This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

A Cloud Reservation System for Big Data Applications

Dan C. Marinescu and Ashkan Paya
Computer Science Department, University of Central Florida, Orlando, Fl. 32816, USA
Email: [dcm, apaya]@cs.ucf.edu

John P. Morrison
Computer Science Department, University College Cork, Cork, Ireland.
Email: j.morrison@cs.ucc.ie

July 14, 2016

Abstract

Emerging Big Data applications increasingly require resources beyond those available from a single server and may be expressed as a complex workflow of many components and dependency relationships - each component potentially requiring its own specific, and perhaps specialized, resources for its execution. Efficiently supporting this type of Big Data application is a challenging resource management problem for existing cloud environments. In response, we propose a two-stage protocol for solving this resource management problem. We exploit spatial locality in the first stage by dynamically forming rack-level coalitions of servers to execute a workflow component. These coalitions only exist for the duration of the execution of their assigned component and are subsequently disbanded, allowing their resources to take part in future coalitions. The second stage creates a package of these coalitions, designed to support all the components in the complete workflow. To minimize the communication and housekeeping overhead needed to form this *package of coalitions*, the technique of combinatorial auctions is adapted from market-based resource allocation. This technique has a considerably lower overhead for resource aggregation than the traditional hierarchically organized models. We analyze two strategies for coalition formation: the first, *history-based* uses information from past auctions to pre-form coalitions in anticipation of predicted demand; the second one is a *just-in-time* that builds coalitions only when support for specific workflow components is requested.

Index terms – Big Data applications, cloud resource management, hierarchical organization, coalition formation, combinatorial auctions.

1 Introduction and Motivation

The vast number of diverse services offered by modern Cloud Service Providers (CSPs) require an extensive infrastructure consisting of large farms of computing and storage servers in a diverse eco-system supporting several cloud delivery models including Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Moreover, this diversity is increasing with advances in technology:
(a). The cloud infrastructure is becoming more heterogeneous; servers with different configurations of multicore processors and attached co-processors such as GPUs and FPGAs are expected to dominate the cloud computing landscape.
(b). The number of cloud services and cloud applications is ever increasing. For example, in recent years AWS has added new services, including Elastic Cache, and DynamoDB. AWS also offers several Elastic Compute Cloud (EC2) instance types including: C3 - compute-optimized, R3 - memory-optimized; M3 - balanced; G2 - graphics with GPUs; I2 - storage-optimized. Each instance type provides different sets of computer resources measured by vCPUs. A vCPU is a hyper-thread of an Intel Xeon core for C3, R3, M3, G2, and I2 instances.

Efficiently managing resources in a dynamic ecosystem is extremely challenging. Resource management policies support reservation systems, admission control, capacity allocation, load balancing, energy optimization, and quality of service [19]. Existing mechanisms implementing these policies are less effective than they could be and are also not scalable because they require accurate information about the state of individual servers.

Poor resource management can potentially result in high economic and ecological costs. Cloud over-provisioning, the current preferred mechanism for supporting elasticity, demands high initial costs and leads to a low system utilization; this strategy is not economically sustainable [7]. As a result, the average cloud server utilization is in the 18-30% range. The power consumption associated with over-provisioning has been shown to be excessive and has a negative ecological impact [3]. A 2010 survey [4] reported that idle or underutilized servers contribute 11 million tones of unnecessary

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

CO_2 emissions each year and that the total yearly cost for the idle servers is $19 billion.

At this stage in the evolution of compute clouds, a question needs to be posed regarding how far the limits of composability of computing and communication systems can be pushed, while still being able to support the required policies for resource management and effective mechanisms implementing these policies. Hierarchical organization of the cloud infrastructure is ubiquitous [3]. In these systems, resource management decisions require some knowledge of the system state, but the reliability of state information degrades as we move from the servers to the upper levels of the system hierarchy. Moreover, the value of this information is ephemeral, if not acted upon immediately wrong decisions may be taken, since the state may change rapidly. In contrast to current approaches, market-oriented mechanisms for resource management are scalable, self-regulating, and are widely used in many areas of human activity.

We are currently working on a Big Data project in condensed-matter physics [22]. The space, computation, and communication complexity of the tensor network contraction algorithm at the heart of this project, increases dramatically from one iteration to the next. Only an application-centric resource allocation mechanism, similar to the one discussed in this paper, can minimize waste and support algorithm optimization by balancing computing and communication costs.

The contributions of this paper. We address the problem of efficiently determining the most appropriate heterogeneous cloud resources to solve Big Data problems expressed as a workflow of service components. We acknowledge the scalability issues associated with traditional, centralized, resource managers that rely on, often inaccurate, monitoring information to make resource allocation decisions. In effect, the evolving complexity of the cloud is increasingly falling short of the received wisdom from control theory that tells us that accurate state information and a tight feedback loop are the critical elements of an effective control system. In essence, *only local information, used locally, is reliable*. In deference to this principle, we focus on market-oriented resource allocation for the IaaS cloud delivery model. In our system, coalitions of servers use local state information to make local decisions, which have beneficial emergent global properties. In previous work, we investigated the use of market-oriented mechanisms in a large-scale computing system and reported that a simple bidding scheme is much more effective than hierarchical management based on monitoring [20]. We also investigated the effectiveness of combinatorial auctions and in [21] we reported a better than 80% success rate.

Market mechanisms address the tensions between local and global objectives. These tensions manifest themselves in such questions as: How can we balance value to the user with CSP profit? How can we dynamically adapt the price for resources to the real-time demand? The market-based approach gives us the tools to begin addressing these, and other challenging, questions.

We recognize the need for a holistic solution to resource management and, in particular, that its mechanisms should be complemented by specific characteristics in the physical system organization. To this end, we assume the warehouse scale computer organization described in [3] and add constraints to support effective local decision-making. We introduce the concept of server coalitions to execute workflow components and two strategies for coalition formation. The first strategy is *history-based* (HB) and uses historical information to pre-form coalitions that have been used successfully in the past. To our knowledge this is the first attempt to address cloud resource management based on coalition formation and combinatorial auctions where individual servers learn from past behavior. The second, *just-in-time* (JST) strategy, forms coalitions dynamically to meet real-time demands. This paper focuses on rack-level coalition formation; the subsequent aggregation of these coalitions into packages to support the complete workflow using combinatorial auctions is presented in [21]. Formation of coalitions and of coalition package are the two stages of a resource reservation system.

In passing we note that, distinct from the ideas presented here, the research literature makes reference to coalition formation *for cloud federations*. Our coalitions have a short life-span. They cease to exist once the service component they are executing terminates. Our system allows for free resources to choose to offer themselves on the *spot market*, or to wait to join a coalition in preparation for the next auction. Long-term coalition stability is critical for cloud federations, but is a non-issue for our coalitions. In addition, locality is important for us, but is a non-issue for cloud federations that use the Internet to communicate. We focus on rack-level coalitions and assert that communication across multiple hierarchical layers of the cloud infrastructure is undesirable to avoid increased latency and falling bandwidth.

The solution discussed in this paper involves concepts, policies, and algorithms from several well-established areas of economics and computer science: coalition formation and virtual organizations; auction theory and practice; system organization and computer architecture; and self-organization and self-management of complex systems. Related work and the system model are discussed in Sections 2 and 3, respectively. Algorithms for rack-level coalition formations and the results of simulation experiments are presented in Sections 4 and 5, respectively. Conclusions, future work, and the feasibility of self-organization are discussed in Section 6.

2 Related Work

Existing cloud resource management systems. Cluster management systems such as Borg [36] and Omega [33] are used by Google in its cloud infrastructure. Google supports containers allowing cloud users to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

run their applications in a resource-isolated manner.

Kubernetes, is a system developed at Google for managing containerized applications across a cluster of nodes [14]. Docker uses *cgroups* to group processes running in the container. Amazon and Google support the creation of Docker-based containers [1, 9]. Twitter's infrastructure is managed by Mesos [11]. A storage management system used by VMware is described in [35].

A 12,000-server Google cluster, managed by the Borg system, achieves aggregate CPU utilization of 25-35% and aggregate memory utilization of 40% [8]. The aggregate CPU utilization of systems using Mesos is consistently below 20%, even though reservations reach up to 80% of system capacity. The Quasar system developed at Stanford University improves resource utilization in a 200-server EC2 cluster by 47% [8].

Existing systems can manage clusters with tens of thousand servers but the challenges outlined in Section 1 persist and motivate the search for effective and scalable policies and mechanisms for cloud resource management [5, 6, 16, 23, 24, 28, 39]. To respond to the needs of increasingly more complex applications consisting of multiple phases and requiring workflow management, CSPs are already offering workflow management services such as SWM (Simple Workflow Management) and EBS (Elastic Bean Stock) at AWS.

Market Mechanisms. In spite of all the advantages of market mechanisms and the vast literature on market-based resource allocation [18], the practical application of these ideas is rarely seen [30]. For many years the research continued to be focused on global optimization of system-centric metrics such as mean average job completion time, throughput, and system utilization, while now considerations regarding user preferences including cost, security, and quality of service (QoS) have to be added to the mix.

Our recent results confirm that the communication complexity of hierarchical control with resource monitoring is more than two orders of magnitude higher than that of a simple bidding scheme [20]. Hierarchical control is quite inefficient for a cloud consisting of multiple Warehouse-Scale Computers (WSCs).

Coalition formation is a widely used method for increasing the efficiency of resource utilization and for providing convenient means to access these resources. In recent years, the emergence of large-scale electronic markets, grid and cloud computing, sensor networks, and robotics have amplified the interest in coalition formation and virtual organizations [15, 29, 31].

Different aspects of resource management in computational grids including load balancing, job-allocation, and scheduling, as well as revenue sharing when agents form coalitions or virtual organizations are discussed in [10, 13, 25, 38]. Grid resource allocation is modeled as cooperative games [13] or non-cooperative games [25]. Resource co-allocation is presented in [38].

There is little surprise that the interest in coalition formation migrated in recent years from computational grids to cloud resource management. The majority of the on-going research in this area is focused on game-theoretic aspects of coalition formation for cloud federations. A *cloud federation* is a set of CSPs collaborating to provide services to a cloud user community. A stochastic linear programing game model for coalition formation is presented in [24]; the authors analyze the stability of the coalition formation among cloud service providers and show that resource and revenue sharing are deeply intertwined. An optimal virtual machine (VM) provisioning algorithm ensuring profit maximization for CSPs is introduced in [6].

A cloud federation formation described as a hedonic game and focused on the stability and the fairness of the game is discussed in [23]. The profit maximization for each federation is formulated as an integer programming problem and the game is augmented with a preference relation over the set of federations. The paper adopts a payoff division based on the Banzhaf value. Currently, individual CSPs may believe that they have a competitive advantage due to the unique value of their services and may not be motivated to disclose relevant information about the inner working of their systems. Thus, the formation of cloud federations seems uncertain, at least in the immediate future.

A combinatorial coalition formation problem is described in [15]. The 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. An algorithm to find optimal coalition structures in cooperative games by searching through a lattice like the one in Figure 1 is described in [26]; in this algorithm the coalition structures are grouped according to so-called *configurations* reflecting the size of the coalitions.

Auctions. Auctions are a widely used mechanism for resource allocation with numerous applications such as: the auctioning of airport take-off and landing slots, spectrum licensing by the Federal Communication Commission, and industrial procurement. An online auction mechanism for resource allocation in computer clouds is presented in [38].

Different phases of an application may require coalitions of servers with different types of resources so there is the need to investigate combinatorial auctions where packages of items are auctioned. A *combinatorial auction* is one where a buyer requires simultaneous access to a *package* of goods. An auction allows the seller to obtain the maximum available profit for the auctioned goods; it is organized by an *auctioneer* for every *request* of a consumer. A *proxy* is an intermediary who collects individual bids from the buyers participating in an auction, computes the total cost of the package from the bids, and communicates this price to the auctioneer. A vast literature including [2, 37] covers multiple aspects of combinatorial auctions including bidding incentives, stability, equilibrium, algorithm testing, and algorithm

optimality.

Package bidding assumes that a seller offers $\mathcal{N}$ different types of items. A buyer bids for packages of items. A *package* is a vector of integers $\mathcal{Z} = \{z_1, z_2, \ldots, z_{\mathcal{N}}\}$ which indicates the quantity of each item in the package; the price of items is given by $\mathcal{M} = \{m_1, m_2, \ldots, m_{\mathcal{N}}\}$.

Package bidding can be traced back to generalized Vickerey auctions based on the Vickerey-Clarke-Groves mechanisms. In Vickerey auctions a bidder reports its entire demand schedule. The auctioneer then selects the allocation which maximizes the total value of the package and requires a bidder to pay the lowest bid it would have made to win its portion of the final allocation, considering all other bids.

In a *clock auction* 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 more excess demand. On the other hand, when the offering exceeds the demand, the price decreases. In a clock auction the bidding agents see only aggregate information, the price at a given time, and this eliminates collusive strategies and interactions among bidding agents. The auction is *monotonic*, the amounts auctioned decrease continually and this guarantees that the auction eventually terminates. When the price of a package can be computed as the sum of products of prices and quantities it is said that an auction benefits from *linear pricing*.

The *clock-proxy-auction* is a hybrid auction based on an iterative process with two phases [2]. A *clock* phase is followed by a *proxy* round. During the proxy round the bidders report the values they have submitted to the proxy which in turn submit bids for the package to the auctioneer. A bidder has a single opportunity to report the quantity and the price to the proxy, bid withdrawals are not allowed, and the bids are mutually exclusive. The auctioneer then selects the winning bids that maximize the seller's profit.

Autonomic computing and self-organization. The recognition of the challenges posed by the design and implementation of large-scale systems lead to new research areas closely related to self-organization such as autonomic computing, proposed by IBM in 2003, and organic computing. Over 8,000 papers, nearly 200 conferences, and some 200 patents issued and more than 100 pending, are some of the results of a decade long research effort in autonomic computing [12].

Small systems based on these ideas have been described in the literature, but scalability of the solutions proposed seems to be an insurmountable obstacle, as evidenced by the fact that no such large-scale systems exist today. Indeed, this is an indication of the challenges to a practical implementation of these concepts in systems with a very large number of components interacting with each other in unpredictable ways and operating in a dynamic environment with a large and diverse user population.

Surely, an important reason for the slow progress of solutions of this type is the absence of suitable technical definitions of these concepts, definitions that would lead to practical design principles for self-organizing systems and to the quantitative evaluation of the results. A powerful definition of self-organization was given by Turing "the spontaneous emergence of global coherence out of local interactions" [34]. However, this definition contains no hint at these elusive design principles.

The design principle may not be explicit but at least we have many real examples of self-organization. Market-based ecosystems, for example, exhibit this type of behavior. There, individuals with competing interests and inherent conflicts interact to form a state of dynamic equilibrium.

The coalitions and combinatorial auctions introduced in this paper differ from those discussed in the grid and cloud computing literature. Coalitions are based on cooperative games and are short-lived. Combinatorial auctions involve multiple buyers and sellers.

3 System Model

The demand for computing resources will increase for Big Data applications and could be considerably larger than a single server can provide. Only coalitions of servers will be capable of offering such resources. For optimum performance the members of a coalition should communicate effectively thus, they must be be in close proximity to each other. This constrains the coalition formation protocol.

System architecture. We assume a hierarchical organization of the cloud infrastructure similar to the one described in [3] in which a data center consists of multiple warehouse-scale computers (WSCs). Each WSC has multiple cells, each cell has multiple racks and each rack houses multiple servers. A WSC connects 50,000 to 100,000 servers and uses a hierarchy of networks.

The servers are housed in racks; typically, the 48 servers in a rack are connected by a 48 port Gigabit Ethernet switch. The switch has two to eight up-links which go to higher level switches in the network hierarchy [3]. The bandwidth for communicating outside the rack is much smaller than the one within the rack; this has important implications for resource management policies and becomes increasingly difficult to address in systems with a large number of servers.

Model assumptions. We assume that the servers in a rack have identical processors with the same number of cores, the same amount of main storage, cache, and secondary storage and an identical configuration of GPUs, FPGAs, or other hardware. We also assume that all servers in a rack support the same type of services. Our WSC is heterogeneous, the servers in different racks may have different architectures and configurations. The same service may be offered by multiple racks, possibly using different hardware; for example, a rack could offer a service using GPUs, whereas, another may offer the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

same service using FPGA. This heterogeneity in the service offering introduces an element of choice making it possible to match the user requirements and constraints with the available resources.

The system we envision supports both *reservation* and *spot* resource allocation. The reservation system has two stages: (A) coalition formation, and (B) combinatorial auctions. Spot allocation is supported by a bidding process for each type of resource. Time is quantified in *allocation slots* and reservations are made as a result of auctions carried out at the beginning of each slot; for example, an allocation slot could be one hour.

4 The Reservation System

Coalition formation as a cooperative game. Coalition formation can be modeled as a cooperative game where the goal of all servers is to maximize the reward for the CSP, rather than being selfish and competing with one another. Here, coalitions are formed in all racks of a WSC before an auction. We discuss the formation in one rack with N servers: $\{s_1, s_2, \ldots, s_N\}$.

A *coalition* $\mathbb{C}_i$ is a non-empty subset of N servers. A *coalition structure* $\mathbb{S} = \{\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_m\}$ is a set of m coalitions satisfying the following conditions:

$|\bigcup_{i=1}^{m} \mathbb{C}_i| = N$ and $i \neq j \Rightarrow \mathbb{C}_i \bigcap \mathbb{C}_j = \emptyset$.

Figure 1 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; in our example at level $L1$ there is one coalition structure $\{s_1, s_2, s_3, s_4\}$ and at $L2$ there are groups of two coalitions, e.g., $\{s_1\}, \{s_2, s_3, s_4\}$. The number of coalitions structures at level k for a population of N servers is given by the Stirling Number of the Second Kind:

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

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

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

The total number of possible combinations of coalitions structures increases exponentially with the number of servers. Searching for the optimal coalition structure $\mathbb{C}$ is computationally challenging due to the size of the search space, e.g. , for n=11 and n=12 there are 115,975 and respectively 678,570 possible combinations.

History-based rack-level coalition formation. *History-based* (HB) coalition formation is based on the learning process shown in Figure 2. In contrast to the cooperative game, it would appear to be more natural to build coalitions based on current user demands and we call this strategy *just-in-time* (JST). In Section 5 we compare these two strategies.

Recall from Section 3 that in our model all servers in a rack have the same architecture and identical configuration. This realistic assumption considerably simplifies the complexity of the search for an optimal coalition structure as the servers within a rack are indistinguishable from one another. We have a system with two stages and feedback, see Figure 2. In the second stage the coalitions created during the first stage are included in successfully auctioned packages thus, we can determine precisely the value of all coalitions structures.

Coalitions have a short lifetime; a coalition participates in an auction and, if successful, the coalition persists for the duration of the contract. Servers of an unsuccessful coalition are free to participate in the coalition formation process in the next allocation slot or to offer their services on the spot market during the current allocation spot. The revenue earned by a coalition is evenly distributed among its members. The income from a package of services supported by several coalitions is divided according to the resources supplied by each one of them.

The coalition formation process involves four steps: (1) Selection of a rack leader; (2) Ranking of coalitions most likely to succeed based on historic performance data; (3) Assignment of available servers to coalitions according to their bids; and (4) Sharing information about successful coalitions after an auction.

The role of rack leader is passed from one server to the next to balance the load and to ensure fault-tolerance. The rack leader may or may not be included in any coalition. Selection of a rack leader can be done automatically to avoid a communication-intensive election process. For example, when the server IDs are $\{1, 2, \ldots, j, \ldots, S\}$ then in slot k the role of rack leader can be automatically assigned to the server with ID $j = k \mod S$. To further reduce communication complexity all servers maintain a list of previously successful coalitions. The list is ordered according to the revenues brought in by coalitions with the same number of servers. In case of a tie the coalition with the largest number of servers is placed before the others. For example, such a list could be $\mathcal{R} = \{6(75), 11(40), 8(43), \ldots, 2(97), 1(198)\}$ meaning that historic data show that coalitions of 6 servers were successful 75 times and brought the largest revenue, while coalitions of 1 server were successful 198 times and brought the least amount of revenue. The list could be extended to include the most likely combinations of successful coalitions.

At the beginning of an allocation slot the rack leader receives bids from the N_a servers which are not members of an active coalition and then chooses the coalition set to compete for service packages in the current allocation

[1] For $N = 5$ and $N = 6$ the Stirling Numbers of the Second Kind are respectively $1, 15, 25, 10, 1$ and $1, 31, 90, 65, 15, 1$.

[2] The Bell numbers B_N, (1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, ...) describe the number of ways a set with N elements can be partitioned into disjoint, non-empty subsets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

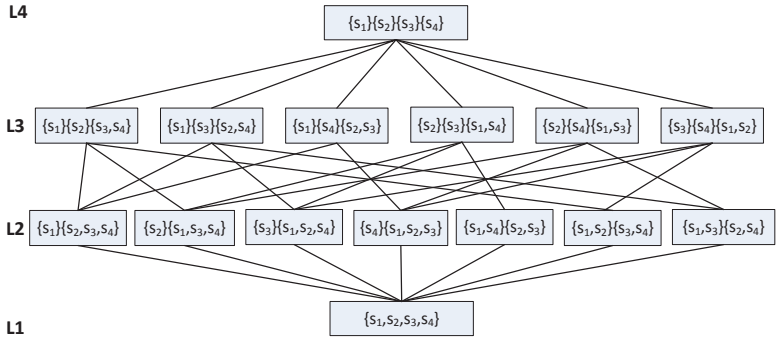

Figure 1: A lattice with four levels $L1, L2, L3$ and $L4$ shows the coalition structures for a set of 4 servers, s_1, s_2, s_3 and s_4. The number of coalitions in a coalition structure at level L_k is equal to k.

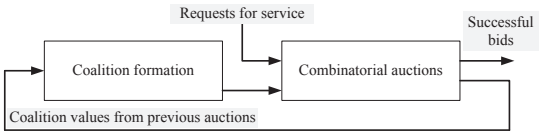

Figure 2: HB - feedback about past coalitions determines the current coalition structures.

slot. For example, if $N_a = 27$ this set could be $\mathcal{C} = \{6, 11, 8, 2\}$, a subset of $\mathcal{R}$. The leader ranks the bids based on the revenue reported by each bidder and assigns the first 6 servers to the first coalition, the next 11 to the second coalition, and so on. Finally, after the auction, the leader broadcasts the information about successful coalitions, e.g., $\{6, 8\}$ and verifies that the leader for the next allocation slot is ready to assume its role.

Just-in-time (JST) coalition formation. The service requests are dispatched to a WSC where they are analyzed to determine the service type and the desired coalition size for each. This information is then broadcast to the racks of the WSC where an attempt is made to dynamically create appropriate coalitions. The algorithm for coalition formation is similar to the one for the HB strategy. The only difference is that the coalition sizes are now the ones demanded by the service request rather than those determined by past activity. Once the coalitions are formed, all racks of the WSC independently bid for all service requests. An auction takes place and the racks hosting successful coalitions are informed.

The combinatorial auction protocol. The protocol is based on the clock algorithm discussed in Section 2. The auctioneer announces prices and bidders indicate the quantities they wish to buy at the current price. Then the auctioneer adjusts the prices based on the current demand. In our simulation, after coalition formation, coalitions start bidding on the service request and the auctioneer analyzes these bids and prioritizes them based on the prices they offered.

Several criteria drive the decisions of the auctioneer when selecting the coalitions for the packages required by the workflow. These decisions are based on the price discovered during the clock phase. Our implementation uses a hierarchical decision making process. The decision factors in priority order are: the *coalition size* - the larger the coalition size, the higher the priority; the *requested service duration* - the longest duration has the highest priority.; the *rack load* - the lower the rack load, the higher the priority; and finally *cell load* - the lower the cell load, the higher the priority. The rack and cell load refer to the load allocated to the servers in the rack and cell, respectively. If two coalitions with the same size and same duration, but with different rack load levels bid for a request, the priority is given to the coalition from the rack with the lower load. This decision process aims to balance the load on racks and cells.

Next section reports on experiments to compare JST and HB coalition formation algorithms.

5 Simulation Experiments

Motivation. The systems we are considering have several million components and, at this scale, numerical simulation is the only realistic investigative approach. Analytical modeling is not feasible due to the complexity of the interactions among the large number of components. We acknowledge the limitations of our approach and concede that empirical results produced on a testbed configuration may, or may not, hold true at scale. As pointed out in [3] "..the WSCs are a new class of large-scale machines driven by new and rapidly evolving sets of workloads. The size alone makes them difficult to experiment with, or to simulate effectively...".

Simulation assumptions and system model. Several simplifications are necessary to simulate a cloud infrastructure with several WSCs. (1) Auctions are or-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

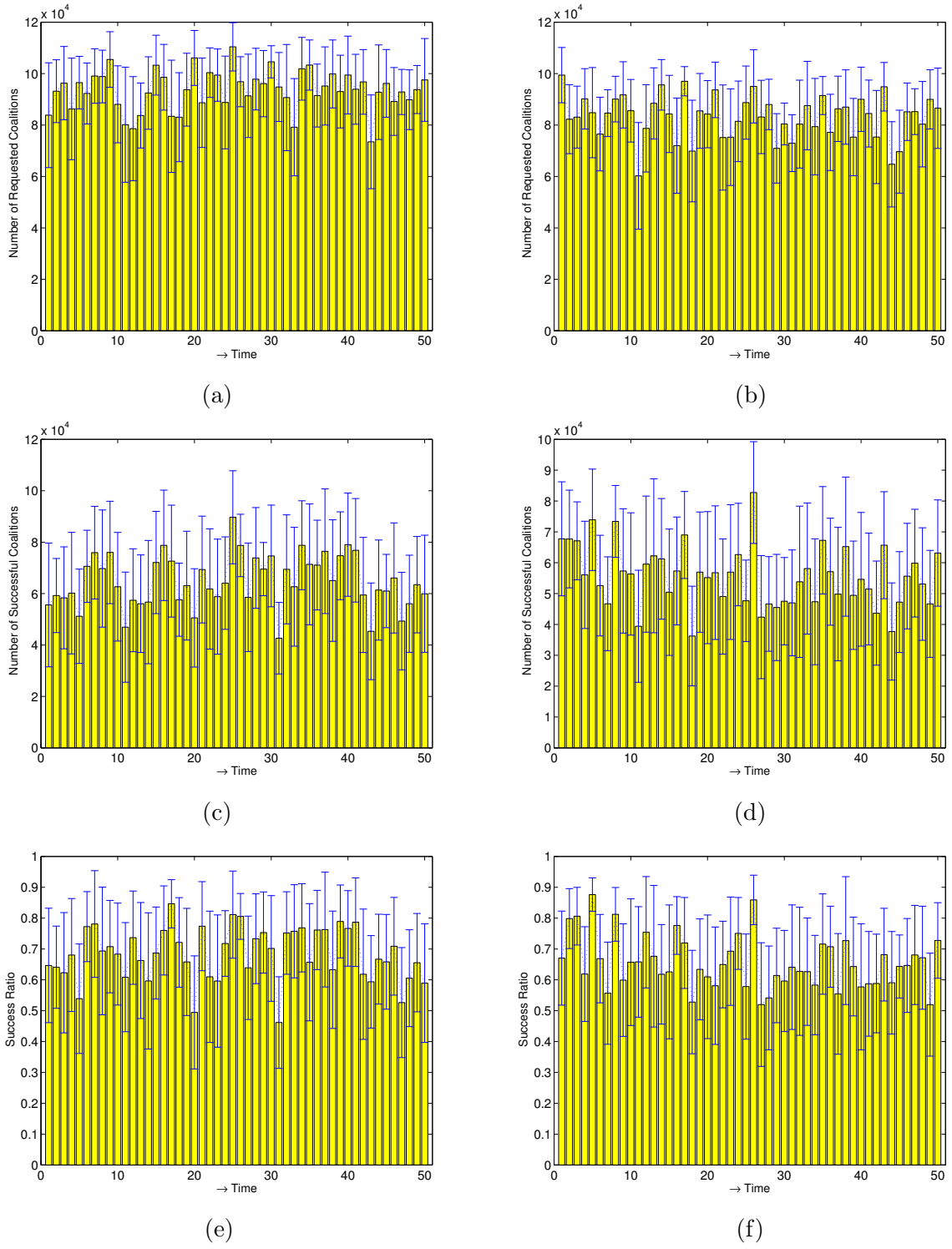

Figure 3: Time series of the number of coalitions requested, successful coalitions, and the success ratio at 20% initial system load for all service types. 95% confidence intervals are shown. Strategies for coalition formation: HB (a)-requested (c)-successful, and (e)-success ratio; JST (b)-requested, (d)-successful, and (f)-success ratio.

ganized periodically and the time between two consecutive auction is called an *allocation slot*. We require a service request to specify the service duration as an integer number of slots. The number of auctions is equal to the number of allocation slots. (2) We measure the communication complexity of the two reservation systems discussed, HB and JST, by the number of messages they require and avoid a detailed timing analysis

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

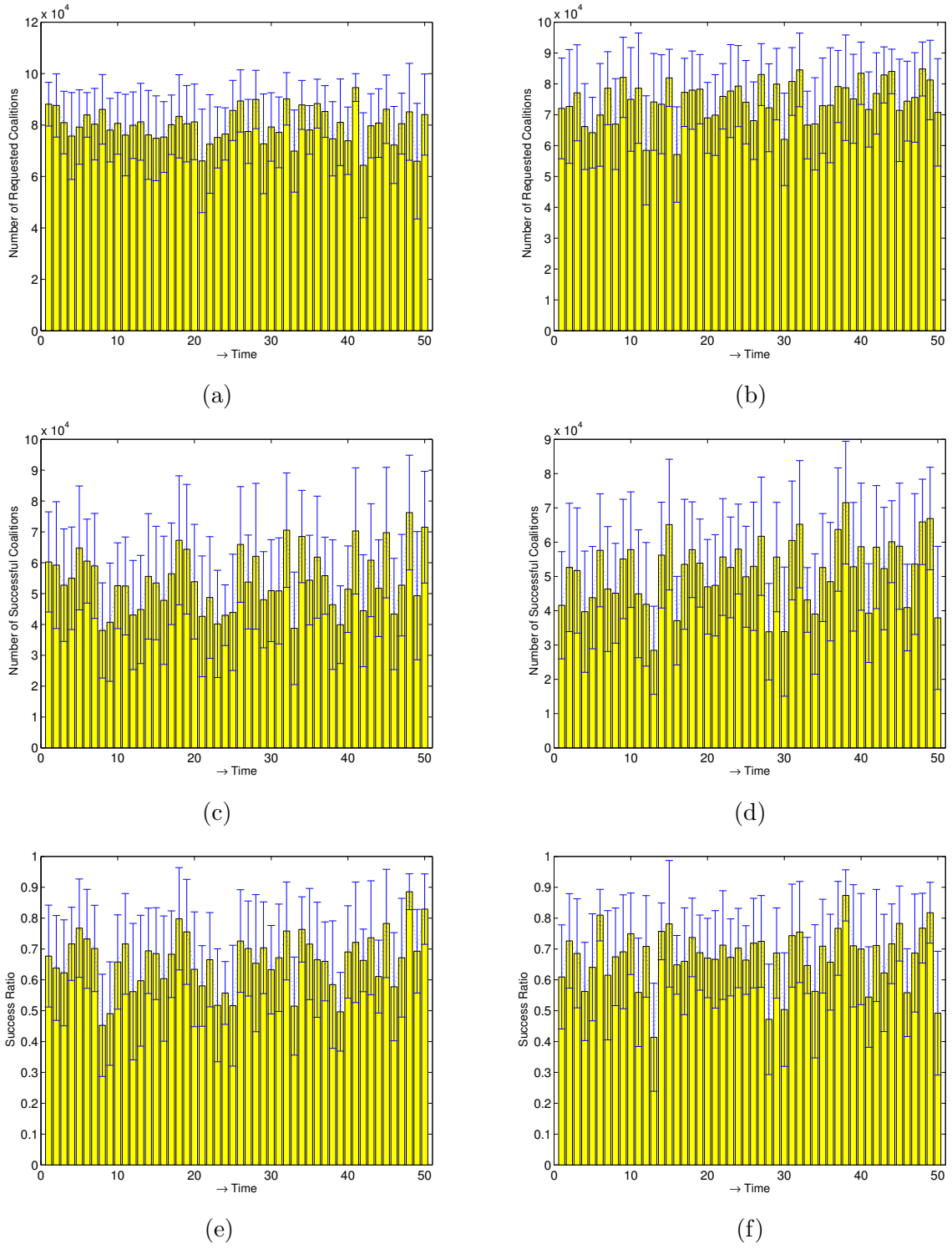

Figure 4: Time series of the number of coalitions requested, successful coalitions, and the success ratio at 80% initial system load for all service types. 95% confidence intervals are shown. Strategies for coalition formation: HB (a)-requested (c)-successful, and (e)-success ratio; JST (b)-requested, (d)-successful, and (f)-success ratio.

of the communication delays, which would require modeling contention at different levels of the network hierarchy. (3) Lastly, we simulate the system for only 500 time slots for practical reasons. A longer history would lead to more accurate predictions of the size of successful coalitions for each service type thus, we expect the results to improve in time for the HB strategy.

We found some data in the literature to guide our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

Table 1: Statistical results for the two coalition formation strategies, history-based (HB) and just-in-time (JST), for two different initial loads, 20% and 80%. The mean and the variance of the number of coalitions requested and the number of successful coalitions chosen during the auctions, and the success ratio (SR). Results are shown for: (Left) all types of service requests; (Right) two randomly chosen types.

Method	Initial load	Stats	Requested all	Successful all	SR all	Requested two types	Successful two types	SR two types
HB	20%	Mean/ Std	93,648.96/ 7,691.05	64,805.99/ 10,134.88	68% 0.09	11,991.71/ 1.076.32	7,124.12/ 1,101.33	60% 0.09
	80%	Mean Std	79,882.37/ 6,540.43	54,192.35/ 9,786.67	66% 0.09	10,421.61/ 1,037.72	5,957.67/ 1,149.43	57% 0.08
JST	20%	Mean/ Std	83,194.86/ 8445.98	55,565.27/ 9787.62	66% 0.09	10,689.07/ 881.93	60,68.37/ 1,035.98	57% 0.09
	80%	Mean Std	74572.12/ 6602.59	51381.41/ 9699.45	67% 0.09	9672.03/ 924.87	5,690.32/ 1,128.16	59% 0.09

Table 2: Communication complexity of HB and JST for all service types and two different system initial loads, 20% and 80%. Number of messages: Nr - at rack level; Nt - at all levels; SRr - per service request at rack level; SRt -per service request at all levels; per successful coalition formation at: SCFr- rack level and SCFt - all levels.

Method	Load	Nr	Nt	SRr	SRt	SCFr	SCFt
HB	20%	18,823,492	46,824,480	201	500	290	722
	80%	6,299,012	19,970,590	79	250	116	368
JST	20%	8,129,402	20,798,714	98	250	146	374
	80%	14,241,563	37,286,059	191	500	277	725

choice of the system configuration [3] and where no such guidance was available, we used sensible choices to describe the workload. These choices included: the number of coalitions in each package of services in a client's request, the number of different service types provided by the system, service intensity measured by the number of vCPU requested, and service duration. A service type can refer to a specific service, e.g., Map-Reduce, or a generic type, e.g., CPU-intensive. We used uniform distributions for all random variables. For example, the number of coalitions in a package is uniformly distributed in the range 5-25. Service requests are generated continually and all service requests, arriving after an auction has begun, waited to participate in the next auction. A coalition request must specify the service type, the service duration, and the service intensity expressed as the number of vCPU. The last parameter allowed us to determine a sensible coalition size.

The simulated system consists of 4 WSC, each one with 25 cells; each cell has 100 racks and each rack has 40 servers. The server capacity, measured by the number of vCPU, is uniformly distributed in the range of 10 - 50 vCPU. The system provides 20 different service types. A request for service specifies several coalitions and for each one the service types: service intensity, and service duration. The service intensity is uniformly distributed in the range 10-500 vCPU and the service duration is uniformly distributed in the range 5-25 allocation slots. A service request unsatisfied in an allocation slot participates in the auction organized in the next slot.

Simulation results. Figures 3 and 4 display time series of the number of coalitions requested, coalitions successful during the auctions, as well as, the success ratio for two initial system loads, 20% and 80%, respectively, when all services are taken into account. It is rather difficult to control the average system load and we only report the initial system load. The duration of the simulation is 500 allocation slots and each bin shows the average over 10 of them. The mean and the variance of the three simulation results, the numbers of coalitions requested and coalitions formed, and the success ratios are summarized in the left columns of Table 1.

The average number of successful coalitions decreases when the load increases, about 16% for HB and about 7% for JST. This difference is not reflected in the success ratios of the two strategies; as we shall see next, it is most likely a sign that HB better reflects the internal state of the system than JST. The coalition formation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

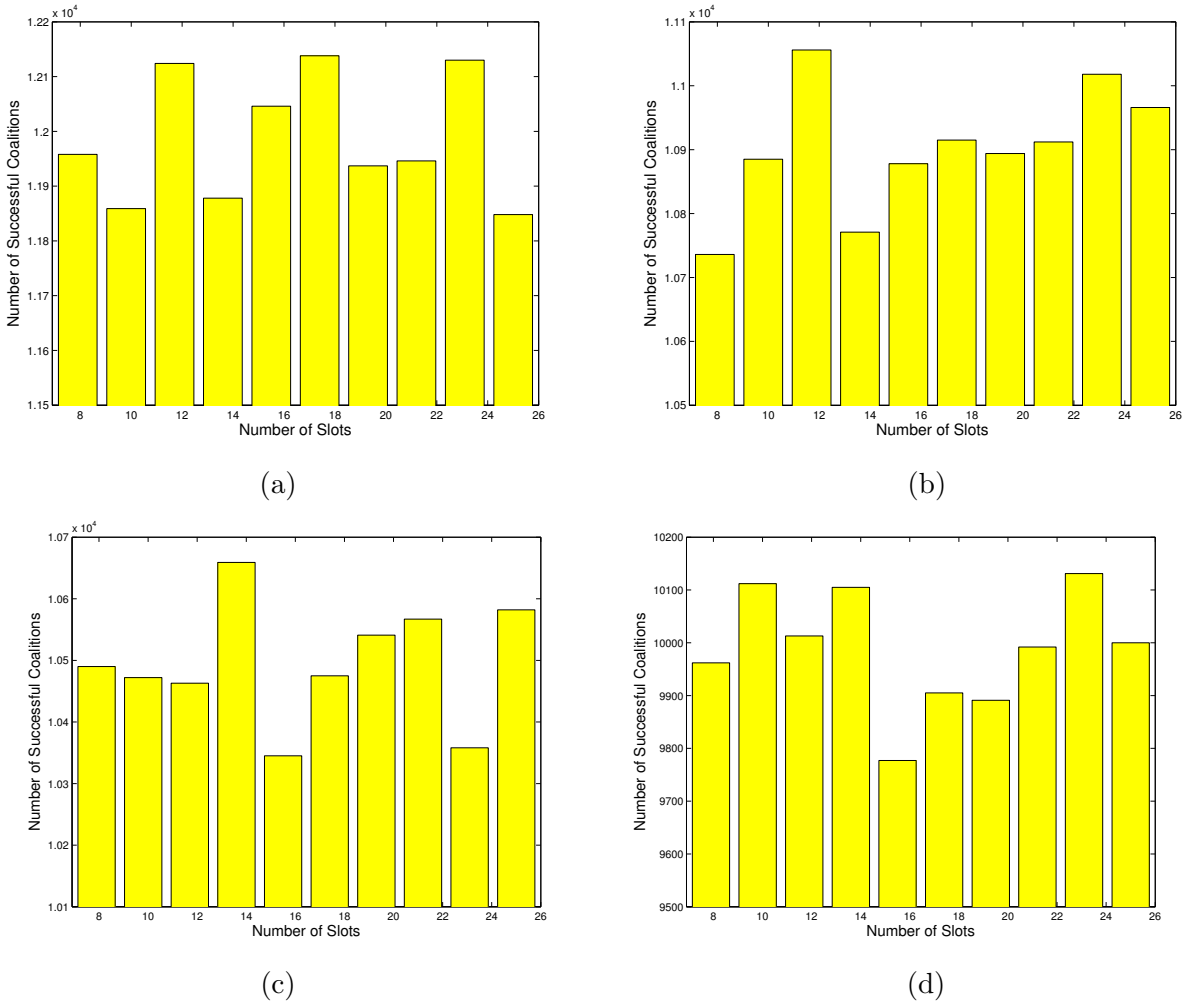

Figure 5: Histograms of coalition life-time. Average initial load 20% of system capacity: (a) HB; (b) JST. Average initial load 80% of system capacity: (c) HB; (d) JST.

success ratio varies only slightly with the load. When the initial average system load increases from about 20% to 80% the success ratio decreases from 68% to 66% for HB and increases from 66% to 67% for JST. *The results show that a resource utilization of* 80% *of the system capacity can be supported, much higher than the average one reported for existing systems; they also indicate that both market-based strategies are robust.*

Table 2 shows the communication complexity at the low and high average initial system load for the two strategies HB and JST. We differentiate between messages exchanged at the rack level from those at higher levels of the network hierarchy (cell and WSC), where contention for network access is more intense. As expected, the total number of messages for the HB strategy decreases significantly from about 47 million to 20 million, as the load increases, because fewer servers are available for coalition formation. The situation is reversed for JST, we notice a sharp increase from about 21 to 37 million. This is expected because the coalition formation process is driven by the external service requests. Note that in the case of JST, 70% of all messages exchanged at high load are at the cell and WSC level, where contention for communication bandwidth is considerably higher.

In HB, the coalition formation process is driven by the internal state: only servers with available capacity participate in coalition formation and subsequently in the combinatorial auctions, while in case of JST the process is driven by external factors. At low load, the number of rack-level messages per either service request, or successful coalition formation, is almost twice as large for HB than for JST, e.g., 201 versus 98 and 290 versus 146, respectively, see Table 2. At high load the situation is reversed 79 versus 191 and 116 versus 277.

The same observation applies to the total number of messages per service request and per successful coalition formation. At low load the ratios for HB and JST are 500 versus 250 and 722 versus 374, respectively. At high load the ratios are 250 versus 500 and 368 versus 725.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2594783, IEEE Transactions on Parallel and Distributed Systems

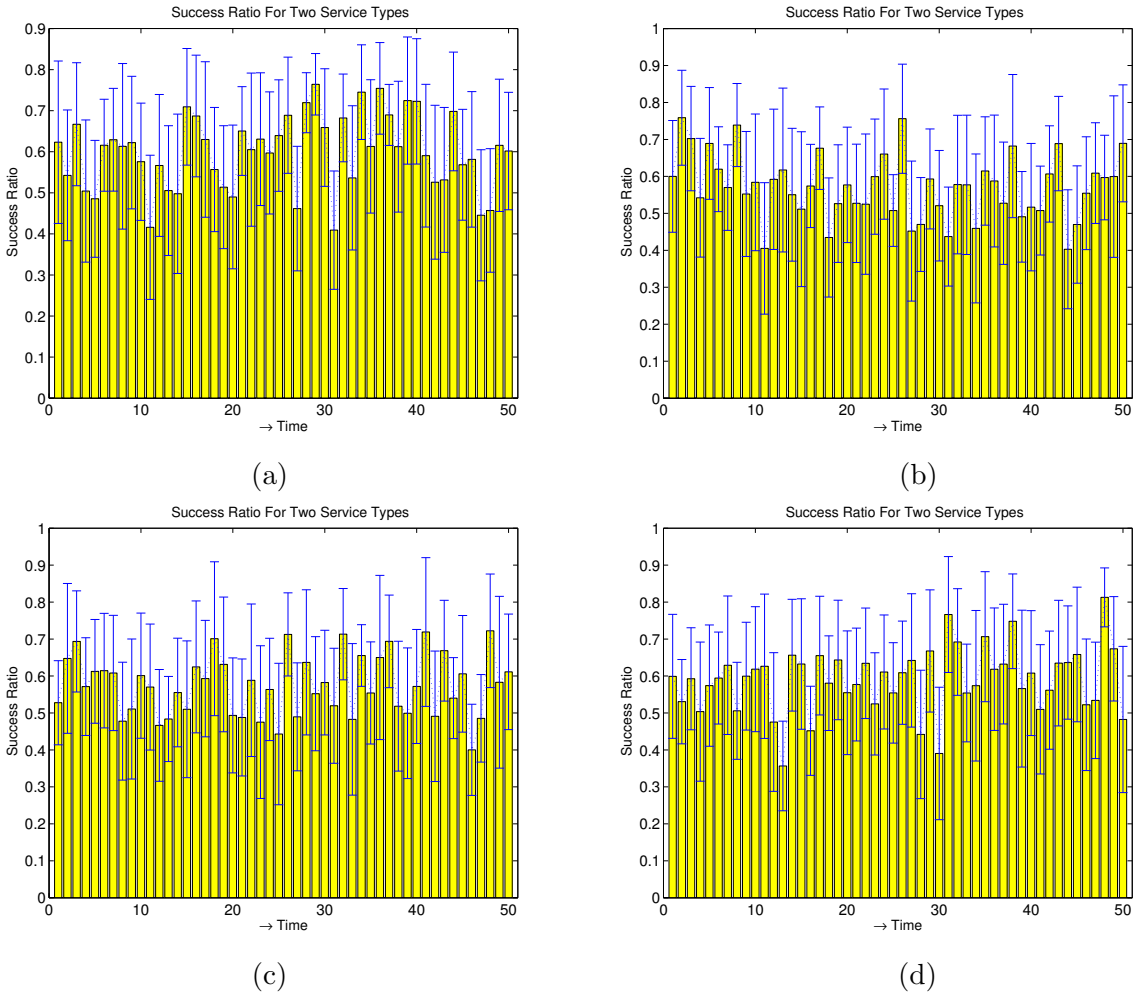

Figure 6: Time series of the success ratio for two randomly selected service types with 95% confidence intervals. Initial load: (Top) 20%; (Bottom) 80%. Two strategies for coalition formation: (a) and (c) HB; (b) and (d) JST.

Figures 5 presents the life-time of coalitions for all service requests when the initial load is 20% and 80% of system capacity, respectively, for the HB and JST reservation systems. This life-time of a successful coalition reflects the duration of the corresponding service request.

Lastly, we randomly selected two particular types of service and investigated their behavior. The statistical results are summarized on the right side of Table 1 and show that indeed the number of requests, as well as, the number of successful coalitions is about one tenth of the total number of the corresponding entries for all service requests. Figure 6 shows time series of the success ratios for the two randomly selected service types.

The big picture. While we can model in great detail the operation of an individual server, the interactions among the servers of a WSC is what gives the cloud infrastructure its flexibility and power. The model should also describe the interactions of the infrastructure with an environment consisting of a large population of users, rather than describing the infrastructure in isolation.

A realistic alternative is to develop high-level rather than detailed models of the cloud infrastructure, conduct simulation experiments using these models, and draw qualitative conclusions from the simulation results. Such models must be carefully crafted and avoid details that unnecessarily complicate the simulation, or make it infeasible. The goal should be to identify trends and determine the effect of different model parameters to gain insights based on a qualitative, rather than a quantitative analysis of the simulation results.

The question we discuss now is if the system organization and the algorithm we propose have the potential to outperform existing solutions for cloud resource management. The three most important criteria for comparison are scalability, performance under heavy load, and the ability to supply resource packages tailored to specific user demands. The results presented in this section show that our algorithms and protocols are scalable well beyond the 12,000 Google cluster managed by Borg or the 200 cluster managed by Quasar. In [20] and in [21] we show that a simple bidding scheme scales well for

multiple WSCs and outperforms a hierarchical management scheme.

The results in Table 1 and the time series presented in Figures 3 and 4 show that under heavy load the system performance is similar to the one under light load, while typical results reported in the literature show average system utilization in the 20% range. The communication complexity thus, the overhead for resource management, decreases under heavy load and this attests to the virtues of the HB strategy, see Table 2. The energy efficiency measured as the number of operations per Watt is dramatically better for the system we propose. Even under heavy load the system we propose is application-centric and able to offer precisely the amount of resources needed by the application, for the desired period of time, as one can see from Figures 4 and 5.

6 Conclusions and Future Work

In this paper we focus on a particular class of cloud applications that require groups of autonomous servers to work in concert. Such Big Data applications typically have multiple phases and often each phase requires servers with different amounts of cache, memory, secondary storage, CPUs, and attached processors such as GPUs. We assumed the hierarchical cloud organization of today, and most likely that of the future, is the WSCs. We also assume that all servers in a rack are indistinguishable from one another, but possibly different in each rack. The reservation system we propose is designed primarily for the IaaS cloud delivery model, but it could be extended to the PaaS delivery model.

We stressed the need to support *spatial locality* as the communication bandwidth decreases and the communication latency increases due to more intense contention at the higher levels of the network hierarchy. We also wish to allow applications to run uninterrupted until they complete their execution. This *temporal locality* is beneficial to the users, it reduces the response time; it is also beneficial to CSPs as it avoids the overhead associated with saving the application state before checkpointing and restoring the state when restarting it later.

These considerations led us to propose a market-oriented reservation system based on coalition formation and combinatorial auctions. Rack-level coalitions provide the resources required by such applications and combinatorial auctions allow the acquisition of packages of resources; a package consists of several coalitions. Spatial locality is guaranteed by the coalition formation algorithms and temporal locality is guaranteed by combinatorial auctions when services are auctioned for any number of consecutive allocation slots.

We report on a comparative analysis of two strategies for coalition formation: History-based and Just-in-time. They perform equally well in terms of success ratios at both low and high system load. We choose to measure the overhead for the implementation of both strategies with respect to the communication complexity. Our results show that at high system load the History-based strategy performs much better than the Just-in-time one and we attribute this difference to the fact that decisions of the History-based strategy are based on precise knowledge of the internal state of the autonomous servers.

We conducted our experiments on a simple system model. We will replicate our experiments for richer system models, which address some of the concerns discussed above. We will also develop more sophisticated learning algorithms and conduct experiments for longer periods of simulated time to convince ourselves that the system improves over time. As learning is specific to a system and its associated environment, it seems rather improbable that a universally suitable technical definition, that can lead to practical design principles for developing self-organizing systems will emerge soon; however, such a definition would prove to be invaluable.

We are motivated to further explore the principles described in this paper. It is reasonable to postulate the existence of more sophisticated learning algorithms to discover usage patterns at different times of the day, month and year, rather than averaging over past behavior. Moreover, there is no impediment in our work to-date, apart from the desire to simplify the initial investigation, to making coalitions persistent and long-lived. Coalition stability, which could be improved by enhanced information about individual users and applications, would offer the opportunity to further lower communication complexity and this alone merits further investigation.

The results reported in Section 5 cause us to reflect on the question of the feasibility of cloud self-organization. We now examine the current state of affairs and present some arguments for and against cloud self-organization, rather than attempting a definite answer to this question. The results of our experiments are encouraging. They show that the systems can evolve through local decision making only, to an efficient equilibrium, especially when the system is heavily loaded. Moreover, the system seems capable of responding well to external stimuli. But self-organization requires a long history for the formation of stable structures, while the coalition structures, we create here, are short-lived.

The reservation system, presented here, makes reference to only some of the policies of a cloud resource management system. For example, it does not address the question of error recovery nor admission control mechanism, which, in theory, could be extended to enforce a safety margin by keeping a set of resources on standby for use in the case of server failures.

While over-provisioning as implemented in today's clouds is wasteful, it supports elasticity and allows users to scale their resource consumption dynamically. Market-oriented mechanisms, and in particular auctions, require consumers to have a clear idea of the type of resources they are bidding for, and the quantities they need. This is rarely the case in practice; this lack of

precise information, is one of the challenges faced by market-based resource allocation [30].

This problem can be overcome by stipulating that unused resources can be offered on a secondary market, or on "spot" markets, but this solution is not without problems of its own. It complicates the system and encourages the bidders to be greedy. As we can see, simplicity and elegance are affected as we try to optimize resource allocation.

Lastly, reservations appear to contradict the basic tenets of self-organization and autonomic computing which predicate that a system should react to environment changes. No adaptive system can have an instantaneous reaction to a rapidly changing environment. This may be one of the reasons why practical application of self-organization principles has not been successful in the design of large-scale system. The study of self-organization in nature shows that it takes time.

Learning from the past allows a system to create the structures it needs and if the interaction with the environment is an ergodic process, then perhaps self-organization could address a subset of the limitations discussed in Section 1.

Acknowledgments. The authors are grateful for the constructive comments of anonymous reviewers. The research of Dan Marinescu and of John Morrison is partially supported by the NSF CCR grant 1525943 "Is the Simulation of Quantum Many-Body Systems Feasible on the Cloud?" and by the H2020 EU project "CloudLightning," respectively.

References

[1] Amazon Docker. *http://aws.amazon.com/docker*, accessed May 2015.

[2] L. Ausubel, P. Cramton, and P. Milgrom. "The clock-proxy auction: a practical combinatorial auction design." *Chapter 5,* in *Combinatorial Auctions*, P. Cramton, Y. Shoham, and R. Steinberg, Eds. MIT Press, 2006.

[3] L. A. Barossso, J. Clidaras, and U.Hözle. *The Datacenter as a Computer; an Introduction to the Design of Warehouse-Scale Machines.*, Morgan & Claypool, 2013.

[4] M. Blackburn and A. Hawkins. "Unused server survey results analysis." *www.thegreengrid.org/media/WhitePapers/Unused %20Server%20Study _WP_101910_v1. ashx?lang=en* (Accessed on December 6, 2013).

[5] D. Bruneo. "A stochastic model to investigate data center performance and QoS in IAAS cloud computing systems." *IEEE Trans. Parallel & Distributed Systems,* **25**(3):560–569, 2014.

[6] S. Chaisiri, B. Lee, and D. Niyato. "Optimization of resource provisioning cost in cloud computing." *IEEE Trans. Services Comp.,* **5**(2):164–177, 2012.

[7] V. Chang, G. Wills, and D. De Roure. "A review of cloud business models and sustainability." *Proc. 3rd Int. Conf. on Cloud Computing*, pp. 43–50, 2010.

[8] C. Delimitrou and C. Kozyrakis. "Quasar: Resource-efficient and QoS-aware cluster management." *Proc. ASPLOS14,* pp. 127–144, 2014.

[9] Google Docker. *https://cloud.google.com/container-engine,* accessed May 2015.

[10] L. He and T. R. Ioerger. "Forming resource-sharing coalitions: a distributed resource allocation mechanism for self-interested agents in computational grids." *Proc. Symp. Appl. Comp.,* pp. 84–91, 2005.

[11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A.D. Joseph, R. Katz, S. Shenker, and I. Stoica. "Mesos: A platform for fine-grained resource sharing in the data center." *Proc. 8th USENIX Symp. on Networked Systems Design and Implementation*, pp. 295–308 2011.

[12] J. O. Kephart. "Autonomic computing, the first decade." *Int. Conf. on Autonomic Computing, http://www3.cis.fiu.edu/conferences/icac2011/files/Keynote_Kephart.pdf*, 201, accessed May 2015.

[13] S. U. Khan and I. Ahmad. "A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids." *IEEE Trans. Parallel & Distributed Systems,* **20**(3): 346-360, 2009.

[14] Kubernets. *https://cloud.google.com/container-engine/docs/tutorials*, accessed May 2015.

[15] C. Li and K. Sycara. "Algorithm for combinatorial coalition formation and payoff division in an electronic marketplace." *Proc. AAMAS02,* pp. 120–127, 2002.

[16] H. Li, C. Wu, Z. Li, and F. Lau. "Profit-maximizing virtual machine trading in a federation of selfish clouds." *Proc. INFOCOM,* pp. 25–29, 2013.

[17] H C. Lim, S. Babu, J. S. Chase, and S. S. Parekh. "Automated control in cloud computing: challenges and opportunities." *Proc. First Workshop on Automated Control for Datacenters and Clouds,*, ACM Press, pp. 13–18, 2009.

[18] D. C. Marinescu, H. J. Siegel, and J. P. Morrison. "Options and commodity markets for computing resources," In *Market Oriented Grid and Utility Computing,* R. Buyya and K. Bubendorf, Eds., Wiley, ISBN: 9780470287682, pp. 89–120, 2009.

[19] D. C. Marinescu. *Cloud Computing; Theory and Practice.* Morgan Kaufmann, New York, 2013.

[20] D. C. Marinescu, A. Paya, J. P. Morrison, and P. Healy. "Distributed hierarchical control versus an economic model for cloud resource management." *http://arXiv:.org/pdf/1503.01061.pdf*, 2015.

[21] D. C. Marinescu, A. Paya, and J. P. Morrison. "Coalition formation and combinatorial auctions; applications to self-organization and self-management in utility computing." *http://arXiv:.org/pdf/1406.7487.pdf*, 2015.

[22] D. C. Marinescu. *Complex Systems and Clouds: A Self-organization and Self-management Perspective.* Morgan Kaufmann, New York, 2016.

[23] L.Mashayekhy, M.M.Nejad, and D.Grosu. "Cloud federations in the sky: formation game and mechanisms." *IEEE Trans. on Cloud Computing,* 2015.

[24] D.Niyato, A.Vasilakos, and Z.Kun. "Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach." *Proc. Symp. Cluster, Cloud, and Grid Comp.* pp. 215–224, 2011.

[25] S. Penmatsa and A. T. Chronopoulos. "Price-based user-optimal job allocation scheme for grid systems." *Proc IPDPS,* pp. 8-16, April 2006.

[26] T. Rahwan, S. D. Ramchurn, N. R. Jennings, and A. Giovannucci. "An anytime algorithm for optimal coalition structure generation." *Journal of Artificial Intelligence Research,* **34**:521–567, 2009.

[27] S.i D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jenkings. "Coalition formation with spatial and temporal constraints." *Proc. AAMAS 2010)*, pp. 1181-1188 , 2010.

[28] N. Samaan. "A novel economic sharing model in a federation of selfish cloud providers." *IEEE Trans. Parallel & Distributed Systems,* **25**(1):12–21, 2014.

[29] S. Sen and P. S. Dutta. "Searching for optimal coalition structures." *Proc. ICMAS 2000 - 4th Int. Conf on Multiagent Systems,* pp. 287–295, 2000.

[30] J. Shneidman, C. Ng, D. C. Parkes, A. AuYoung, A. C. Snoeren, A. Vahdat, A., and B. Chun. "Why markets could (but don't currently) solve resource allocation problems in systems." *Proc. 10th Conf. on Hot Topics in Operating Systems*, 2005.

[31] M. Sims, C. V. Goldman, and V. Lesser. "Self-organization through bottom-up coalition formation." *Proc. Int. Conf. on Autonomous Agents and Multi Agent Systems,* pp. 867–874, 2003.

[32] B. Snyder. "Server virtualization has stalled, despite the hype." *http://www.infoworld.com/print/146901*, accessed on December 2013.

[33] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. "Omega: flexible, scalable schedulers for large compute clusters." Proc. EuroSys13, pp. 351 – 364, 2013.

[34] A.M. Turing. "The chemical basis of morphogenesis." *Philosophical Transactions of the Royal Society of London,* Series B 237:37–72, 1952.

[35] VMware. "VMware vSphere Storage Appliance." *https://www.vmware.com/files/pdf/techpaper/VM-vSphere-Storage-Appliance-Deep-Dive-WP.pdf*, accessed August 2015.

[36] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J Wilkes. "Large-scale cluster management at Google with Borg." *Proc. EuroSys15,* pp. 124–139, 2015.

[37] S de Vries and R. Vohra. "Combinatorial auctions; a survey." *INFORMS Journal of Computing*, **15**(3):284–309, 2003.

[38] H-J. Zhang, Q-H. Li, and Y-L. Ruan. "Resource co-allocation via agent-based coalition formation in computational grids." *Proc 2 Int. Conf. Machine Learning & Cybernetics,*, pp. 1936–1940, 2003.

[39] G. Wei, A. Vasilakos, Y. Zheng, and N. Xiong. "A game-theoretic method of fair resource allocation for cloud computing services." *The Journal of Supercomputing,* **54**(2):252–269, 2010.

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 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. Prof. 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.