Resource Allocation in Cloud Computing Using the Uncertainty Principle of Game Theory

Parvathy S. Pillai, *Student Member, IEEE*, and Shrisha Rao, *Senior Member, IEEE*

***Abstract*—Virtualization of resources on the cloud offers a scalable means of consuming services beyond the capabilities of small systems. In a cloud that offers infrastructure such as processor, memory, hard disk, etc., a coalition of virtual machines formed by grouping two or more may be needed. Economical management of cloud resources needs allocation strategies with minimum wastage, while configuring services ahead of actual requests. We propose a resource allocation mechanism for machines on the cloud, based on the principles of coalition formation and the uncertainty principle of game theory. We compare the results of applying this mechanism with existing resource allocation methods that have been deployed on the cloud. We also show that this method of resource allocation by coalition-formation of the machines on the cloud leads not only to better resource utilization but also higher request satisfaction.**

***Index Terms*—Cloud computing, coalition formation, game theory, resource allocation, uncertainty principle.**

I. Introduction

INFRASTRUCTURE as a service (IaaS) cloud systems offer computational infrastructure services to multiple clients by means of virtual machines (VMs). The recent surge in the popularity of IaaS cloud systems can be attributed to the on-demand availability of computing resources such as processor cores, memory, disks, etc., packages as VMs that are billed on use [1], [2]. At times, the service requirements are such that not all VMs can be hosted on a single machine. Incoming task requests may demand VMs of different capacities as well. In this scenario, coalitions of machines need to be formed to service a particular request [3]. In particular, in IaaS systems, VMs are available in specific configurations that are further customized by clients by adding operating systems and software on top [2]. Coalitions of preconfigured VMs created before the arrival of requests address scalability concerns to a large extent, in addition to shortening response times.

In this paper, we model the cloud as a multi-agent system that is composed of agents (machines) with varied capabilities. Allocation of resources to perform specific tasks requires agents to form coalitions, as the resource requirements may be beyond the capabilities of any single agent (machine). Coalition formation is modeled as a game and uses the uncertainty principle of game theory [4] to arrive at approximately optimal strategies of the game [5].

Manuscript received July 19, 2013; revised September 11, 2013, January 12, 2014, and March 23, 2014; accepted March 26, 2014.

P. S. Pillai is with the School of Computing, National University of Singapore, Singapore 117417 (e-mail: parvathysp@ieee.org).

S. Rao is with the International Institute of Information Technology Bangalore, Bangalore-560100, India (e-mail: shrao@ieee.org).

Digital Object Identifier 10.1109/JSYST.2014.2314861

Optimizing resource allocation to ensure the best performance can be done in many ways. Present IaaS service providers, largely unaware of application-level requirements, do not provide any optimization by configuring the required software on the VMs. Relying only on application-level optimization is not sensible, as such is restricted to an existing infrastructure allocation. The placement of VMs is, however, in the hands of the IaaS provider and can be changed based on the topology of the machines in the cloud system. Application-level optimization techniques [6]—along with topology-based VM placement—offers better chances of performance improvement with lower resource wastage.

The cloud may be built up of tens of thousands (or even more) of physical machines, and considering each one as a possible host for a requested VM rapidly grows out of proportion as requests keep coming. Cloud providers today have stuck to offering services of VMs of specific sizes. For instance, Microsoft's Azure [7] provides VMs of configurations as shown in Table I. Hence, the cloud providers' current situation is that they know the type of VMs that may be requested but are unaware of the exact request specifications such as the number of instances of a particular type of VM. The other fact to be highlighted is that VMs are not capable of getting configured on the go. They have to be up and running on the host machines.

Game-theoretic approaches in general help simplify complex problems to a great extent. Former game-theoretic approaches to similar problems have needed the use of integer programming for solving the payoff matrix for optimization. Integer programming has a high complexity and goes out of bounds if the number of machines and requests increase even by a modest amount [8]. Our approach, however, has been to find only the machine combinations by solving the simpler equation of Theorem 1 and offers an alternative to integer programming.

Our resource allocation mechanism for the cloud prioritizes different aspects.

1) We wish to take advantage of any knowledge of the types of VMs that may be requested. This foresight of the demand allows us to form coalitions of machines to host VMs even without the actual request being yet available.
2) In data-intensive applications such as MapReduce [9] or even web search [10], the placement of disks that hold data can affect performance to a great extent. Taking this into account, coalitions of machines that are closer in proximity (facilitating lowered data movement) are given higher payoffs.
3) The exact task information is uncertain until the arrival of the actual request.

TABLE I
MICROSOFT AZURE VM CONFIGURATIONS [7]

Type	CPU (GHz)	Memory (GB)	Storage (TB)
Small	1 × 1.6	1.75	0.22
Medium	2 × 1.6	3.5	0.48
Large	4 × 1.6	7	0.98
Extra Large	8 × 1.6	14	1.99

Thus, we implement a resource allocation mechanism for the cloud that is demand-aware, topology-aware and uses a game-theoretic approach based on coalition formation of machines for requests with uncertain task information. With these ideas in place, we can use our agent-based resource allocation mechanism for the IaaS cloud.

The evaluation of the efficacy of our approach is carried out by comparison with common commercial allocation strategies on the cloud. We evaluate it based on randomly generated VM requests that include data-intensive requests. These workloads are chosen because they are very generic and do not solely apply to a specific application domain. Evaluations consider the time for task allocation, resource wastage, and request satisfaction. The results are, as we shall discuss later, quite encouraging.

The rest of this paper is as follows. The related work is touched upon in Section II. Section III briefly introduces the uncertainty principle of game theory and coalition formation. Our system model and proposed solution are presented in Section IV. Section V presents the algorithms, and Section VI explains the approach taken for experimentation and discusses the results. Section VII concludes this paper.

II. RELATED WORK

A. Comparison With Earlier Work

In earlier work [5], we developed an agent-based resource allocation mechanism under uncertain task specification, using the principle of coalition formation from game theory. Servers in server farms were modeled as agents differing in their resource capabilities based on their compute, memory, and storage capacities; the platforms they support; their software configurations; and so on. Each request from clients may require some multitude of these resources to be utilized. Servers form coalitions prior to the arrival of task requests from clients. We assumed that the server agents know a possible set of tasks that may be allotted to them, but not the exact one. Each server agent chooses a coalition (strategy) from a set of feasible coalitions (strategies). Prioritizing coalitions by agents happens by applying the uncertainty principle of game theory for zero-sum games. Such pre-computation is an important benefit as it avoids the delays inherent in reacting to requests after they have been made. The proposed model is for nonoverlapping coalitions and considers the priority of the task involved in calculating the payoff associated with the game instances.

As compared to our previous work, the present modified approach is for the cloud rather than server farms. The host machines are capable of being part of multiple coalitions, provided that their participation does not exceed the maximum number specified for each machine. The coalitions are also not limited to having a pre-defined number of members.

B. Literature

Resource allocation [11] has been widely researched with respect to operating systems, grid computing, server farms, and cloud computing management. The aim of allocation mechanisms is to guarantee a mapping to satisfy client requirements with the provider's infrastructure. In this regard, many resource allocation strategies have been proposed, which focus on different aspects of performance. Some considerations taken into account while devising a resource allocation strategy are the current workload of the cloud, minimum response time, maximum satisfaction of requests, and minimum resource wastage [11]. Hence, we can say that resource allocation is an optimization problem [12].

Demand-based resource allocation assigns resources on the cloud dynamically based on the current workload. Such algorithms also help improve energy efficiency. Shanmuganathan *et al.* [13] proposed two algorithms, which are called Distributed Binary Search and Base + Proportional Excess, that dynamically allocate the overall capacity among VMs based on their demand, user importance, reservation, and limit settings. Demand forecasting techniques are used by Cao *et al.* [14] to implement a power-saving approach to resource allocation, which itself is modeled as a modified knapsack problem. Another on-demand strategy proposed by Chen *et al.* [15] can satisfy user requirements in limited time, with guaranteed lowest request refusal rates. This algorithm uses resource reconfiguration as a strategy to satisfy user requirements.

Game-theoretic approaches to resource allocation in distributed systems are well known. A market-like environment is used in auction-based decentralized resource scheduling [16] to mitigate the complexities involved in message passing, reaching a closure, and allocating the final schedule, which are issues associated with distributed systems in general. Raman *et al.* [17] modeled resource management as an instance of matchmaking. Game-theoretic resource allocation on the cloud as proposed by Wei *et al.* [18] follows a two-step mechanism. In the first step, each participant solves its optimization problem independently, without considering the multiplexing of resource assignments. In the second, an evolutionary mechanism changing the multiplexed strategies of the initial optimal solutions (minimizing their efficiency losses) is designed. Prasad and Rao [19] used auction-based mechanisms for resource procurements by cloud brokers.

Different methods for task allocation via agent coalition formation are discussed by Shehory and Kraus [20], whose algorithm consists of two main steps involving calculation of possible coalitions and coalition formation, in scenarios of nonoverlapping and overlapping coalitions. Automated multi-party negotiation and task-oriented resource-sharing coalitions for computational grids are studied by He and Ioerger [21]. Agent-based resource allocation in grid computing [22] and resource and revenue sharing on the cloud using coalition formation [23] are also known. Niyato *et al.* [23] considered

the multiple cloud providers as agents who form the coalition. These approaches to coalition formation are very useful when clear task specifications exist, but less so when they do not. Combining coalition formation and game theory, Hassan *et al.* [24] proposed a solution for dynamic resource allocation in a cloud federation. Cloud providers are defined with price functions that give incentives to other clouds to contribute resources and to form a federation. Coalition-based games are effective in fields that need cooperation from different agents. Saad *et al.* [25] discussed coalition formation in wireless networks to maximize utilities of network partitioning of antennas. Mashayekhy and Grosu [26] proposed dynamic virtual organization formation on the grid based on coalitional game theory. In later work, Mashayekhy and Grosu [12] designed a mechanism that enables cloud providers to dynamically form a cloud federation to maximize their profit.

Communication-aware job placement has been worked on in the grid computing community. Such work mostly deals with the overheads of WAN communication between grid sites [27]. For data-intensive applications such as MapReduce, Lee *et al.* [28] proposed a topology-aware resource allocation mechanism for IaaS-based cloud systems, using a prediction engine and genetic algorithm to find an optimal solution in a large search space.

An approach to coalition formation with uncertain heterogeneous information [29] was used for Request For Proposal management, where each task is divided into multiple subtasks. The heuristic ranks the possible coalitions and follows it with a negotiation step for establishing coalitions. An auction-based approach for coalition formation under uncertainty is described by Hosam and Khaldoun [30] and uses a Markov decision process (MDP) for formalizing subtask selection.

C. Notes

Tasks, resources, and their relationships are involved in the decision of their execution schedule on any system. Subtask dependencies and their turnaround times are important considerations in a scheduling problem. Our work follows the resource allocation problem in particular, as compared to the on-demand method for scheduling. Wei *et al.* [18] applied game theory in an evolutionary algorithm to improve upon the initial independent optimal allocation formed by solving binary integer programs. We use game theory for ordering *coalitions* of host machines based on the uncertainty principle of game theory.

The on-demand strategies for resources allocation proposed by Shanmuganathan *et al.* and Chen *et al.* [13], [15] take into account the workload at the time of task arrival. Our work differs from these approaches and the forecasting techniques used by Cao *et al.* [14], as we rely on pre-computed open coalitions that are demand-aware. The coalition formation approach for multi-agent systems proposed by Shehory and Kraus [20] has high communication complexity. As we check for coalition feasibility based on a preference list, the communication costs are considerably reduced. Our pre-computed coalitions involve host machines—as compared to coalition formation among self-interested agents that are either resource users or companies owning resources on the grid [21], [22]. Coalitions among cloud providers [23], [24] have been called cloud federations. The dynamics of these coalitions are different, because the agents are inherently self-interested due to different ownerships, and make their own decisions according to their budgets, capabilities, goals, etc. Optimal coalition formation under uncertain resource consumption by tasks is planned by an MDP by Hosam and Khaldoun [30]. It is known that exact solutions are intractable for large MDPs [31]. Our concern about uncertainty is which VMs will be required at what time, i.e., which task arrives when. Kraus *et al.* [29] developed a protocol and employed a heuristic-based approach for task domains, where each task should necessarily be present in a different agent. The uncertainty there is about the capabilities of other agents. In our case, since we consider machine coalitions for specific VM configurations on the cloud, not necessarily across providers, the machines know of the others' capabilities.

It may be noted that the problem of resource *allocation* in cloud computing that we address in this paper is very different from that of resource *procurement* [19]. The problem of allocation has to do with the cloud vendor or provider choosing what resources to devote to which purposes in anticipation of possible demands from cloud users, whereas the problem of procurement has to do with the cloud users (or brokers acting on their behalf) choosing what cloud resources to obtain from which cloud vendors for the best possible match to the users' needs.

III. Background

A. Coalition Formation

As previously mentioned, coalition formation by agents occurs in systems with multiple agents which must cooperate to get a multitude of tasks done. Agents cooperate in a coalition, as a single agent may not have all the capabilities needed to execute a task. A typical multi-agent system can be formulated with the following specifications (see Table II for notation).

1) *Agents*: There is a set of n agents, $\mathbb{H} = \{H_1, \ldots, H_n\}$. Each agent H_i has a *capabilities vector*, $B_{H_i} = \{b_1^i, \ldots, b_k^i\}$. The capabilities vector for an agent measures its resource capacity.
2) *Tasks*: There is a set of m tasks, $\mathbb{G} = \{G_1, \ldots, G_m\}$. Each task G_j has a *necessities vector*, $N_{G_j} = \{d_1^j, \ldots, d_k^j\}$. The necessities vector for a task specifies its minimum resource requirements.
3) *Payoffs*
   a) Task payoff: Each task G_j is associated with a payoff, P_{G_j}, based on the resources it requires. A task requiring a higher necessity set has a higher payoff. Thus
   $$P_{G_j} \propto N_{G_j}. \tag{1}$$
   b) Agent payoff: Each agent H_i receives a payoff for being a part of coalition C_l that services task G_j. The agents are rewarded payoffs based on their contributions to any task assigned to the coalition they are part of
   $$\mathbb{P}(H_i, C_l, G_j) \propto \langle B_{H_i}, P_{G_j} \rangle. \tag{2}$$

TABLE II
NOTATION

Symbol	Description
$\mathbb{H}$	set of agents $H_1, \ldots, H_n$
$\mathbb{G}$	set of tasks $G_1, \ldots, G_m$
B_{H_i}	capability vector for agent H_i; elements of H_i are non-negative reals
N_{G_j}	necessity vector for task G_j; elements of G_j are non-negative reals
P_{G_j}	payoff for task G_j
$\mathbb{P}(H_i, C_l, G_j)$	agent H_i's payoff for task G_j in coalition C_l
$E(C_l, H_i)$	expected payoff for an agent H_i coalition C_l
D_{C_l}	capability vector of coalition C_l; elements of C_l are non-negative reals
$\mathbf{A}$	payoff matrix with $a_{l,j}$
$\mathcal{R}$	request from the client
HM_p	p^{th} Host machine
VM_q	q^{th} Virtual machine

c) Expected payoff: Each H_i calculates its expected payoff $E(C_l, H_i)$ for being part of coalition C_l. This is done to form a preference list of the coalitions that it can be part of.

4) *Preference list computation*: A preference list is created by solving instances of two-player zero-sum games wherein each agent is a player and the central authority that allocates the tasks, which we call the *task allocator* [8], is the other player. The game specifications are as follows.
   a) Strategies for the agent: the different possible coalitions that it can be part of.
   b) Strategies for the task allocator: the different tasks.
   c) Payoff matrix, $\mathbf{A} = (a_{l,j})$: The rows correspond to the strategies (coalitions) of the agent, and the columns correspond to the strategies (tasks) of the task allocator; $(a_{l,j})$ is the payoff that the agent receives from the task allocator, if the agent chooses coalition C_l and the authority chooses task G_j.
   d) Objective: The game progresses as the agent tries to maximize the minimum payoff it can obtain from the task allocator, and the allocator tries to minimize the maximum payoff it has to pay to the agent.

B. The Uncertainty Principle of Game Theory

The game-theoretic analog of Heisenberg's well-known uncertainty principle [32] is a lower bound on the entropy of optimal strategies of zero-sum games [4]. In zero-sum games, the lower bound for randomness of optimal solutions is given in terms of the value δ of the commutator of two nonlinear operators, namely, maximum and minimum, on the payoff matrix. This is *the uncertainty principle of game theory*. The commutator gives the extent to which the $\min$ and $\max$ operators can be commutative. This lower bound is determined by a single parameter δ of the game, as given by

$$\delta = \left(\min_z \max_w - \max_w \min_z\right) a_{w,z}. \tag{3}$$

Theorem 1 (Székely and Rizzo [4]): If $G(\delta)$ denotes the class of two-player, finite, zero-sum games with commutator coefficient δ, and $h(\delta)$ is the entropy of the two-point distribution $(1/(1+\delta), \delta/(1+\delta))$, then a lower bound for entropies of optimal solutions (x^*, y^*) of games in $G(\delta)$ is given by

$$\min\left(H(x^*), H(y^*)\right) \geq h(\delta). \tag{4}$$

Moreover, $h(\delta)$ is the greatest lower bound. □

When $\delta > 0$, the players realize that a mixed strategy is optimal but cannot be sure of the optimal mixed strategy chosen by the other player. Hence, each player tries to find a probability distribution that approximates the opponent's actions. The entropy is small when the probability mass is very concentrated on a few strategies. Among mixed strategies to be analyzed, the easiest ones are those of a two-point distribution. Minimum entropy corresponds to the entropy $h(\delta)$ of the two-point distribution $(1/(1+\delta), \delta/(1+\delta))$. This means that the optimal mixed strategy achieves the minimum entropy when the strategy is supported on exactly two points with probabilities $(1/(1+\delta), \delta/(1+\delta))$ [4].

C. Application of the Uncertainty Principle of Game Theory to Coalition Formation

We find that the uncertainty principle of game theory can be applied to find the good strategies reasonably quickly [5], [8]. The game is formulated to be played between each agent and the allocator. It is not a single game but multiple games (one instance of the game is between an agent and the allocator). Each agent–allocator combination can form their payoff matrix, as it is assumed that they know the calculation of expected payoff for a coalition that serves a particular task. Hence, each agent (assumed to be selfish) tries to make the allocator pay and forms its own list of coalitions. A coalition becomes feasible only if all its members are interested. The coalition list computation is essentially decentralized and happens at each agent, and the coalition is finally formulated after the feasibility check.

When δ is equal to 0, a saddle point exists, and the optimal strategy is a pure strategy. In such games, the solution may be easily found by repeated application of iterative dominance. When δ is greater than 0, a mixed strategy is optimal. Since the optimal strategy of the other player is unknown, the player has to opt for a probability distribution over the pure strategies (i.e., it must choose a mixed strategy). In general, computation of optimal strategies is expensive when the number of strategies available to a player is large and the computation is time-bound. In such situations, an alternative solution strategy is proposed [4]. The solution obtained by that mechanism is optimal, in the sense that the expected payoff is guaranteed to be above a certain lower bound (refer Theorem 1).

At each instance of the game, when $\delta > 0$, the two coalitions with probabilities $(1/(1+\delta), \delta/(1+\delta))$ are removed from the list of possible coalitions for the agent and are added to its preference list. In the next iteration, the game is played with the remaining coalitions, and the next two best coalitions are found. This process is repeated until all the possible coalitions are listed out in the order of preference.

The purpose of applying Theorem 1 for forming a preference list, rather than a near-optimal mixed strategy, is that there are multiple coalitions computable for an agent. However, a

computed coalition becomes feasible only by the informed consent of all the constituent agents. Hence, the near-optimal coalition according to an agent may not be favored by another member. By creating a preference list, the host machine can now check for the feasibility of the next coalition in the list, rather than doing the full computation yet again.

By iteratively applying this principle over the payoff matrix after removing the strategy solutions from the previous iteration, we order the near-optimal strategies. This is similar to the ordering in the Gale–Shapley algorithm for the stable marriage problem [33]. We employ it for the ranking of coalitions for an agent. In view of the non-determinism of the types of tasks to be served and the machines available, it is safe to assume (without loss of generality) that saddle points rarely occur with random payoff matrices [4]. It has been also noted that the application of the uncertainty principle has resulted in errors as low as 0.1 in 98% of the games which involved random matrices [4].

We summarize the discussion and notation based on the above as follows [5].

1) An agent H_i prefers a coalition C_e over C_f if $E(C_e, H_i) > E(C_f, H_i)$.
2) A coalition C_l is said to be the best coalition for an agent if, $\forall C_w$, $E(C_l, H_i) > E(C_w, H_i)$, $l \neq w$, where C_w is a possible coalition for H_i excluding C_l.
3) An agent H_i is said to be part of a disinterested coalition C_e if $E(C_f, H_i) > E(C_e, H_i)$, where C_f is another possible coalition.
4) The system is stable when there are no disinterested coalitions.
5) The capability of a coalition C_l is the sum total of the capabilities of the agents that are part of the coalition, i.e., $D_{C_l} = \sum_{\forall H_i \in C_l} B_{H_i}$.
6) If a coalition C_l is able to satisfy the necessities of task G_j, we denote this by $N_{G_j} \sqsubseteq D_{C_l}$ (see Algorithm 3).

IV. System Model

We make the following assumptions as part of the model [5].

1) The original capabilities of the machines are known.
2) A coalition is feasible only if all the machines involved in it agree to cooperate.
3) Machines can calculate the expected payoffs correctly.
4) Machines remain in the coalition until their allotted task is completed.
5) The possible VMs that need to be hosted are known beforehand. These VMs should be up and running on the host machines. The exact task information is available only at the time of client request.

A. The Knowledge Base

The knowledge base at the cloud is like a data bank, which is accessible to all the machines in the cloud and the task allocator. The functionality of the knowledge base can be enumerated as follows.

1) Maintaining a list of possible VMs: We assume that the cloud has knowledge of the type of VM requests that it has to handle. Each VM_q is a tuple with configuration specifications as

$$VM_q = \langle \text{size\_}VM_q, \text{core\_}VM_q, \text{memory\_}VM_q, \text{storage\_}VM_q, \text{host-os\_}VM_q \rangle.$$

The specifications include the size of VM_q (size_VM_q), the number of cores needed for VM_q (core_VM_q), the amount of memory needed (memory_VM_q), the disk space needed (storage_VM_q), and the host operating system (OS) needed on the physical machine running VM_q. The values taken by these parameters are specified in Table I.

2) Listing capabilities of host machines: Each machine HM_p on the cloud has a set of capabilities, i.e.,

$$HM_p = \langle \text{ID}, \text{rack\_id}, \text{core\_}HM_p, \text{memory\_}HM_p, \text{storage\_}HM_p, \text{host-os\_}HM_p. \rangle$$

The specifications include the rack (rack_id) in which HM_p is placed, the number of cores (core_HM_p), the amount of memory (memory_HM_p), the disk space (storage_HM_p), and the host OS running.

This knowledge base forces an information symmetry across the host machines and the allocator. Information transfer (or status updates) from the host machines to the allocator is required. However, large data center and cloud systems already have significant monitoring tools that provide near-real-time updates of various systems to their controllers [34].

B. Resource Allocation Through Coalition Formation

We assume that s VM configurations are available with the cloud service provider. A typical IaaS client request for VMs with Microsoft Azure [7] has the following form:

$$\mathcal{R} = \langle \text{no\_}VM_1, \ldots, \text{no\_}VM_s \rangle$$

where the number of each VM instance needed by the particular request is specified.

The payoff associated with a VM request $\mathcal{R}$ is calculated as $P(\mathcal{R})$ and is a function of the number of cores core_VM_q and the disk space storage_VM_q since the payoff for VMs scales with the resources needed. We denote this by

$$P(\mathcal{R}) \propto \langle \text{core\_}VM_q, \text{storage\_}VM_q \rangle. \quad (5)$$

The payoff rewarded to a host machine HM_p for servicing a request $\mathcal{R}$ as part of a coalition C_l is, in turn, a function of both the payoff for the particular request and the machine's contributions, i.e.,

$$\mathbb{P}(HM_p, C_l, \mathcal{R}) \propto \langle HM_p, P(\mathcal{R}) \rangle. \quad (6)$$

The mapping between coalition formation and resource allocation on the cloud is depicted in Table III. The two players involved in the game formulation are the *task allocator* and the *host machine*. Each host machine chooses a coalition (strategy) from a set of feasible coalitions (strategies). Each coalition has a different payoff associated with it. The set of computed

TABLE III
COALITION FORMATION IN RESOURCE ALLOCATION

Two-player zero-sum game	Cloud resource allocation problem
Player 1	Task Allocator
Player 2	Host Machine(HM_p)
Strategies for Player 1	Task requests
Strategies for Player 2	Coalitions
payoff for Player 1	Task Completion
payoff for Player 2	Preferred Coalition Formation

TABLE IV
SAMPLE PAYOFF MATRIX

Tasks / Coalitions	G_1	G_2	G_3
C_1	M_{11}	M_{12}	M_{13}
C_2	M_{21}	M_{22}	M_{23}
C_3	M_{31}	M_{32}	M_{33}

coalitions for each host machine has all coalitions that it may be part of. The payoff is calculated by considering the task allocated to the coalition. The opponent player or the task allocator has a strategy set consisting of the set of all possible VM configurations that may be requested. IaaS systems follow a utility-based pricing of resources [2], [7]—the higher the quantity and quality of resources, the greater the price billed from the user. Hence, the cloud provider's and thereby the allocator's choice would be to employ a suitable coalition with the largest capability. The payoff calculation depends on the capabilities of the coalition and the task, which means that the coalition that is able to satisfy a task with the least wastage of resources is entitled to a higher payoff than a coalition that has higher capabilities than that demanded by the task. In effect, this is a win-win situation wherein the allocator plays a min–max game and the host machine plays a max–min game. The host machines prefer to be in their favored coalitions and to serve requests that give maximum payoffs. The allocator tries to forcibly allocate tasks to machines irrespective of their preferred coalitions. Thereby, it is a two-player zero-sum game, as the gain received by a host machine is equal to the payoff given by the allocator. In order to bring in optimization, the request has to be serviced with the minimum wastage of resources. This is done by making the host machines prioritize their coalitions. The solution of the two-player zero-sum game is the near-optimal coalition for the host machine.

As depicted in the sample payoff matrix (see Table IV), each entry corresponds to the payoff to be given by the task allocator. $M_{l,j}$ represents the payoff by the task allocator if the coalition chosen by the host machine is C_l and the task performed by the coalition is G_j. The host machines try to maximize their minimum payoffs, whereas the allocator attempts the inverse.

Solving for optimal coalitions may be done by integer programming. The computational infeasibility of this is on account of the fact that, as the number of strategies (both coalitions for the host machine and tasks for the allocator) increases, finding optimal coalitions gets out of bounds [4]. It can be inferred that, as host machines are allowed to form coalitions without any prior constraints on the size of or participants in a coalition, the search space for each agent exponentially grows, the possible number of coalitions being 2^n for n agents. The communication overhead and computation costs in a coalition also increase with its size. Unless a heuristic is applied to adjust constraints and hence simplify the problem, solving the game for finding the optimal strategy seems infeasible. Hence, it is justifiable that, although our approach gives near-optimal strategies, it is of lower complexity and does not involve heuristics.

TABLE V
EXAMPLE: PAYOFF MATRIX

	G_1	G_2	G_3
$C_{(I,II)}$	$\frac{-1}{3}$	$\frac{-2}{3}$	-1
$C_{(I,III)}$	$\frac{5}{4}$	$\frac{-1}{4}$	$\frac{-1}{2}$
$C_{(I,IV)}$	$\frac{4}{5}$	$\frac{6}{5}$	$\frac{-1}{5}$
$C_{(I,V)}$	$\frac{3}{6}$	$\frac{5}{6}$	$\frac{7}{6}$

Example: Payoff Matrix Calculation: Consider a setting with five machines, namely, HM_I, HM_{II}, HM_{III}, HM_{IV}, and HM_V, with $B_{\mathrm{HM}_I} = 1$, $B_{\mathrm{HM}_{II}} = 2$, $B_{\mathrm{HM}_{III}} = 3$, $B_{\mathrm{HM}_{IV}} = 4$, and $B_{\mathrm{HM}_V} = 5$ cores, respectively. Let the tasks that these machines have to accomplish require $N_{G_1} = 4$, $N_{G_2} = 5$, and $N_{G_2} = 6$ core VMs. Originally, B_{HM_p} and N_{G_j} are vectors; however, for the sake of simplicity of illustration, we consider them as ordinal numbers indicating the number of cores. The candidate coalitions for host machine HM_I are $C_{(\mathrm{HM}_I,\mathrm{HM}_{II})}$, $C_{(\mathrm{HM}_I,\mathrm{H}_{III})}$, $C_{(\mathrm{HM}_I,\mathrm{HM}_{IV})}$, and $C_{(\mathrm{HM}_I,\mathrm{HM}_V)}$ with capabilities $D_{C_{(\mathrm{HM}_I,\mathrm{HM}_{II})}} = 3$, $D_{C_{(\mathrm{HM}_I,\mathrm{HM}_{III})}} = 4$, $D_{C_{(\mathrm{HM}_I,\mathrm{HM}_{IV})}} = 5$, and $D_{C_{(\mathrm{HM}_I,\mathrm{HM}_V)}} = 6$. The payoff to HM_I for being a member of the candidate coalitions is calculated as

$$\begin{cases} (D_{C_l} - N_{G_j}) \times \frac{B_{\mathrm{HM}_p}}{D_{C_l}} & \text{if } D_{C_l} < N_{G_j} \\ c \times \frac{B_{\mathrm{HM}_p}}{D_{C_l}} & \text{if } D_{C_l} = N_{G_j} \\ \left[c - (D_{C_l} - N_{G_j})\right] \times \frac{B_{\mathrm{HM}_p}}{D_{C_l}} & \text{if } D_{C_l} > N_{G_j} \end{cases}$$

where D_{C_l} and N_{G_j} denote coalition C_l's resource capability and task G_j's resource necessity, respectively. B_{HM_p} represents the capability of a host machine HM_p. The resultant payoff matrix when the scaling factor c is taken as $N_{G_j} + 1$ is depicted in Table V. δ is computed from (3) as $(4/6) > 0$. Hence, the near-optimal two-point strategy is supported by $((8/25), (17/25))$ as given by Theorem 1. Reducing the payoff matrix to the form described in [4], the strategies with minimum entropy and nearest to the calculated values are $C_{(\mathrm{HM}_I,\mathrm{HM}_{IV})}$ for G_1 and $C_{(\mathrm{HM}_I,\mathrm{HM}_V)}$ for G_2.

C. Optimization Problem

The resource allocation strategy for the cloud aims to optimize the usage of the different resources. It is an optimization problem with the following:

- Job completion time: Allocating resources to *minimize* the total execution time results in good utilization of resources. It maps to the monetary cost of executing the task on an IaaS system [28]. Minimum execution time means lower payment by the client for engaging the VM. It also means that more VM requests can be serviced.

- Constraints
  1) Task characteristics: We assume that approximate execution times of the different VM request types are available beforehand along with their resource requirements.
  2) Resource availability: The number of simultaneous VMs on a particular machine that may be allocated to tasks is known, as is the resource capability of each machine. During task allocation, open coalitions that are not already servicing other tasks should be preferred to ensure execution time reduction.
  3) Request satisfaction: Allocating resources to increase the number of client requests satisfied results in not only efficient resource division among the current requests, but also reduced number of idle host machines. This results in lowered resource wastage and better profit to the IaaS provider, as the number of client requests that can be billed increases.
  4) Topology: It is obvious that VMs within the same host machine need the least communication time, followed by those on host machines in the same rack, and then by machines in different racks. If the topology of the cloud is taken into consideration while forming coalitions, higher payoffs should be given to coalitions involving machines with lower communication costs, i.e., coalitions with the least internal communication costs should be preferred for task allocation.

D. System Mechanism

Host machines are categorized into those that are i) part of a coalition; ii) ready to be in a coalition; and iii) unwilling to be in a coalition.

The resource allocation mechanism in the cloud is pictorially represented in Fig. 1.

The task allocator and the host machines have access to the knowledge base, which has the exhaustive list of the VM types that may be requested and the host machine configurations. This information is used in computing the expected payoff matrix. The calculation of the expected payoff assumes a common user priority for all tasks, as this precedes the arrival of the actual requests. The tasks, however, have different overall payoffs, as the resources they require are different. By solving their respective zero-sum games, host machines arrange themselves in open coalitions. Open coalitions are those coalitions that are not assigned any user request as yet, but are ready for service.

The arrival pattern of tasks is non-deterministic, and that is essentially the uncertainty involved. While pre-computing open coalitions, tasks are not considered to be held in a reserve. In commercial cloud platforms (e.g., Microsoft Azure that we have considered), the configurations of VMs provided are already available even before the exact arrival of the task. A task requests a certain number of VMs of available configurations. It is against the already known VM configurations that open coalitions are pre-computed. As and when the task arrives, the request per VM can be satisfied by an open coalition.

The formation of open coalitions has two major steps [5].

1) *Calculation of coalition preference*: If t is the number of host machines that are ready, there are $\binom{t-1}{r-1}$ possibilities for a host machine to choose its partners in a coalition of size r. This is followed by the calculation of the expected payoff for each potential coalition, for each of the probable tasks involving specific VMs. The payoff calculated from the preceding step is used in building a preference list, which specifies the order of the coalitions favored by the host machine. The preference list is populated by finding the two high-payoff coalitions in each iteration of the two-player zero-sum game between the host machine and the task allocator using the uncertainty principle of game theory.

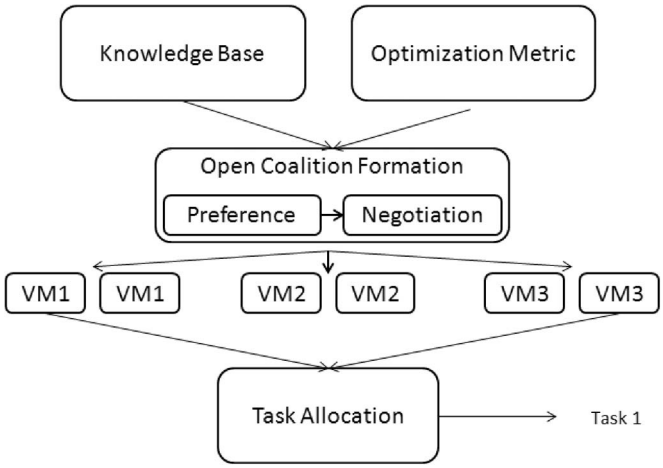

Fig. 1. Resource allocation.

2) *Negotiation step*: The host machines use their preference lists to check for the feasibility of coalitions in that order. A coalition C_l is feasible if the host machines forming that coalition agree to be part of it. If a host machine HM_p's preferred coalition is not agreeable to the other machines in it, HM_p takes up its next preferred coalition from its preference list for feasibility check. On passing a feasibility check, C_l becomes an open coalition, with HM_p and the other machines for its members. Once a machine reaches the maximum number of open coalitions it can be part of, it is removed from the list of ready machines. The other machines in the ready list remove from their preference lists those coalitions involving the machines that have reached their maximum number of concurrent open coalitions. Once an open coalition is assigned a task, the host machines remain part of the coalition until the task is complete. Coalitions are of fixed sizes once formed. If, on the arrival of a request none of the existing open coalitions have the capabilities for executing the task involved, the allocator assigns a host machine or a group of host machines ready to be a part of a coalition to serve the request. The allocator then has to ensure that the necessities of a task are met while allocating forcibly to machines or a group of machines. After the completion of a task, the coalition is dissolved, and the machines in that coalition update their concurrent coalitions list. If they were previously at their maximum capability, they become available after the coalition dissolution.

TABLE VI
DATA STRUCTURES IN ALGORITHMS

Structure	Data Members	Description
HostMachine (HM_p)	ID	uniquely identifies the host machine
	rack-id	the rack in which the machine p placed on the cloud
	status	takes values available, open_coalition, max_coalition, unavailable
	potential_coalitions_list	list of the coalitions the server agent is capable being part of
	cur_no_coalitions	number of coalitions that the machine is currently part of
	max_no_coalitions	maximum number of simultaneous coalitions that an agent can be part of
	capability_vector (B_{HM_p})	resource specifications(core, memory, storage, host-OS) for the machine
	preference_list	potential coalitions in decreasing preference order
	payoff_matrix A	payoff associated with each task for being a part of some coalition
Coalition (C_l)	ID	uniquely identifies the coalition
	status	takes values OPEN: open coalition, ENGAGED: servicing a request
	members_list	host machines that are part of that coalition
	task_id	identifies the task allocated to the coalition
	capability_vector	collective capabilities of the members in the coalition
VM	size	specifies the size of the virtual machine
	necessity_vector	resource specifications(core, memory, storage, host-OS) for the VM
Task (G_j)	ID	uniquely identifies the task
	status	denotes whether it is completed or not
	coalition_id	identifies the coalition which is servicing the task
	VM_list ($\mathcal{R}$)	number of each type of VMs needed for satisfying the task
	necessity_vector	collective necessities of VM_list ($\sum_{q=0}^{s}$VM_list[q].necessity_vector)

V. ALGORITHMS

We modify and use the algorithms (Algorithms 1, 2, and 3) from our earlier work [5] to perform resource allocation on the cloud. The algorithms are for open coalition formation, coalition dissolution, and task allocation (see Table VI for the data structures of the algorithms). The proofs of correctness can be found in our previous paper [5].

1) Open Coalition Formation Algorithm: The Open Coalition Formation Algorithm (Algorithm 1) can be broken down into two major steps. The first step is the calculation of coalition preference. This step includes the evaluation of all the feasible coalitions for a host machine and creating a preference list on them. The complexity lies in the fact that each of the host machines might have different preferences based on the expected payoffs, and hence, disinterested coalitions may occur. This is avoided by the second step called the negotiation step, wherein the machines exchange mutual information with regard to the feasibility of the coalition. With this, the formation of only stable coalitions is ensured. If a host becomes part of multiple coalitions, it is ensured that the one offering lesser payoff is formed only after the one that offers a higher payoff is already formed. From our previous paper [5], the Open Coalition Formation Algorithm has been modified to allow host machines to participate in more than a single coalition.

Algorithm 1 first computes the set of machines available to be part of coalition, in lines 2–4. In line 6, a particular host machine's set of feasible coalitions is populated. Line 7 computes the payoff the host machine receives for a particular coalition, and based on this, the machine's preference list of coalitions is populated in line 8. Line 15 is the negotiation step between the machines where the feasibility of the coalition is checked. It is checked whether it is possible to be part of a best possible coalition. Lines 17–22 form the coalition. If the number of coalitions that an agent is part of reaches the maximum capacity, lines 23–26 remove that machine from the list of available hosts. The preference list for the available host machines is updated in lines 27 and 28 by removing the coalitions involving unavailable hosts.

Algorithm 1: Open Coalition Formation Algorithm

```
   Data: host_machine_available
   Data: host_machine_total
   Result: coalitions_set
1  begin openCoalitionFormationAlgorithm
2      foreach host_machine in host_machine_total do
3          if host_machine.status=available then
4              add(host_machine_available, host_machine)
5      foreach host_machine in host_machine_available do
6          host_machine.potential_coalitions_list ←
           ComputeCoalitions(host_machine)
7          host_machine.payoff_matrix ←
           ComputeMatrix(host_machine.potential_coalitions_list)
8          host_machine.preference_list ←
           ComputePriorities(host_machine.payoff_matrix)
9      while host_machine_available ≠ NULL do
10         Count ← 0
11         foreach host_machine in host_machine_available do
12             Count ← Count+1
13             BestPossible ← host_machine.preference_list[1:Count]
14             foreach coalition in BestPossible do
15                 if isfeasible(coalition) then
16                     if host_machine.cur_no_coalitions <
                       host_machine.max_no_coalitions then
17                         JoinCoalition(coalition, host_machine)
18                         coalition.status ← OPEN
19                         if host_machine.status=available then
20                             statuschange(host_machine,
                               open_coalition)
21                         host_machine.cur_no_coalitions ←
                           host_machine.cur_no_coalitions + 1
22                         add(coalitions_set,coalition)
23         foreach host_machine in host_machine_available do
24             if host_machine.cur_no_coalitions =
               host_machine.max_no_coalitions then
25                 host_machine_available ← remove(host_machine_available,
                   host_machine)
26                 statuschange(host_machine, max_coalition)
27         foreach host_machine in machines_available do
28             update host_machine.preference_list
```

Since the algorithm is meant for a cloud, which is most often a distributed environment, the complexity of Algorithm 1 is calculated based on the number of messages that are sent during the execution. Algorithm 1 computes all the feasible coalitions for each host machine HM_p. In order to avoid increased communication cost and to maximize the payoff for each host machine, it would be preferable to limit the number of members in a coalition. For a coalition of size r and machines being allowed to be part of a single open coalition at a time, the number of messages that are sent in the system overall is $r \times \sum_{u=0}^{(t/r)-1} \binom{t-u \cdot r}{r-1}$ [5]. Considering that an agent can be part of more than one coalition in the modified approach, we still see that, after each iteration, the number of host machines yet to join an open coalition decreases monotonically. If we consider, on an average, h as the number of host machines that are removed from the set of available machines in each round, then the number of iterations for the algorithm is (t/h) [5]. In every successive rounds, the algorithm chooses $r-1$ host machines from $t-h-1$. The feasibility check of each coalition would require $r-1$ messages to be sent. If the number of open coalitions that a host machine can be part of simultaneously is v, then the total number of messages sent in the system sums to $v \times r \times \sum_{u=0}^{(t/h)-1} \binom{t-u \cdot h}{r-1}$.

2) Coalition Dissolving Algorithm: Algorithm 2 for dissolving a coalition takes in the coalition to be dissolved and the task being serviced by it as inputs. The algorithm checks for the completion of the task. If the task under consideration is completed, the members of the coalition are added to the set of available agents.

Algorithm 2: Coalition Dissolving Algorithm

```
   Data: coalitions_set
   Data: coalition
   Data: host_machine_available
   Data: task
1  begin coalitionDissolvingAlgorithm
2      if task.status=COMPLETE and task.id=coalition.task_id and coalition.ID
       = task.coalition_id then
3          foreach host_machine in coalition do
4              add(host_machine_available, host_machine)
5              update host_machine.capability_vector
6              if host_machine.cur_no_coalitions = 1 then
7                  statuschange(host_machine, available)
8              else
9                  statuschange(host_machine, open_coalition)
10             host_machine.cur_no_coalitions ←
               host_machine.cur_no_coalitions - 1
11         remove(coalitions_set,coalition)
12         coalition ← NULL
```

If the task serviced by the coalition is found to be complete, the members of the coalition are added to the set of machines that are available in line 4. The released capabilities of the host machine is updated in line 5. The statuses of the member host machines are updated according to the current number of coalitions they are part of. If it is zero, the status is changed to available, else it is changed to open_coalition in lines 6–9. The current number of coalitions for the members is updated in line 10, the coalition list is updated in line 11, and the coalition itself is dissolved in line 12.

3) Task Allocation Algorithm: Algorithm 3 for task allocation takes in the task G_j to be allocated and the set of coalitions (coalitions_set) as inputs. If the necessities for the task are found in the capability vector of an OPEN coalition C_l, i.e., $N_{G_j} \sqsubseteq D_{C_l}$, then the task is allocated to the coalition in lines 6–7. The status of the coalition that is allotted the task is changed to ENGAGED in line 8. The capability vectors of the member host machines are updated in lines 9–10. If none of the open coalitions satisfy the necessity vector for the task, then the task allocator is asked to forcibly allocate the task to a group of host machines in lines 11–12.

Algorithm 3: Task Allocation Algorithm

```
   Data: task
   Data: coalitions_set
1  begin taskallocate
2      foreach coalition in coalitions_set do
3          if task.necessity_vector ⊑ coalition.capability_vector then
4              if task.status ≠ COMPLETE then
5                  if task.coalition_id=NULL and coalition.status=OPEN
                   then
6                      coalition.task_id ← task.id
7                      task.coalition_id ← coalition_id
8                      coalition.status ← ENGAGED
9                      foreach host_machine in coalition do
10                         update host_machine.capability_vector
11     if task.coalition_id=NULL then
12         forceallocate(task)
```

VI. Experiments

A. Setup

The experimental setup is similar to that of Mashayekhy and Grosu [12] and consists of the following specifications.

1) The four types of VM instances are specified in Table I. The instance types and pricing are similar to the ones used by Microsoft Azure [7].
2) Each task specifies the number of instances of each VM type it requires, i.e., $\mathcal{R} = \langle \text{no\_VM}_1, \ldots, \text{no\_VM}_s \rangle$, representing the number of small, medium, large, and extra large configurations of VMs, respectively.
3) We consider four different types of task requests, namely, (10, 0, 0, 0), (10, 10, 0, 0), (10, 10, 10, 0), and (10, 10, 10, 10), with the first number in each task request indicating the number of small VMs needed, the second indicating the number of medium VMs needed, and so on. The numbers of these requests were varied during runs, and the performances were compared as the number of tasks increased.
4) Three different task sets are considered: 1) Task set 1-one instance each of the four types of requests (total of 4 task requests); 2) Task set 2-two instances each of the four types of requests (total of eight task requests); and 3) Task set 3-three instances each of the four types of requests (total of 12 task requests).
5) The results are an average of 20 runs of the experiments.
6) The results obtained are based on simulations coded in Java run on the Java Run-Time environment on a 3.00 GHz Intel quad-core personal computer with 8 GB RAM.
7) The virtual topology consists of a total 400 machines, with 100 machines of each configuration. VMs of identical configurations were adjacently placed. The host

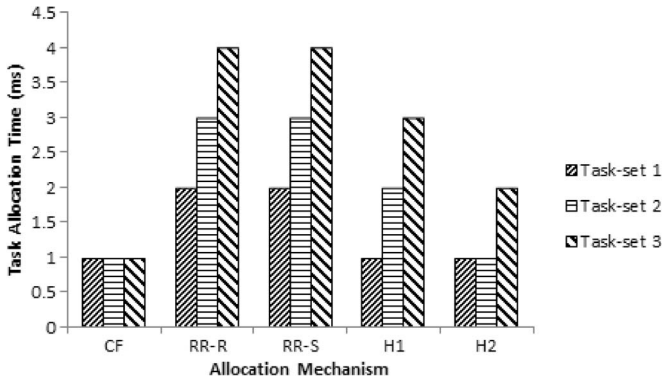

Fig. 2. Time for task allocation for the different strategies.

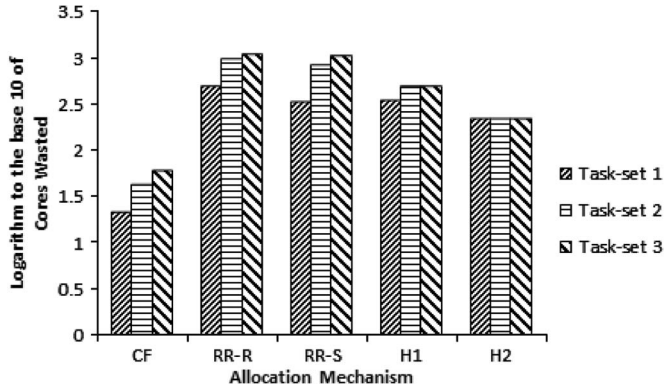

Fig. 3. Core wastage for the different strategies.

machine configurations are specified based on their number of cores and hard disk storage. The configurations considered are as follows: 1) one core, 125-GB disk storage; 2) two cores, 250-GB storage; 3) four cores, 500-GB storage; and 4) eight cores, 1000-GB storage.

8) Round Robin methods have been found useful for applications such as Hadoop [35]. The performance of our resource allocation algorithm was compared with common cloud resource allocation strategies [28], which include the following:
   a) RR-R: allocation in a round robin (RR) manner across the racks (R).
   b) RR-S: allocation in a round robin (RR) manner across the host machines (S). This is the default policy used by Eucalyptus [36] (cf. [37]).
   c) H-1: hybrid policy combining RR-S and RR-R preferring host machines in a rack but only to select a maximum of 20 host machines per rack.
   d) H-2: similar policy as H1 but to select only a maximum of 10 servers per rack.

B. Results

Through our experiments, we compared the different resource allocation strategies listed above with our resource allocation by coalition formation (denoted by CF in the graphs) based on the following criteria.

1) Task allocation time: The task allocation time is the duration between the submission of a request at the cloud service provider and the allocation of a single/group of machines to host the corresponding VMs. The results of the comparison on task allocation time for the above listed strategies are depicted in Fig. 2.
2) Resource wastage: The number of cores for each of the strategies was compared for the task sets, the results of which have been depicted in Fig. 3. This gives a measure of the resource wastage with the normal allocation strategies.

   The amount of storage wasted in GB for each of the strategies was compared, the results of which have been depicted in Fig. 4. This gives yet another measure of the resource wastage with the normal allocation strategies.

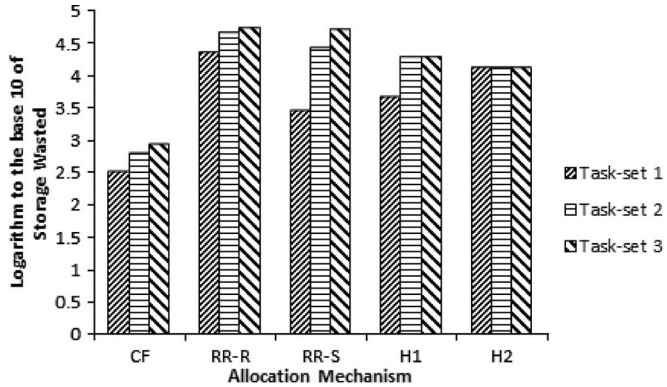

Fig. 4. Storage wastage for the different strategies.

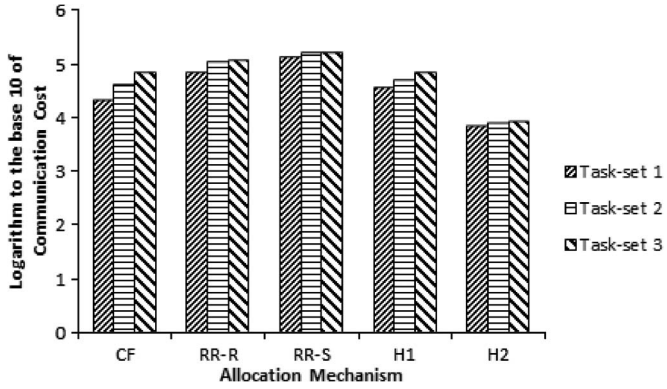

Fig. 5. Communication cost for the different strategies.

3) Communication cost: The communication cost is the cost of establishing communication among the members of a coalition. It is directly proportional to the number of coalition members and their placement on the racks. Closely-placed machines have lower communication costs. The logarithm to the base 10 of the communication cost between host machines in coalitions is plotted, as depicted in Fig. 5.
4) Request Satisfaction
   a) The number of unsuccessful VM assignments for each of the strategies was compared for Task sets 1–3, as shown in Fig. 6. This indicates that the lower task allocation time for certain strategies does not always mean that all the requested VMs have been successfully assigned.

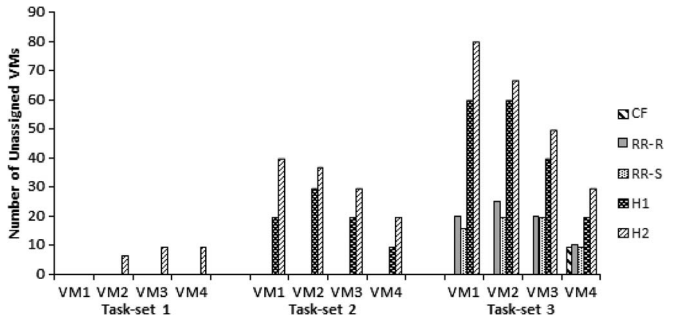

Fig. 6. Unassigned VMs for different strategies for task sets.

C. Discussion

From the results we listed in the previous section, we find that our resource allocation mechanism fares better than the resource allocation strategies commonly used in commercial IaaS clouds. The values for the graphs for resource wastage (core and storage) were log-normalized $(\log_{10})$ in order to scale the values of the resource allocation strategies. The results lead us to deduce the following points.

1) Task allocation time remains mostly constant for our resource allocation by coalition-formation approach. This can be attributed to the fact that open coalitions are formed even before the arrival of the actual requests. All that needs to be done is a lookup in order to assign the coalition with the least resource wastage to the request when it arrives.
2) Lower task allocation time does not always indicate successful VM assignments; it can be also caused by the inability to accommodate VMs using the remaining host machines based on that particular allocation strategy, as is evident from the cases of policies H1 and H2.
3) Resource wastage, whether with respect to the number of cores or the storage, remains lowest for our allocation strategy. This is also due to pre-configuration of coalitions, which can be assigned for tasks after ensuring minimum wastage.
4) As the number of task requests increases, the number of VMs that are successfully assigned gets rapidly lower with the H1 policy. For the other policies, the rate, although less dramatic, is higher than in our allocation strategy.
5) Since our allocation strategy is topology-aware, it assigns a higher payoff for VM coalitions that are closely-placed. This results in reduced communication cost and resource wastage. In Fig. 5, the hybrid strategies have lower costs, on account of incomplete request satisfaction resulting in lower communication.

VII. Conclusion

Resource allocation on the cloud aims at avoiding underutilization of resources. Through this work, we have shown the use of the uncertainty principle of game theory to model coalition formation among machines on the cloud. This is done to satisfy requests needing capabilities beyond that of a single machine. Virtualization of the required resources is facilitated by forming coalitions of host machines. For doing so, we devise a mechanism for resource allocation for tasks whose pattern of arrival is unknown. The advantage of our approach is that, by solving the optimization problem of coalition formation, we avoid the complexities of integer programming. In addition, our resource allocation mechanism, when deployed, is found to perform better with respect to lower task allocation time, lower resource wastage, and higher request satisfaction. As cloud systems are large, themselves costing billions of dollars, and handle requests costing tens of billions more, we believe that these are very significant real-world advantages.

References

[1] Amazon elastic compute cloud, Jun. 2013. [Online]. Available: http://aws.amazon.com/ec2/
[2] IBM Cloud Computing: Infrastructure as a Service, Jun. 2013. [Online]. Available: http://www.ibm.com/cloud-computing/us/en/iaas.html
[3] T. W. Sandholm and V. R. Lesser, "Coalitions among computationally bounded agents," *Artif. Intell.*, vol. 94, no. 1/2, pp. 99–137, Jul. 1997.
[4] G. Székely and M. L. Rizzo, "The uncertainty principle of game theory," *Amer. Math. Mon.*, vol. 114, no. 8, pp. 688–702, Oct. 2007.
[5] P. S. Pillai and S. Rao, "A resource allocation mechanism using coalition formation and the uncertainty principle of game theory," in *Proc. 7th Annu. IEEE Int. SysCon*, Orlando, FL, USA, Apr. 2013, pp. 178–184.
[6] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proc. 22d ACM SOSP*, 2009, pp. 261–276.
[7] Windows Azure, Jun. 2013. [Online]. Available: http://www.windowsazure.com
[8] P. K. Enumula, "Coalition formation in multi-agent systems with uncertain task information," M.S. thesis, Int. Inst. Inf. Technol. Bangalore, Bangalore, India, Jun. 2008.
[9] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[10] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Proc. 38th ACM ISCA*, 2011, pp. 319–330.
[11] G. E. Goncalves, P. T. Endo, and T. Damasceno, "Resource allocation in clouds: Concepts, tools, and research challenges," in *Proc. 29th Simpósio Brasileiro Redes Comput.*, 2011, pp. 197–240.
[12] L. Mashayekhy and D. Grosu, "A coalitional game-based mechanism for forming cloud federations," in *Proc. IEEE/ACM 5th Int. Conf. UCC*, Chicago, IL, USA, 2012, pp. 223–227.
[13] G. Shanmuganathan, A. Gulati, and P. Varman, "Defragmenting the cloud using demand-based resource allocation," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2013, pp. 67–80.
[14] J. Cao, Y. Wu, and M. Li, "Energy efficient allocation of virtual machines in cloud computing environments based on demand forecast," in *Proc. 7th Int. Conf. Adv. GPC*, 2012, pp. 137–151, Springer-Verlag.
[15] X. Chen, J. Zhang, J. Li, and X. Li, "Resource virtualization methodology for on-demand allocation in cloud computing systems," *Serv. Oriented Comput. Appl.*, vol. 7, no. 2, pp. 77–100, Jun. 2013.
[16] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason, "Auction protocols for decentralized scheduling," *Games Econ. Behav.*, vol. 35, no. 1/2, pp. 271–303, Apr. 2001.
[17] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *Proc. 7th Int. Symp. HPDC*, 1998, pp. 28–31.
[18] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," *J. Supercomput.*, vol. 54, no. 2, pp. 252–269, Nov. 2010.
[19] A. S. Prasad and S. Rao, "A mechanism design approach to resource procurement in cloud computing," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 17–30, Jan. 2014.
[20] O. Shehory and S. Kraus, "Methods for task allocation via agent coalition formation," *Artif. Intell.*, vol. 101, no. 1/2, pp. 165–200, May 1998.
[21] L. He and T. R. Ioerger, "Forming resource-sharing coalitions: A distributed resource allocation mechanism for self-interested agents in computational grids," in *Proc. 12th ACM SAC*, 2005, pp. 84–91.
[22] G. Yong, Y. Li, W.-M. Zhang, J.-c. Sha, and C.-y. Wang, "Methods for resource allocation via agent coalition formation in grid computing systems," in *Proc. IEEE Int. Conf. Robot., Intell. Syst. Signal Process.*, Oct. 2003, vol. 1, pp. 295–300.

[23] D. Niyato, A. Vasilakos, and Z. Kun, "Resource and revenue sharing with coalition formation of cloud providers: Game theoretic approach," in *Proc. 11th IEEE/ACM Int. Symp. CCGrid Comput.*, May 2011, pp. 215–224.
[24] M. Hassan, B. Song, and E.-N. Huh, "Distributed resource allocation games in horizontal dynamic cloud federation platform," in *Proc. 13th IEEE Int. Conf. High Perform. Comput. Commun.*, 2011, pp. 822–827.
[25] W. Saad, Z. Han, M. Debbah, and A. Hjrungnes, "A distributed coalition formation framework for fair user cooperation in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 9, pp. 4580–4593, Sep. 2009.
[26] L. Mashayekhy and D. Grosu, "A merge-and-split mechanism for dynamic virtual organization formation in grids," in *Proc. 30th IEEE Int. Perform. Comput. Commun. Conf.*, 2011, pp. 1–8.
[27] O. Sonmez, H. Mohamed, and D. Epema, "Communication-aware job placement policies for the KOALA grid scheduler," in *Proc. 2nd IEEE Int. Conf. e-Science Grid Comput.*, 2006, pp. 1–8.
[28] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz, "Topology-aware resource allocation for data-intensive workloads," in *Proc. 1st ACM APSys Workshop*, 2010, pp. 1–6.
[29] S. Kraus, O. Shehory, and G. Taase, "Coalition formation with uncertain heterogeneous information," in *Proc. 2nd Int. Joint Conf. AAMAS*, 2003, pp. 1–8.
[30] H. Hosam and Z. Khaldoun, "Planning coalition formation under uncertainty: Auction approach," in *Proc. Int. Conf. Inf. Commun. Technol.*, 2006, vol. 2, pp. 3013–3017.
[31] A. Bai, F. Wu, and X. Chen, "Online planning for large MDPs with MAXQ decomposition," in *Proc. 11th Int. Conf. AAMAS*, Richland, SC, USA, 2012, vol. 3, pp. 1215–1216.
[32] W. Heisenberg, "Über den anschaulichen Inhalt der quantentheoretischen kinematik und mechanik," *Z. Phys.*, vol. 43, no. 3/4, pp. 172–198, Mar. 1927.
[33] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *Amer. Math. Mon.*, vol. 69, no. 1, pp. 9–15, Jan. 1962.
[34] Oracle Enterprise Manager, Strategies for scalable, smarter monitoring using Oracle Enterprise Manager Cloud Control 12c, Jun. 2013. [Online]. Available: http://www.oracle.com/technetwork/oem/sys-mgmt/wp-em12c-monitoring-strategies-1564964.pdf
[35] D. Quan, R. Basmadjian, H. Meer, R. Lent, T. Mahmoodi, D. Sannelli, F. Mezza, L. Telesca, and C. Dupont, "Energy efficient resource allocation strategy for cloud data centres," in *Computer and Information Sciences II*, E. Gelenbe, R. Lent, and G. Sakellari, Eds. London: Springer-Verlag, 2012, pp. 133–141.
[36] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *Proc. 9th IEEE/ACM Int. Symp. CCGrid*, 2009, pp. 124–131.
[37] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proc. 16th ACM Conf. CCS*, 2009, pp. 199–212.

Parvathy S. Pillai (S'13) received the M.Tech. degree in information technology from the International Institute of Information Technology Bangalore, Bangalore, India. She is currently working toward the Ph.D. degree in computer science in the School of Computing, National University of Singapore, Singapore.

Her research interests include machine learning, data analytics, medical informatics, and cloud and distributed systems.

Ms. Pillai is a Student Member of the IEEE Computer Society and the Association for Computing Machinery.

Shrisha Rao (M'08–SM'13) received the M.S. degree in logic and computation from Carnegie Mellon University, Pittsburgh, PA, USA, and the Ph.D. degree in computer science from The University of Iowa, Iowa City, IA, USA.

He is currently an Associate Professor with the International Institute of Information Technology Bangalore, a graduate school of information technology in Bangalore, India. His research interests are in distributed computing, specifically algorithms and approaches for concurrent and distributed systems and include solar energy and microgrids, cloud computing, energy-aware computing, and demand-side resource management.

Dr. Rao is a member of the IEEE Computer Society, the Association for Computing Machinery, the American Mathematical Society, and the Computer Society of India.