Autonomic Resource Allocation for Cloud Data Centers: A Peer to Peer Approach

Mina Sedaghat, Francisco Hernández-Rodriguez, Erik Elmroth
Dept. of Computing Science, Umeå University, Sweden
{mina, hernandf, elmroth}@cs.umu.se

***Abstract*—We address the problem of resource management for large scale cloud data centers. We propose a Peer to Peer (P2P) resource management framework, comprised of a number of agents, overlayed as a scale-free network. The structural properties of the overlay, along with dividing the management responsibilities among the agents enables the management framework to be scalable in terms of both the number of physical servers and incoming Virtual Machine (VM) requests, while it is computationally feasible. While our framework is intended for use in different cloud management functionalities, e.g. admission control or fault tolerance, we focus on the problem of resource allocation in clouds. We evaluate our approach by simulating a data center with 2500 servers, striving to allocate resources to 20000 incoming VM placement requests. The simulation results indicate that by maintaining an efficient request propagation, we can achieve promising levels of performance and scalability when dealing with large number of servers and placement requests.**

***Keywords*-Cloud computing; Peer to Peer; Resource management;**

I. Introduction

The explosive growth of data centers, in terms of both size and number of servers, has greatly increased the complexity of managing data center resources. The most recent type of large scale data centers are cloud data centers, which are typically used for on-demand service provisioning. One key challenge in cloud data centers is to develop resource management frameworks and mechanisms to provide efficient resource utilization while they are also scalable and computationally feasible, with respect to the size of the data centers, the incoming load and their dynamic nature.

Most existing approaches to resource management [1], [2], [3] are highly centralized and do not scale with the number of servers in the data center. Typically, a centralized manager is required to execute the necessary complex algorithms and must also be aware of the state of all servers, which can be challenging in large and highly dynamic data centers [4].

In contrast, distributed approaches to resource management can cope with large numbers of resources without requiring centralized control. Within such approaches, the management responsibilities are divided among identical autonomic elements (nodes), helping the management structure to scale as the number of nodes increases. Global management is achieved through co-operative interactions between autonomic elements [5].

Peer to Peer (P2P) systems have proven to be scalable and robust for distributed resource management. In such systems, each peer performs a task based on locally-available information, and goal-oriented coordination among the tasks enables the system to achieve its global objective. The system thus benefits from a high degree of concurrency and decentralization of control with no central bottleneck.

However, P2P systems also face challenges due to the lack of global view of the system and not having a centralized point of reference. To compensate for this lack of global view, attempts have been made to extract and discover the required information via discovery algorithms that allow individual elements to obtain sufficient information when required.

In this paper, we address the issue of resource management in large cloud data centers, approaching it as an information discovery problem in a P2P structure. We propose a P2P resource management framework consisting of an agent community that interacts in a goal-oriented fashion. The agent community is structured as a scale-free network, enabling the agents to efficiently discover the information required for their decisions, using a simple *local search* algorithm. Our main objective is to identify a solution that is scalable both in terms of the number of servers and incoming VM requests while still being computationally feasible.

While our framework is intended to support different cloud management functionalities, e.g. admission control or fault tolerance, our primary focus is on the problem of resource allocation in clouds. As part of the work, we propose a resource allocation mechanism that aims to maximize data center utilization and profitability by ensuring high utilization of active nodes while minimizing overall power consumption by putting the remaining nodes into energy-saving mode.

We evaluated our approach by simulating a data center that has 2500 servers and must allocate resources to 20000 incoming VM placement requests. We analyzed our approach with respect to diverse performance criteria including data center utilization, profit, rejection ratio, request processing time (in terms of the number of hops per request), and scalability. We also investigated various factors that might affect performance. Our approach is shown to maintain good performance with respect to the examined criteria.

The remainder of the paper is organized as follows: Section II introduces the framework by describing the P2P overlay and the agent model. Section III presents the problem of resource allocation, the main objectives and presents a *local search* algorithm to solve the resource allocation problem. Section IV and Section V discuss our experimental setup and the results obtained in the simulations conducted to evaluate the approach. Section VI provides a brief overview of related studies, and concluding remarks are presented in Section VII.

II. Resource Management Framework

Resource management problems are often formulated as optimization problems. In order to solve them, we adopt a P2P approach, using a distributed *local search* algorithm on a population of peers, where each peer considered as a potential solution checking its neighbors in the hope of finding an improved solution.

In our design, the physical servers are structured as peers, and peers that are connected to one-another are considered neighbors. Each peer is associated with an agent that is responsible for functional tasks and local managerial decisions. Relevant information is exchanged among agents via a gossip protocol exploiting the environment formed by the peers. Each agent makes local decisions with respect to its local view and policies, and the system as a whole progresses towards the global objective via the emergent outcome of these local decisions.

A. *Overlay Construction*

In P2P systems, the overlay specifies the logical interconnections between peers. The structural properties of the overlay affect the efficiency of the discovery and propagation of information within the system, so it must be designed carefully. The problem of choosing an overlay can be formulated as a graph theoretic problem, with the physical servers (nodes) being the vertices, logical links being the edges, and nodes that are connected to one-another via an edge being neighbors that collectively form a neighborhood.

The goal is to find an overlay that is robust to failures and capable of supporting fast discovery while having a low maintenance cost (i.e. the cost of keeping nodes up-to-date about their neighbors). In graph theoretic terms, such a graph is characterized as being highly connected, sparse, with a low diameter.

Scale-free networks are a family of graphs that are widely used for structuring P2P overlays because they satisfy the criteria listed above. Scale-free networks are scalable and robust to random node failures. In addition to their robustness, these graphs have short distances between any two randomly chosen vertices and each vertex can be reached within a limited number of steps. This enables fast resource discovery, which is essential for our purposes.

In our method, the servers in the data center are structured in the form of a scale-free network that is constructed using the Barabsi Albert (BA) algorithm with a preferential attachment mechanism [6].

Such a logical structure for the resource management framework can be simply mapped into the future data center's network architectures [7], and can benefit from faster communications, resulted from the compatibility between the logical structure and the physical network structure [7].

B. *Agent Model*

On top of the P2P overlay, we build an agent community that performs functional tasks while enabling goal-oriented communication. Each agent is an autonomous entity that acts on behalf of a physical server (peer) or an application.

We associate each physical server and each application with an agent. These agents interact with each other to perform their designated tasks.

1) **Node agents** process information received from their neighbors to advance their local goal, e.g. increasing their own resource utilization or that of their neighborhood. They also direct relevant information to their neighbors. Each agent tracks information on the state of its associated physical server, including its utilization and available capacity, as well as information on its neighbors such as their state (e.g. whether they are idle, crashed or active), utilization, and free capacity.
2) **Application agents** are responsible for monitoring the application's resource demand, generating requests for more or fewer resources as the demand changes, and interacting with *nodeAgents* to deploy the new resources. The *applicationAgent* resides on one of the physical machines on which the application is deployed, and keeps track of the VMs allocated to the application.

To solve a resource management problem, each *nodeAgent* solves a local optimization problem by searching for a locally optimal solution within its own local scope. The search proceeds iteratively from one potential solution to an improved alternative until no better solution can be found among the *nodeAgents*. Relevant information is either exchanged or distributed among agents via gossiping. The *nodeAgents* use heuristics to identify the node that offers the highest objective value.

III. Resource Allocation

We formulate the problem of placing VMs on a set of physical servers as an optimization problem.

We model the data center as a set of n physical servers, structured as a scale-free network, where each server has the capacity C_{server}. Using existing CPU/memory based capacity tools, the capacity of a server is defined in terms of number of available slots that can accommodate VMs. For clarity we assume that the servers are homogeneous, although the formulation can easily be extended for heterogeneous servers.

The data center offers *k* VM types, where VM-type$_i$ (*i = 1,.., k*) has capacity C_i compute units ($C_i < C_{i+1}$ and $C_k \leq C_{\text{server}}$) whose price is proportional to its size.

Moreover, assume that there are *m* VM placement requests, where each request *j, j = 1,..,m* demands capacity *Demand*$_j$. The capacity that is actually allocated to request j is denoted as *Res*$_j$. The problem is to allocate resources (slots) on the physical servers to the VMs in order to optimally fulfill a data center management objective, e.g. to maximize resource utilization and overall profit.

We define the data center utilization as the total resources allocated at time *t* to all VM placement requests divided by the total available capacity of the data center. The resulting optimization problem is formulated as:

$$Maximize\ U_{dc}(t) = \frac{\sum_{j=1}^{m} Res_j(t)}{n \times C_{server}} \qquad (1)$$

subject to:

$$\sum_{j=1}^{m} Res_j(t) \leq n \times C_{server} \qquad (2)$$

Where *m* is the number of placement requests, *Res*$_j$*(t)* is the allocated capacity for *request*$_j$ at time *t*, *n* is the total number of servers (considering both idle and active servers), and C_{server} is the capacity of each server.

We can also formulate the resource allocation problem to optimize *profit*. As shown in Equation (3), we define the profit as the revenue earned from allocating the VMs minus the associated operational cost, which is formulated in terms of the cost of the servers' power consumption. The power consumption is modeled using a linear function that is shown in Equation(4), with a fixed consumption for the idle state and additional power usage proportional to the server's utilization [8]. The profit optimization problem is thus formulated as maximization of the following function:

$$Profit(t) = \sum_{j=1}^{m} Res_j(t) \times price_{Res_j} - \sum_{i=1}^{n} (P_i(t)/P_{max}) \times cost_i \qquad (3)$$

This objective is also subject to constraint (2). $P_i(t)$ is the power consumption of the server at time *t*, which is calculated as:

$$P_i(t) = (P_{max} - P_{idle}) \times U_{node_i}(t) + P_{idle} \qquad (4)$$

Here, $price_{Res_j}$ is the price of renting *Res*$_j$*(t)* from the data center (i.e. the data center's income), $cost_i$ is the power consumption cost for a fully utilized server, $P_i(t)$ is the power consumption of the server at time *t* when its processor utilization is $U_{node_i}(t)$, P_{max} is the power consumption at maximum utilization, and P_{idle} is the server's power consumption when idle.

The objective function is maximized when the aggregated power consumption of all active servers is minimized. In a homogeneous datacenter, this happens if the demand of the VMs is consolidated over the minimum servers [9].

A. *Resource Allocation Algorithm*

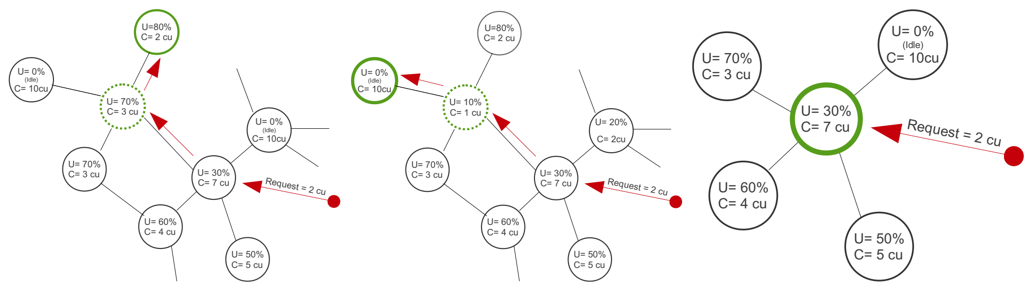
We propose a resource allocation algorithm, based on *local search* heuristics. This algorithm is, intrinsically, a discovery algorithm that searches for nodes according to a set of rules and specifications. A *VM placement request* traverses the network looking for a set of nodes that satisfy its resource demands. We use the term *request* as an abbreviation for *VM placement request* in the remainder of the paper.

The *nodeAgent* receiving a request selects the best potential node with sufficient capacity that can host the VM based on its locally stored information about its neighborhood. If the selected node is one of the neighbors rather than the node that received the request, the *nodeAgent* will forward the request to that neighbor so that the search for the desired resource can continue. *NodeAgents* iteratively forward the request from one potential solution to a better one until the visited *nodeAgent* is not able to find any better solution than itself or the request is expired. There are multiple heuristics that could potentially be used by the *nodeAgents* to select the best neighbor, including:

1) **Most-Utilized**: Selects the *most-utilized* node with sufficient capacity from among its neighbors (including itself). The utilization of a node is the ratio of its utilized resources to its total capacity.
2) **Least-Utilized**: Selects the *least-utilized* node with sufficient capacity from among its neighbors (including itself).
3) **First-Fit**: Selects the *first* node with sufficient capacity from among its neighbors (including itself).

If the *nodeAgent* and one of its neighbors both offer equal objective values, the algorithm chooses the neighbor over the processing *nodeAgent*, to increase the chance of finding a better solution in the future visits and possibly skips the local optima. If none of the neighbors or the *nodeAgent* itself have sufficient capacity, the *nodeAgent* forwards the request to a random neighbor. Hence, if a request is stocked in a neighborhood with no available resources, random selection helps the request to bounce between the nodes and find a way out of the saturated neighborhood so that the search can continue.

Whenever a request is forwarded to a next *nodeAgent*, it is considered to have taken one *hop*. The total number of hops required to successfully locate a node represents the time required to process the request. We limit the request processing time using a *maximum Hops To Live (HTL)* threshold. The request is rejected if the required number of hops exceeds the *HTL*. Limiting the number of hops reduces the quality of the resulting solution and increases the



(a) Most-Utilized heuristic (b) Least-Utilized heuristic (c) First-Fit heuristic

Figure 1: The selection of a node for further processing of a request based on three different heuristics

Algorithm 1 Allocation

Input: Request [Demand, HTL, DecisionFlag]
Output: The location to place the request, DecisionFlag

```
On receiving a request :
if DecisionFlag = false then
  if $HTL > 0$ then
    t = identify the node w.r.t the heuristic policy and local view
    if t != $Me$ then
      if t =-1 then
        t = random neighbor from the neighborhood
      end if
      HTL = HTL - 1
      forward the request to t
    else
      location = t
      DecisionFlag = true (decision is made)
    end if
  else
    location = -1 (Location not found)
    DecisionFlag= true (Decision is made)
  end if
end if
return location and DecisionFlag
```

number of rejections. However, it also prevents the indefinite propagation of requests.

We also assume that requests can either reach the system from a unique entry point (which is referred to as "*central entry*") or from multiple entry points ("*distributed entry*"). In both cases, they propagate from their entry points according to the same rules.

There are thus six possible combinations of heuristics and entry policies, as shown in Table I.

Policies	Request entry	Heuristics
Central-FF	Central	*First-Fit*
Central-Min	Central	*Min Utilization*
Central-Max	Central	*Max Utilization*
Dist-FF	Distributed	*First-Fit*
Dist-Min	Distributed	*Min Utilization*
Dist-Max	Distributed	*Max Utilization*

Table I: Resource allocation policies based on different entry policies and heuristics

B. Characterization of the Resource Allocation Protocol

The efficiency of the resource allocation protocol is defined with respect to our two main objectives, i.e. the maximization of data center utilization and high profit. It depends on the efficiency of request propagation and the distribution of the allocated resources.

1) **Request propagation**: An efficient allocation algorithm should efficiently propagate the *VM placement requests* within the environment in a way that maximizes the likelihood of an effective visit. Effective visits are those that increase the chance of finding a node that offers a higher objective value within the *HTL* limit.
2) **Allocation distribution**: The second major factor that affects the efficiency of a resource allocation protocol is the distribution of the allocated resources. When allocated resources are sparsely distributed, it is probable that a large number of servers will have low utilization. This leads to high power consumption, increased operational costs and reduced profit. These consequences can be mitigated by distributing the allocation of resources such that a few servers are highly utilized and the rest can be put into energy saving mode.
   A sparse resource distribution also leads to fragmentation of the resources over the data center's resource pool such that the available resources on each node may be too small to place a VM even though the aggregate available capacity is still large. This can cause increased request rejection and decreased utilization.

C. Optimal Re-consolidation

As discussed in Section III-B, the distribution of the allocated resources directly affects the performance of the allocation algorithm. The ultimate allocation is an emergent consequence of the heuristic adopted by the *nodeAgents*. In addition to the *nodeAgent's* heuristic decisions, the frequent arrivals and terminations of VMs can also lead to a highly sub-optimal distribution of allocated resources over time. Therefore, it can sometimes be advantageous to migrate a previously allocated VM from one node to another, either to switch off a server or to optimize the allocation distribution to open up space for larger VMs.

Algorithm 2 Re-consolidation

```
if MyLoad < Re-consolidationThreshold then
    for i:=1 to NumberVMsDeployedOnMe do
        request = initiate a request for $VM_i$ [Demand$_i$, HTL, false]
        newHost = Allocation (request)
        if newHost != - 1 then
            Migrate $VM_i$ to newHost
        end if
    end for
end if
Recalculate MyLoad
if MyLoad = 0 then
    Set Me into energy saving mode
end if
```

This process is also known as *re-consolidation* of currently deployed VMs with the goal of increasing a node's utilization and potentially reducing the incidence of rejections due to resource fragmentation. To perform *re-consolidation*, *nodeAgents* representing lightly loaded servers autonomously or regularly migrate their loads to more heavily loaded nodes by initiating a request, similar to the initial placement request, for each of their deployed VMs. This request searches for the most highly utilized node with sufficient capacity to be the new host for the VM. If all the node's VMs are migrated successfully to other nodes, the node can be switched into a power saving mode. Re-consolidation also makes it easier to accommodate larger VMs and reduces the likelihood of rejection.

During re-consolidation often a performance impact can be expected. This impact can be modeled [10] and be taken into consideration before deciding on the re-consolidation. However, it has been shown that advancements in virtualization techniques [11] and technologies effectively reduce the performance overheads and its associated impacts [12]. Modeling this impact is not the main focus of this study, however we can simply extend our model to perform a cost-benefit analysis before re-consolidation, considering the overhead costs.

IV. Experimental Setup

This section describes an evaluation of the performance of the proposed approach through a simulation of a data center with 2500 physical nodes. We simulated our framework in the Netlogo environment and built our P2P overlay using the scale-free network model as implemented by [13].

Each physical node was associated with a *nodeAgent* and assumed to be capable of serving VMs of different types. The maximum capacity of each physical node was set to 10 compute units and the provider was assumed to offer 10 VM types with capacities ranging from 1 to 10 compute units. The price of a VM providing 1 compute unit was 0.01$ per time unit, similar to that for a Linux micro reserved-instance in Amazon's EC2 system. The prices of larger VMs were proportional to their capacity.

We assumed a total of around 20000 incoming VM placement requests, arriving the system following a Poisson arrival rate. Each request had a capacity demand (in compute units) that was selected at random from the set {1,...,10} and was mapped to a VM type. The VMs were deployed and terminated over the course of each simulation. Services running in clouds usually have an indefinite lifetime, so we modeled VM lifetime using a normally distributed random variable in order to eliminate the potential for systematic bias associated with specific application types and to represent the diversity of applications that may be deployed in a cloud data center.

In order to avoid infinite request forwarding in the environment, we constrained the number of hops per request to *HTL = 20 hops*.

The *re-consolidation threshold*, i.e. the load at which a node's resources are considered for re-consolidation, was set to 40% of the node's total capacity. Simulations were allowed to run for 20000 time units with each time unit representing 0.1 sec of simulation time and one hour of resource usage.

A. *Performance Parameters*

We evaluated our approach with respect to the following performance parameters:

1) **Data center utilization** ($U_{dc}(t)$)**:** This variable represents the utilization of the data center. It is defined as the total capacity used by the allocated VMs at time t relative to the total available capacity in the data center. Data center utilization is calculated using Equation (1), which was introduced in Section III.
2) **Average node utilization** ($U_{node}(t)$)**:**

$$U_{nodes}(t) = \frac{\sum_{j=1}^{m} Res_j(t)}{n_{active} \times C_{server}} \quad (5)$$

where n_{active} is the number of active nodes in the environment. This metric provides insight into the distribution of allocations and how efficiently the currently active servers are being utilized. It is directly proportional to the system's power consumption and the associated costs.

3) **Number of hops:** This is the number of steps required to locate a node that has the capacity required by the VM request. This metric measures how quickly the algorithm can locate a suitable node and respond to a request, and can be compared to the computation time in centralized approaches.
4) **Rejection ratio:** This is the proportion of request demands that are not satisfied.

$$RR(t) = \frac{\sum_{j=1}^{m} \mathit{Demand_j(t)} - \sum_{j=1}^{m} \mathit{Res_j(t)}}{\sum_{j=1}^{m} \mathit{Demand_j(t)}} \quad (6)$$

Rejections may occur for various reasons, including:
   - A failure to locate a suitable resource within the *HTL* limit.
   - A lack of sufficient resources to serve the request.
   - Fragmentation of the resource pool.
5) **Profit:** This represents the revenue of the data center with respect to the service provided and its operational costs. The profit is calculated using Equation (3), introduced in Section III.

V. Results and Discussion

This section describes how the performance is influenced by two key properties, namely request propagation and allocation distribution, and their impact on allocation policies. We evaluate performance in terms of the five metrics introduced above, *Data center utilization*, *Node utilization*, *Number of hops*, *Rejection ratio* and *Profit*. Finally, we study the scalability of the approach when each of the 6 allocation policies is adopted.

A. *Impact of Request Propagation on Performance*

In the first series of experiments, we studied the impact of request propagation on the performance of the resource allocation mechanism. Request propagation can be affected by the entry of requests to the system, the constraints imposed by *HTL* threshold, and modifications of the overlay topology. Due to paper limits, we only discuss the impact of requests' entry on the performance.

To determine how entry policy affects system performance, we compared the impact of adopting *central* and *distributed* entry policies (see Section III-A) for VM requests on system performance.

Figure 2 shows the number of hops, rejection ratio, data center utilization, node utilization, and profit for each of the policies listed in Table I. In general, *distributed entry* provides better request propagation, requiring fewer hops to place the VM. This is because *distributed entry* of requests automatically increases the probability of request propagation to a 'better' node and thus reduces the number of hops. The lower the number of hops, the lower the likelihood of exceeding the *HTL* and thus the lower the likelihood of request rejection. Reducing rejection ratios also increases data center utilization and profits. However, *distributed entry* generally produces lower node utilization ($U_{node}(t)$) than *centralized entry* because it results in a more sparse placement of VMs.

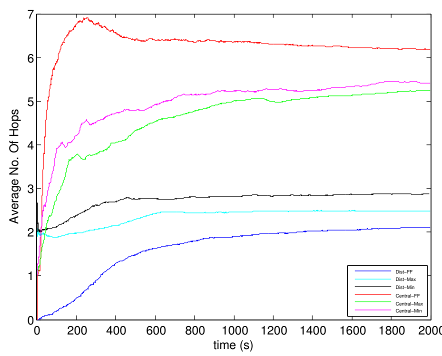
Central entry requires more hops to place a VM, leading to a higher rejection ratio and reduced data center utilization. That is to say, such policies suffer from *weak request propagation* in comparison to *distributed entry* alternatives. When requests are bound by a *central entry* policy, a small proportion of nodes experience a large number of visits (especially those in the vicinity of the entry node) while others are never visited. Consequently, the VMs tend to cluster in the neighborhood of the entry node. After a while, these nodes become fully loaded because they are so frequently visited, and become unable to accept new VMs. Subsequent requests must therefore travel beyond the saturated neighborhood, reducing the likelihood of successful allocation within the *HTL*. This in turn increases the rejection ratio. Because some fraction of the nodes can never be reached within the *HTL*, data center utilization is reduced and profit decreases.

B. *Impact of Allocation Distribution on Performance*

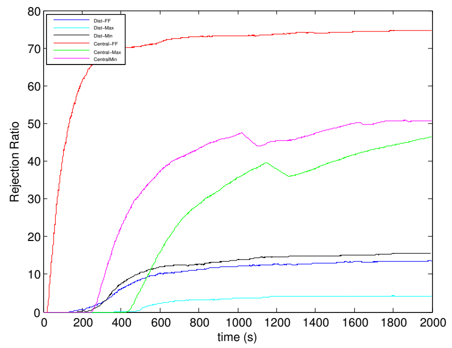
The distribution of allocated resources over the data center resource pool is the second major factor that affects the performance. It is determined by the local heuristic decisions of each *nodeAgent* in conjunction with the system's request entry policy. To determine the impact of allocation distribution on performance, we compared the 6 policies presented in Table I when used in conjunction with 3 heuristics: *Least Utilization*, *Most Utilization* and *First-Fit*.

The average node utilization, $U_{node}(t)$, is a useful metric for analyzing the distribution of allocated resources. For a given amount of allocated capacity, we can either distribute the allocations sparsely to provide a large number of lightly-loaded nodes, or we can consolidate them across a small number of nodes with high $U_{node}(t)$ values.

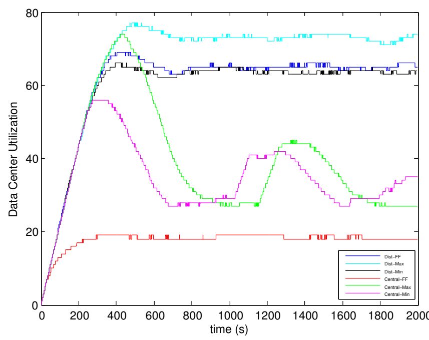
Figures 2a and 2b show the number of hops and the rejection ratios for each allocation policy. Of the six policies *Central-FF* requires the highest average number of hops to find a resource and place the VM, and it also has the highest rejection ratio. This can be explained by the fact that the weak request propagation of the *central entry* policy along with the saturation of the entry node's neighborhood caused by the *First-Fit* heuristic forces requests to make extra ineffective hops. This increases the required number of hops needed to locate a resource, causing requests to exceed the *HTL* and be rejected. The consequence of this is clearly shown in Figures 2b and 2c: the rejection ratio increases and data center utilization decreases. However, because allocated VMs are densely distributed over a small number of nodes within the vicinity of the entry node, this policy achieves high utilization values $U_{node}(t)$ for the active nodes, although the number of active nodes is limited. It is important to point out that data center utilization and



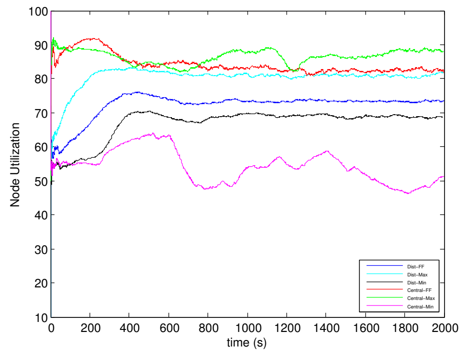
(a) Number of hops



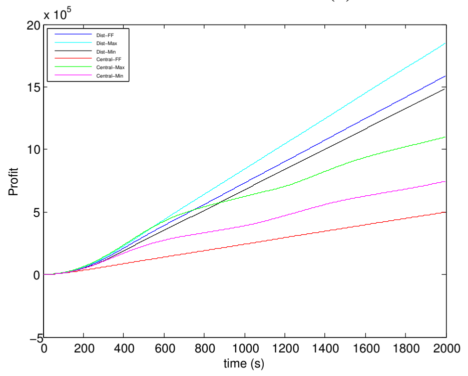
(b) Rejection ratio



(c) Data center utilization



(d) Node utilization



(e) Profit

Figure 2: The effects of different allocation policies in a data center with 2500 nodes

node utilization $U_{node}(t)$ are not necessarily correlated. For example, as shown in Figure 2, while the overall data center utilization rate when using the *Central-FF* algorithm is only 20%, the 20% of active nodes have an average utilization level of 80% (i.e. $U_{dc}(t_1) = 20\%, U_{node}(t_1) = 80\%$). Due to the high rejection ratio, this policy produces the lowest data center utilization and thus generates the lowest profit of all the policies evaluated.

On the other hand, we can see that the *Dist-FF* policy requires the fewest hops because it generates no saturation and the allocated VMs are distributed across the entire data center. The number of hops in this case is low because the *First-fit* heuristic merely selects the first node with available capacity and does not search further. The low number of hops results in a low rejection ratio and a fairly high data center utilization and profit.

Central-Min has the second highest number of hops and rejections. This is because the *Least-utilized* heuristic generates a lot of lightly loaded nodes due to its policy of placing VMs on loads with low utilization. In this situation, the resources are fragmented over the data center's resource pool and it becomes harder to place large requests. Consequently, the number of rejections increases because each available fraction is too small to place a large request, even though the total available capacity remains high. The *central-entry* of the requests is also another reason for the high rejection ratio under the *Central-Min* policy due to its weak request propagation. As shown in Figure 2d, this policy has a low node utilization $U_{node}(t)$ value because it frequently starts idle nodes and generates a large number of lightly utilized active nodes. The low data center utilization and high number of active nodes makes this policy one of the least profitable options.

Dist-Min policy provides better request propagation due to the *distributed entry* of requests, and thus increases the data center utilization relative to its *Central-Min* counterpart. However, the *Least-utilized* heuristic, which is common to both the *Dist-Min* and the *Central-Min* policies, causes resource fragmentation and increases the number of rejections.

As shown in Figure 2, *Dist-Max* achieves the best performance of the tested policies. The *Most-utilized* function selectively places VMs on nodes with sufficient capacity and the highest overall utilization. This heuristic automatically avoids the fragmentation of the resources by consolidating as many VMs as possible onto each active node, and can therefore accommodate more demand than the *Least-utilized* approach. It also avoids activating idle nodes because it tries to place the VMs onto currently active nodes to increase their utilization. This is why *Dist-Max* is the most profitable policy: it achieves the greatest possible data center utilization with the lowest possible number of active physical servers.

In summary, policies based on the *distributed entry* of requests offer better request propagation, yielding better performance and higher profits. The heuristic that offers the best performance and profitability is *Most-utilized*, followed by *First-fit*. Policies based on the *Least-utilized* heuristic has the lowest performance when the main objective is profit and having high consolidation. However, policies adopting the *Least-utilized* heuristics can be effective when other objectives such as load balancing is the main concern.

C. *The Impact of Re-consolidation on Performance*

In the previous section we showed that *Least-utilized* heuristics generate lightly loaded servers, causing fragmentation of the resource pool that leads to low data center utilization and reduces profits. In addition, the frequent arrival and termination of VMs can also lead to a far from optimal allocation distribution that may affect the performance of the resource allocation mechanism over time. In this section, we study how the re-consolidation of VMs can improve performance in such situations.

Table II shows the total number of servers put into power saving mode after re-consolidation during the simulation time when the allocation algorithm used in the simulation follows each of the six above-mentioned policies. The number of servers that undergo re-consolidation under *central entry* policies is low relative to that for *distributed entry* policies. This is because in *central entry* policies, most of the allocations are densely populated within the entry node's vicinity and resources are not sparsely allocated. Consequently, the number of nodes that are lightly loaded enough to trigger the re-consolidation process is much lower than under *distributed entry* policies.

Policies	# nodes hibernated due to re-consolidation
Central-FF	35
Central-Max	202
Central-Min	2693
Dist-Max	1030
Dist-FF	2147
Dist-Min	3679

Table II: Number of nodes put into hibernation after re-consolidation

Both Table II and Figure 3 show that *Least-utilized* policies benefit the most from the re-consolidation process. It is also clear that the resource distributions generated by *Most-utilized* policies are those that change the least following re-consolidation. This is because these policies preferentially deploy VMs onto highly loaded nodes in the first place.

The most significant impact of re-consolidation is on the node utilization, $U_{node}(t)$, because it packs the allocations onto a smaller number of servers and thus reduces the number of active servers while increasing their utilization. We should however note that not all re-consolidations lead to servers being powered down. The *nodeAgent* may look for potential hosts to migrate its deployed VMs one by one, but this does not necessarily mean that appropriate new locations will be found for all of them.

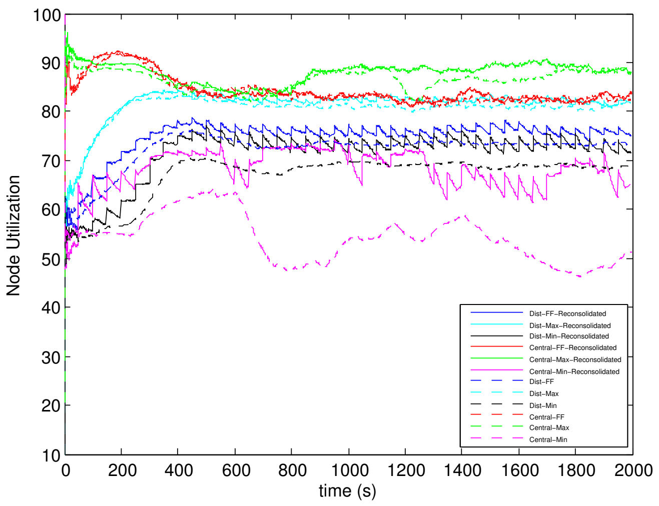


Figure 3: Impact of re-consolidation on node utilization

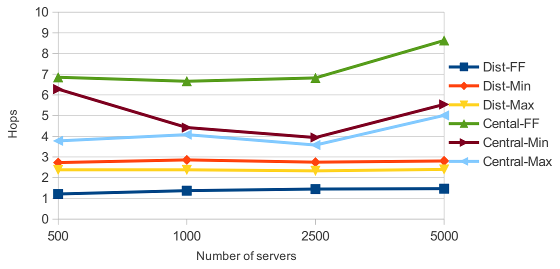
D. *Scalability Analysis*

To study the scalability of our approach, we performed simulations for data centers with 500, 1000, 2500 and 5000 servers, with 800, 1750, 4500, and 9000 VM requests, respectively. In this experiment, VMs were not re-consolidated and also not terminated so that we could study the scalability in an extreme case where all of the resources in the data center are saturated.

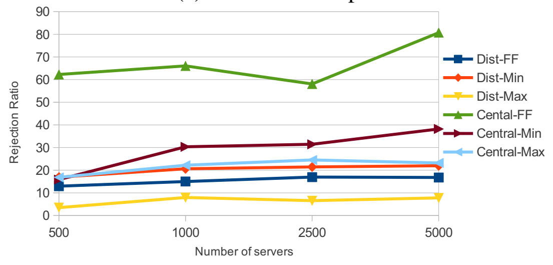
Figure 4 shows the performance of each policy with respect to server count. It is clear that the performance does not depend on system size provided that the allocation policies maintain adequate request propagation. For allocation policies based on *distributed entry*, performance metrics such as the *number of hops*, *rejection ratio* and *data center utilization* remain constant as the number of servers increases. This is because in systems with large numbers of servers, the main concern is to ensure that all nodes can be reached efficiently within an acceptable number of hops. This is straightforward when using *distributed entry* policies due to their favorable request propagation properties. Because requests are propagated efficiently, increasing the number of servers does not increase the number of hops or the frequency of VM rejection. As discussed above, data center utilization is highly dependent on the rejection ratio; because the rejection ratio is independent of the server count in this case, the data center utilization is as well.

However, this approach is not scalable when it is applied in conjunction with an allocation policy that has weak request propagation such as *Central-FF*. This is because such policies prevent requests from reaching most of the servers in the system. Therefore, as the number of servers increases, more requests are rejected and more servers remain unutilized.

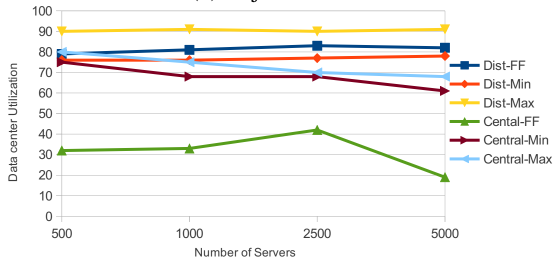
The *Node utilization* metric does not capture the dynamics of the system when the size is increased because it just expresses the utilization of active nodes.



(a) Number of hops



(b) Rejection ratio



(c) Data center utilization

Figure 4: Scalability of the approach with respect to increasing numbers of servers

VI. Related Work

There are a number of researches that are related to our study.

The first group of studies focused on the problem of VM placement adopting a centralized approaches, tackling the problem by formulating it as a knapsack or constraint satisfaction problem and generating solutions using integer programming methods [2], [14], [15]. These methods provide high quality solutions for limited numbers of servers and applications but must compromise on solution quality when applied to large scale data centers in order to achieve computational tractability.

The second group of related research includes studies that examined P2P approaches for resource management in cloud environments. Barabagallo et al. [16] modeled a data center as a P2P network of self-organizing nodes that collaborate with one-another using bio-inspired algorithms. This collaboration allows the nodes to redistribute the load among servers in order to increase the system's energy efficiency. Their idea is to have a number of entities known as scouts that investigate and gather information about virtual machines in the data center. This information is then used by other virtual machines to initiate migrations in order to redistribute the overall load. Their approach is related to our work on optimal re-consolidation. However, we have shown that a P2P approach can be adopted for wider problems such as resource allocation, and that re-consolidation is just one component of the broader resource allocation problem. Our work also deals with business goals such as utilization and profit, and yields improvements in energy efficiency as a consequence of achieving these goals.

Wuhib et al. [17] used a gossip protocol for dynamic resource management in clouds. In their protocol, the nodes interact with a subset of other nodes via small messages. These messages allow nodes to exchange state information and then compute a new configuration with the goal of maximizing cloud utility. If the gain from a new configuration outweighs the cost of change, they adopt the change and update their local state. This approach differs from ours in terms of the type, the purpose of the interactions and gossip: the authors' main focus is on the fairness of their allocations whereas we focused primarily on data center utilization and profit.

Marzolla et al. [18] also adopted gossiping for server consolidation in order to decrease power consumption. Their main focus is on the migration of arbitrary placed applications as a way of decreasing power consumption. As mentioned above, we approach the migration (re-consolidation) process as part of a larger solution to optimal resource allocation.

VII. Conclusions

In this paper we discussed a novel approach to perform VM placements in cloud data centers. Our approach benefits from high degree of concurrency and decentralization of control with no central bottleneck. Our main contributions are:

1) A new formulation of resource management problem through a P2P framework.
2) Proposing a P2P overlay based on scale-free network for robust and efficient discovery of the most suitable potential server for VM placement.
3) A resource allocation algorithm based on local search, designed to maximize data center utilization and profitability. The algorithm ensures high utilization of

active nodes while minimizing overall power consumption by putting the remaining nodes into energy saving mode.

We investigated the impact of different heuristics on the quality of the resulting allocations, with respect to the specified objectives of maximizing data center utilization and profitability. We also studied the scalability of our approach by evaluating its performance with different numbers of servers. Our approach was shown to be scalable up to at least systems of 5000 nodes with 9000 incoming placement requests arriving during the simulation time when using policies that allow for efficient request propagation.

We also present a re-consolidation process as a component of the broader resource allocation process. This enables the optimal re-allocation of currently running VMs. The re-consolidation process is designed to redistribute allocations among the servers in a data center in order to utilize the active nodes more efficiently in cases where the existing allocation has become sub-optimal. Efficient utilization makes it possible to switch off lightly loaded servers and reduce the center's overall power consumption, thereby increasing profits.

Acknowledgment

Financial support for this work is provided in part by the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control and the Swedish Government's strategic research project eSSENCE.

References

[1] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, *et al.*, "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[2] W. Li, J. Tordsson, and E. Elmroth, "Virtual machine placement for predictable and time-constrained peak loads," in *Economics of Grids, Clouds, Systems, and Services*, pp. 120–134, Springer, 2012.

[3] T. Püschel, N. Borissov, M. Macías, D. Neumann, J. Guitart, and J. Torres, "Economically enhanced resource management for internet service utilities," in *Web Information Systems Engineering–WISE 2007*, pp. 335–348, Springer, 2007.

[4] C. Mastroianni, M. Meo, and G. Papuzzo, "Self-economy in cloud data centers: statistical assignment and migration of virtual machines," in *Euro-Par 2011 Parallel Processing*, pp. 407–418, Springer, 2011.

[5] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[6] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[7] J.-Y. Shin, B. Wong, and E. G. Sirer, "Small-world datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, p. 2, ACM, 2011.

[8] L. Minas and B. Ellison, "The problem of power consumption in servers," *Intel Corporation. Dr. Dobb's*, 2009.

[9] F. Wuhib, R. Yanggratoke, and R. Stadler, "Allocating compute and network resources under management objectives in large-scale clouds," *Journal of Network and Systems Management*, pp. 1–26, 2013.

[10] A. Verma, G. Kumar, R. Koller, and A. Sen, "Cosmig: Modeling the impact of reconfiguration in a cloud," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pp. 3–11, IEEE, 2011.

[11] P. Svärd, J. Tordsson, E. Elmroth, S. Walsh, and B. Hudzia, "The noble art of live vm migration -principles and performance of precopy and postcopy migration of demanding workloads,"

[12] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[13] U. Wilensky, "Netlogo preferential attachment model," *Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, Illinois, http://ccl. northwestern. edu/netlogo/models/PreferentialAttachment*, 2005.

[14] D. Breitgand, A. Marashini, and J. Tordsson, "Policy-driven service placement optimization in federated clouds," *IBM Research Division, Tech. Rep*, 2011.

[15] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pp. 103–110, IEEE, 2009.

[16] D. Barbagallo, E. Di Nitto, D. J. Dubois, and R. Mirandola, "A bio-inspired algorithm for energy optimization in a self-organizing data center," in *Self-Organizing Architectures*, pp. 127–151, Springer, 2010.

[17] F. Wuhib, R. Stadler, and M. Spreitzer, "A gossip protocol for dynamic resource management in large cloud environments," *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, pp. 213–225, 2012.

[18] M. Marzolla, O. Babaoglu, and F. Panzieri, "Server consolidation in clouds through gossiping," in *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM),*, pp. 1–6, IEEE, 2011.