. Coalition formation should enforce *spatial locality*.
V. The objective should be to maximize the profit for the CSP rather than the profit for individual autonomous servers. Exploit rack homogeneity for effective application of cooperative game theory to coalition formation.
VI. Combinatorial auctions [18,27] should support *temporal locality*. Favor service requests for longer sets of consecutive ASs to avoid unnecessary and costly checkpointing and restarting of long-running applications.

7 Summary and future work

In an attempt to address the scalability issues associated with centralized resource management in the cloud, this paper introduces a market-based approach which led to the design of a system of self-managed resources. This design uses a reservation system based on coalition formation and combinatorial auctions. Coalitions constitute pools of homogeneous resources capable of implementing large applications; whereas combinatorial auctions are used to create packages of these coalitions that can implement many such applications combined in a work flow. Even though the resources within a coalition are homogeneous by design, different coalitions may be composed of different resource types. Thus, a collection of coalitions implementing a work flow may be heterogeneous, and chosen to optimally support each phase of that work flow.

Coalition formation is modeled as a cooperative game, and information about former successful coalitions can be used to create new, successful, coalitions. The mechanisms reported in this paper, together with the cloud architecture discussed in [28], attempt to address future challenges faced by the cloud, including support for cloud interoperability and the formation of clouds of clouds.

In [17] we reported on a simulation of the combinatorial auction phase of the reservation system discussed in Sect. 5, we are currently extending the investigation of that system to include subtle interactions between the coalition formation and the combinatorial auction phases. A reservation system for Big Data applications based on coalition formation and combinatorial auctions is discussed in [6] and [7].

Acknowledgements The work reported in this paper was partially supported by NSF CCR Grant 1525943 "Is the Simulation of Quantum Many-Body Systems Feasible on the Cloud?" to Dan C. Marinescu and collaborators and by a Grant from the EU H2020 program to J. P. Morrison for the CloudLightning consortium.

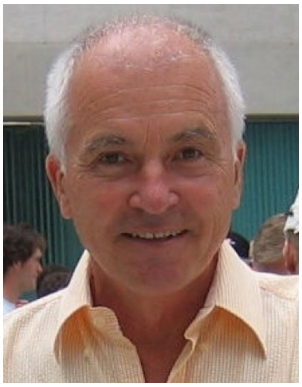
References

1. Barossso, L.A., Clidaras, J., Hözle, U.: The Datacenter as a Computer; an Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool, San Rafael (2013)
2. Chang, V., Wills, G., De Roure, D.: A review of cloud business models and sustainability. In: Proceedings of the IEEE 3rd International Conference on Cloud Computing, pp. 43–50. (2010)
3. Paya, A., Marinescu, D.C.: Energy-aware load balancing and application scaling for the cloud ecosystem. In: IEEE Transaction on Cloud Computing. (2015)
4. Blackburn, M., Hawkins, A.: Unused server survey results analysis. www.thegreengrid.org/media/WhitePapers/Unused%20Server%20Study_WP_101910_v1.ashx?lang=en. Accessed 6 Dec 2013
5. Marinescu, D.C., Paya, A., Morrison, J.P., Healy, P.: Distributed hierarchical control versus an economic model for cloud resource management. arXiv:1503.01061 (2015)
6. Marinescu, D.C., Paya, A., Morrison, J.P.: A cloud reservation system for big data applications. In: IEEE Transaction on Parallel and Distributed Computing. (2016)
7. Marinescu, D.C.: Complex Systems and Clouds: A Self-Organization and Self-Management Perspective. Morgan Kaufmann, Burlington (2016)
8. Müller, I., Kowalczyk, R., Braun, P.: Towards agent-based coalition formation for service composition. In: Proceedings IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 73–80. (2006)
9. Niyato, D., Vasilakos, A., Kun, Z.: Resource and revenue sharing with coalition formation of cloud providers: game theoretic approach. In: Proceedings IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 215–224. (2011)
10. Chaisiri, S., Lee, B., Niyato, D.: Optimization of resource provisioning cost in cloud computing. IEEE Trans. Serv. Comput. **5**(2), 164–177 (2012)
11. Li, C., Sycara, K.: Algorithm for combinatorial coalition formation and payoff division in an electronic marketplace. In: Proceedings AAMAS02—First Joint International Conference on Autonomous Agents and Multiagent Systems, pp. 120–127. (2002)
12. Mashayekhy, L., Nejad, M.M., Grosu, D.: Cloud federations in the sky: formation game and mechanisms. IEEE Trans. Cloud Comput. **3**(1), 14–27 (2014)
13. Sandholm, T.W., Larson, K.S., Andersson, M., Shehory, O., Tohm, F.: Coalition structure generation with worst case guarantees. Artif. Intell. **111**(1–2), 209–238 (1999)
14. Rahwan, T., Ramchurn, S.D., Jennings, N.R., Giovannucci, A.: An anytime algorithm for optimal coalition structure generation. J. Artif. Intell. Res. **34**, 521–567 (2009)
15. Greco, G., Malizia, E., Palopoli, L., Scarello, F.: On the complexity of the core over coalition structures. In: Proceedings of the 22 International Joint Conference on Artificial Intelligence, pp. 216–221. (2011)
16. Shehory, O., Kraus, S.: Methods for task allocation via agent coalition formation. Artif. Intell. **101**(1–2), 165–200 (1998)
17. Marinescu, D.C., Paya, A., Morrison, J.P.: Coalition formation and combinatorial auctions; applications to self-organization and self-management in utility computing. arXiv:1406.7487 (2015)
18. Ausubel, L., Cramton, P., Milgrom, P.: The clock-proxy auction: a practical combinatorial auction design. In: Cramton, P., Shoham, Y., Steinberg, R. (eds.) Combinatorial Auctions. MIT Press, Cambridge (2006)
19. Bradic, I.: Towards self-manageable cloud services. In: Proceedings of the 33 International Conference on Computer Software and Applications, pp. 128–133. (2009)
20. Marinescu, D.C.: Cloud Computing. Theory and Practice. Morgan Kaufmann, New York (2013)
21. Paton, N., de Arago, M.A.T., Lee, K., Fernandes, A.A.A., Sakellariou, R.R.: Optimizing utility in cloud computing through autonomic workload execution. Bull. Tech. Comm. Data Eng. **32**(1), 51–58 (2009)
22. Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, T., Kwiatowska, M., McDermid, J., Paige, R.: Large-scale IT complex systems. Commun. ACM **55**(7), 71–77 (2012)
23. Van, H.N., Tran, F.D., Menaud, J.M.: Autonomic virtual resource management for service hosting platforms. In: Software Engineering Challenges of Cloud Computing, ICSE Workshop at CLOUD09, pp. 1–8. (2009)
24. Minsky, M.: Computation: Finite and Infinite Machines. Prentice Hall, New York (1967)
25. Gell-Mann, M.: Simplicity and complexity in the description of nature. Eng. Sci. **I**(3), 3–9 (1988)
26. Mayer, M.W.: Architecting principles for system of systems. Syst. Eng. **1**(4), 267–274 (1998)
27. Marinescu, D.C., Siegel, H.J., Morrison, J.P.: Options and commodity markets for computing resources. In: Buyya, R., Bubendorf, K. (eds.) Market Oriented Grid and Utility Computing, pp. 89–120. Wiley, New York (2009)
28. Marinescu, D.C., Paya, A., Morrison, J.P., Healy, P.: An auction-driven, self-organizing cloud delivery model. http://arxiv.org/pdf/1312.2998v1.pdf. (2013)

Dan C. Marinescu was an Associate and then Full Professor of Computer Science at Purdue University in West Lafayette, Indiana during the period 1984–2001. Since August 2001 he is a Provost Professor of Computer Science at University of Central Florida. He has published several books and more than 220 papers in referred journals and conference proceedings.

Ashkan Paya got his Ph.D. in EECS from University of Central Florida in August 2015. He graduated from Sharif University of Technology in Tehran, Iran, with a BS Degree in Computer Science in 2011. His research interests are in the area of resource management in large-scale systems and cloud computing.

John P. Morrison is the founder and director of the Centre for Unified Computing. He is a co-founder and director of the Boole Centre for Research in Informatics, a principle investigator in the Irish Centre for Cloud Computing and Commerce and a co-founder and co-director of Grid-Ireland. Professor Morrison has held a Science Foundation of Ireland Investigator award and has published widely in the field of Parallel Distributed and Grid Computing. He is a principle investigator in the Irish Centre from Cloud Computing and Commerce, where he leads the Service LifeCycle Group.



Stephen Olariu has held many different roles and responsibilities as a member of numerous organizations and teams. Much of his experience has been with the design and implementation of robust protocols for wireless networks and their applications. His most recent research interests are in the area of vehicular clouds.