CloudOpt: Multi-Goal Optimization of Application Deployments across a Cloud

Jim ZW Li¹, Murray Woodside¹, John Chinneck¹, Marin Litoiu²
{zwi | cmw | chinneck}@sce.carleton.ca; mllitou@yorku.ca
¹Carleton University, Ottawa, Canada; ²York University, Toronto, Canada

Abstract—CloudOpt is a comprehensive approach to find optimal deployments for large service centers and clouds. It uses a combination of bin-packing, mixed integer programming and performance models to make decisions affecting diverse and strongly interacting goals, including satisfaction of different service level agreements for many different applications (including response time, workload, throughput), host memory requirements and availability, license costs and power consumption. It is scalable and extendable to new objectives.

I. INTRODUCTION

Optimization of cost and energy use in large data centers and clouds poses challenges in scale, complexity of constraints, and speed of solution. For maximum benefit, global optimization should be applied to all the applications in the cloud, each with many processes. Quality of service (QoS) contracts must be satisfied, as well as space requirements of the processes and (in this work) limits on the numbers of replicas of some licensed software.

Present optimal deployment approaches have various limitations. Bin-packing [9][14][25] is simple but addresses multiple goals with difficulty and gives only modest optimization quality. Flow network optimization [26] and mixed integer programming (MIP) [22] are effective, but do not address the nonlinearities. Hill climbing [32] and nonlinear optimization [15] have limited search efficiency.

The CloudOpt approach described here is a strategic combination of bin-packing, network flow and MIP, incorporating nonlinear contention effects computed by a performance model. It finds a deployment for a set of service applications, which:

1. minimizes the cost of processing, in money and/or energy,
2. satisfies multiple response time constraints,
3. respects the memory requirements of processes,
4. deploys as many replicas of each process as needed, subject to soft limits on license availability,
5. provides a result in one or two minutes, for 20 applications with over 200 processes.

In principle it can be scaled further, using a simpler model for performance, and is easily extended. In dynamic management this full optimization can be carried out periodically. In other research [17] we study how to increase the persistence of the optimization solutions, for dynamic management.

II. RELATED RESEARCH

Clouds have become part of many software companies’ offerings [1][12][24], and recent research considers small systems [5] and large ones [10]. Allocation of virtual resources shared by two applications in a two tier centre is optimized in [16] using limited look-ahead, with linear models to describe the applications and their use of resources. The optimization is model-based, inspired by control theoretic optimization. In [29] a genetic algorithm optimizes deployments, with a workflow model describing processing demands and queuing delays.

MIP is applied in [7] to optimize VM deployments, considering computation capacity, power, storage and network bandwidth, but not QoS constraints, service allocation and workload balance across VMs. Power optimization by MIP is also part of [22].

Bin-packing has been used to pack execution requirements [9], execution and communications requirements [30], and memory; all of which have been combined in multidimensional bin-packing [6]. Recently bin-packing was extended to optimize task allocations in virtual environments [4]. New packing approaches include heuristics by Karve [13] and Steinder et al. [25] to distribute workloads on virtual nodes, and a combination of max-flow algorithms and heuristics by Tang et al. [26] to manage large-scale resource allocations on IBM Websphere XD.

Hill climbing has also been used to optimize system management. Examples include maximizing throughput in service systems [21], minimizing replicas to satisfy a QoS requirement [23], and seeking optimal QoS components for distributed systems [32]. However, high evaluation costs limit its application to small and medium scale systems.

Flow networks can model large systems but they ignore resource contention effects. Bokhari et al. [3] pioneered the use of flow networks for partitioning workload among nodes. As mentioned above, Tang et al. [26] combined flow algorithms with heuristics. Toktay and Uzsoy [27] maximize resource utilizations with a flow model for shop scheduling, getting almost the same result as MIP, but much faster. However, their approach ignores delay and contention.

Our earlier work combined resource contention models and linear programming based on network flow models (NFM) to find deployments that satisfy QoS conditions...
This was extended to find a change to accommodate one new application [18]. Memory and license constraints were considered using bin-packing alone, with modest success [20].

CloudOpt goes substantially beyond any of these approaches: it combines more goals with explicit contention calculations, using bin-packing for a starting point, and MIP for good quality optimization.

III. OPTIMIZATION ARCHITECTURE

CloudOpt uses the architecture shown in Figure 1 to manage all the applications in a cloud. It uses tracked performance models to evaluate QoS resulting from deployment decisions. An application (upper left) shows a set of service tasks (processes) running in a virtualized environment, with interfaces offering services (operations) to classes of users. Monitored data from the services, virtual machines and hosts drives decisions and also is used to track the parameters of a performance model of each application. Optimization periodically indicates deployment changes which are implemented by the management systems using deployment effector tools to load and initialize VM images on host processors.

The model tracking is described in [33]. When a new application is loaded, an initial performance model is supplied by the application provider, derived either from knowledge of the application or by tracing its behaviour (see e.g. [32]).

![Figure 1. Cloud Management with Optimization](image)

This work focuses on the optimization elements in Figure 1. A management platform might include one or more optimizers, each with an optimization model to describe its part of the problem and a corresponding solver to seek the optimal solution in terms of the optimization model.

IV. PERFORMANCE MODEL FOR SERVICE SYSTEMS

The performance model is a layered queuing model (LQM) [11], with a queue and a server for every software and hardware component. The structure of a LQM expresses the service architecture, as shown in Figure 2. The bold rectangles represent tasks with attached rectangles for the operations they offer (in LQM the operations are called entries, here they are called services). Service $s$ is labeled with its $cpu$ demand $[d_s]$; these demands are tracked by the ModelTracker of Figure 1. Figure 2 does not show the processors.

Requests from one service to another are shown by arrows between services labeled by the mean number of requests (e.g. (3)). Each task $t$ has a multiplicity label $[m_t]$ representing a thread pool size (infinite, if there is no limit). Each user class $c$ is also represented as a task (labeled here $User_c$), with a label $[N_c]$ for the size of the group of users, a service with a think time $[Z_c]$ (the time between receiving a response and making the next request), and arrows showing the service requests made during a single response.

Figure 2 shows an application LQM which acts as a template for the deployed model. The deployed LQM includes host processors, replication of tasks and division of workload between the replicas. Define:

- $RT_c = \text{mean response time of user class } c$, subject to the QoS constraint $RT_c \leq RT_{c,SLA}$
- $f_c = \text{throughput of user requests/sec, subject to a corresponding QoS-related constraint } f_c \geq f_{c,SLA}$ By Little’s result, for $N_c$ users with think time $Z_c$, 

$$f_{c,SLA} = N_c/(RT_{c,SLA} + Z_c) \quad (1)$$
• \( S_{\text{user},c} \) = the set of all services used by class \( c \),
• \( Y_{cs} \) = the mean total requests to service \( s \in S_{\text{user},c} \) made during one user response of class \( c \). \( Y_{cs} \) is found by tracing all paths from \( c \) to \( s \), multiplying the request rates along the paths, and summing the products. Thus in Figure 2, \( Y_{\text{UserClass},e\text{Read}} = 0.913^*2.1+0.0873^*1^*3 \).

If \( Z_c \) is unknown the worst-case value of zero can be taken. If both \( f_{c,\text{SLA}} \) and \( RT_{c,\text{SLA}} \) are specified for a finite \( N_c \), \( f_{c,\text{SLA}} \) is replaced by max\( (f_{c,\text{SLA}}, N_c/\left(RT_{c,\text{SLA}} + Z_c \right) \). \) If \( N_c \) is infinite and the SLA specifies open arrivals and values for both \( f_{c,\text{SLA}} \) and \( RT_{c,\text{SLA}} \) then a closed approximation is used: a large population \( N_c \) (say, 1000) and a large think time \( Z_c \) are chosen which satisfy Eq (1), and they are used in the performance model.

V. THE OPTIMIZATION PROBLEM

The optimization problem allocates flows of work to hosts, with one unit of flow being the rate of work which makes one “standard” host 100% busy. Each host has a capacity \( \Omega_h \) which is the maximum flow it will be allocated, taking into account:
• it may be faster or slower than a “standard” host by some factor,
• it may have multiple cores or CPUs, and its capacity is summed over these,
• it may be constrained to a value giving less than 100% utilization (in this work we used 80%).

Thus, \( \Omega_h = \left( \text{relative CPU speed} \right) \times (m_h = \text{number of cores or CPUs}) \times (\varphi_h = \text{max permitted utilization per core}) \).

The flow at each host is related to the user request rate via flows at tasks and services in the network flow model (NFM) shown in Figure 3, and defined as:
• \( \alpha_{ht} = \text{work rate on host } h \text{ for task } t \),
• \( \beta_{ht} = \text{total work rate by task } t \text{ for service } s \),
• \( \gamma_{hc} = \text{total work rate by service } s \text{ for user class } c \).

The total flow into task \( t \) represents the work rate by all replicas of the task, on the allocated hosts (those with \( \alpha_{ht} > 0 \)); the total flow into service \( s \) represents the total work rate for that service for all user classes; and the flow \( \gamma_{hc} \) is the work rate for class \( c \), i.e. for \( Y_{ct} \) service requests/sec. The cpu demand of one request for service \( s \) is defined for a “standard” host \( d_s \). Then

\[
\gamma_{hc} = (Y_{ct}, d_s) f_c = d_{sc} f_c
\]

Flow constraints are described further below.

The cost of the deployment per unit time is calculated from the host flows, with a fixed part \( C_{fh} \) and a variable part for each host. At host \( h \):

\[
\text{Cost}_h = C_{fh} + \sum \gamma_{ht} \varphi_h + C_s \sum \alpha_{ht}
\]

The fixed part \( C_{fh} \) is incurred if \( \sum \alpha_{ht} > 0 \) (i.e., the host is used by one or more tasks), the variable part is proportional to work rate or equivalently to host utilization. Possible cases:
a) the cost is in dollars per unit time,
b) the cost is for power. Studies have shown that energy use increases with utilization, roughly linearly (e.g. [2],[15]).
c) energy management scales the processor speeds, with lower speeds (and power) for lighter load.

In case (c) we assume that energy cost (say, \( P_h \)) is linear in the host speed setting, which is reduced from its maximum to a fixed fraction “ratio”. Then

\[
P_h = a + b^*\text{ratio}
\]

for some constants \( a \) and \( b \). Further we suppose that ratio is chosen to give a total host flow rate of \( \Omega_h \).

Then at host \( h \),

\[
\text{ratio} = \frac{\text{flow}}{\Omega_h}
\]

and the cost is linear in the flow. Then it can be written in the form of Eq (3).

A. Overview of the Optimization Approach

The optimization algorithm uses an optimization loop made up of 4 main steps and iterates until it gives a near-optimal solution. It has this outline:

Algorithm MIP+C
1. construct the optimization model (MIP based on NFM) for flow constraints,
2. solve the optimization model for an optimal deployment,
3. construct and solve the LQM for the deployment,
4. If the solution does not meet the QoS constraints, add surrogate flows and repeat from (2).

The details of the algorithm are as following.

B. Flow Constraints

The NFM in Figure 3 is a directed graph with nodes representing hosts \( h \), tasks (processes) \( t \), services \( s \), and user classes \( c \) (see [19]). It has an arc for every pair \((h,t)\), for pair \((t,s)\) in which service \( s \) is offered by task \( t \) \((s \in S_{\text{Task}}, t)\), and for pairs \((s,c)\) for which user class \( c \) uses service \( s \) directly or indirectly \((s \in S_{\text{User},c})\). The flows on arcs have labels \([\text{min flow, max flow, cost per unit flow}]\) with default values (where not shown in the figure) of \([0, \infty, 0]\). At host, task and service nodes the sum of flows into the node always equals the sum of flows out. The flow into node \( h \) on the left is its total flow, limited by the capacity \( \Omega_h \). The flow out of node \( c \) on the right of Figure 3 is the rate of user transaction requests, at rate \( f_s/\text{sec} \).

Node \( c \) for user class \( c \) is a special type called a processing node (see e.g. [8]) which converts the execution flow rates \( \gamma_{hc} \) into transaction flow rates \( f_c \), using fixed proportionalities given by Eq (2).
The solution given by the optimization model (NFM) is an optimistic deployment, 

1. allocating host reservations to tasks, 
2. dividing request traffic between multiple task replicas, where applicable, and 
3. minimizing cost.

In [19] a linear program (LP) is derived from the NFM to minimize the sum of flow costs (the host flows on the left in the figure).

C. Including Contention Effects

Contention effects for a given set of flows are estimated by a “deployed LQM” created from the flows. A replica of task \( t \) is deployed on host \( h \) for each \( \alpha_{ht} > 0 \), and the requests to its services are divided appropriately (as described in [19]).

The user response delay from the LQM solution is increased by resource contention due to queuing at the hosts or the server tasks. Contention delays may make the LQM user flows \( f_c \), smaller than the required flows \( f_c \text{,LQN} \), even if the NFM flows meet the requirement \( f_c \geq f_c \text{,LQN} \). As described in [19], this difference is added to the NFM as a surrogate flow at each service, indicated in Figure 3 by the red arrow. The added flow is

\[
\delta_t = \sum_s (\text{difference at node } s) = \sum_s \Delta_t \text{s,SLA} - f_c \text{,LQN}
\]

The surrogate flows force the NFM to reserve additional capacity at the hosts. As in [19], the algorithm for optimal flows and the solution of the corresponding LQM are combined in a fixed point iteration. After iteration \( i \) the accumulated surrogate flow \( \Delta_{i+1} \), at iteration \( j \) is:

\[
\Delta_{i+1}^j = \sum_{j=1}^i \Delta_t^j
\]

which is used in iteration \( i+1 \).

D. Integer Variables and Constraints

Host memory is limited and each replica task has its own VM with memory requirements. Let:

- \( m_t \) = memory requirement of a replica of task \( t \), including its VM,
- \( M_h \) = memory available at host \( h \),
- \( A_{ht} = 1 \) if task \( t \) runs on host \( h \), i.e. if \( \alpha_{ht} > 0 \), else 0.

Then a deployment must satisfy \( \Sigma A_{ht} M_h < M_h \). It may be wasteful in other ways to deploy many replicas of a task. Commercial tasks are licensed, and there may be an additional cost if the number of replicas exceeds the already licensed number. Let:

- \( L_t \) = licenses owned for application task \( t \),
- \( L'_t = \max(0, \Sigma A_{ht} - L_t) \) = additional licenses,
- \( C_{L'_{t}} \) = cost per additional license for task \( t \).

Additional license cost is included as a soft constraint.

Mixed Integer Program (MIP)

To construct a MIP for optimal deployment we use the variables above plus the additional variables:

- \( S_h = 1 \) if \( \alpha_{ht} > 0 \) for some \( t \), else 0,
- \( T(h) = \) set of tasks with \( \alpha_{ht} > 0 \),
- \( \text{Big}C \) = a large positive number.

Optimization Model: MIP Model

Objective:

\[
\text{Minimize } \Sigma A_{ht} S_h C_{fb} + \Sigma A_{ht} C_{g} \alpha_{ht} + \Sigma A_{ht} L'_t C_{Lt} \tag{4a}
\]

over \( A, L', S, \alpha, \beta, \gamma \geq 0 \) with \( A, S \in \{0,1\}, L' \) integer, and subject to constraints:

- capacity for each host \( h \):
  \[
  \Sigma \alpha_{ht} \leq \Omega_h \tag{4b}
  \]
- for each task \( t \):
  \[
  \Sigma h \alpha_{ht} = \Sigma h \beta_{ht} \tag{4c}
  \]
- for each service, add surrogate flows at node \( s \):
  \[
  \Sigma \beta_{ht} = \Sigma \gamma_{hc} + \Sigma \delta_{ht} \tag{4d}
  \]
- for each class \( c \) and service \( s \):
  \[
  \gamma_{hc} = f_c d_{hc} \tag{4e}
  \]
- force \( A_{ht} = 1 \) for arcs with positive flow:
  \[
  \alpha_{ht} \leq A_{ht} \text{, Big}C \tag{4f}
  \]
- memory space at host \( h \):
  \[
  \Sigma f_t m_t < M_h \tag{4g}
  \]
- license constraint:
  \[
  \Sigma h A_{ht} \leq L_t + L'_t \tag{4h}
  \]
- Set \( S_h = 1 \) if any \( A_{ht} \geq 0 \)
  \[
  A_{ht} \leq S_h \text{ over all } t \tag{4i}
  \]
- SLA constraint
  \[
  f_c \geq f_c \text{,SLA} \tag{4j}
  \]
- non-negative flows
  \[
  \alpha, \beta, \gamma \geq 0 \tag{4k}
  \]

(4f) and (4i) are artificial linear constraints to define \( A_{ht} = 1 \) if \( \alpha_{ht} > 0 \), and \( S_h = 1 \) if \( A_{ht} = 1 \) for any \( t \), else 0.

E. Iterative Solution

The trajectory of the iterations is illustrated conceptually in the sketch in Figure 4 for two user classes, showing the user throughputs found by the performance model at each iteration. The solution is expected to be at the intersection of the constraints, since greater throughputs increase the cost. The solution trajectory wanders considerably before reaching the feasible region in the upper right.

The final step is special, using a linearization of the performance model [17]. It usually comes almost exactly to the constraints, from a solution within a few percent of
them. Insisting on a feasible solution first ensures adequate host resources for the final step.

Unfortunately MIP and MIP+C do not scale well enough for large systems. The method is therefore extended with two heuristics: Heuristic Packing (HP) and Heuristic MIP with HMIP.

VI. HEURISTIC PACKING (HP) AND HMIP

HP assigns task flows to the cheapest processors (smallest $C_h$) with available memory, while seeking to reduce the cost on additional license. The resulting quality of optimization is sometimes poor, so HP and MIP are combined in “Heuristic MIP” (HMIP). HMIP applies MIP with the hosts restricted to those selected by HP, reducing the complexity of the MIP.

HP consists of two steps. Step I allocates workloads to the lowest-cost hosts. The memory and license constraints may mean some hosts are under-utilized, so Step II checks for better hosts to replace them. The goal of Step I is to minimize the sum of costs:

$$Cost_h = \max(M_i/M_{h,i}, d/O_h) C_h^* + d C_h$$

in which the first term estimates the share of fixed costs at host $h$. Additional variables (all $\geq 0$) needed for the HP and HMIP algorithms are:

- $\Omega_h^*$ remaining execution demand space of host $h$
- $\Omega_h$ the used capacity of host $h$, $\Omega_h = \Omega_h^* - \Omega_h^*$
- $M_h^*$ remaining memory space of host $h$
- $M_h^i$ used memory space of host $h$, $M_h^i = M_h + M_h^*$
- $d_i^*$ remaining execution demand of task $i$
- $L_i^*$ Remaining licenses available for use
- $L_i^*$ Total number of licenses in use, $L_i^* = \sum_h A_{h,i}$

Heuristic Packing (HP) Algorithm:

Allocate $(t, d, h)$ // allocation function
allocate demand $d$ for task $t$ to host $h$, and adjust the remaining demand $d_i^* h$ and available memory $M_h^*$

Set $\alpha_{h,i} = d_i$; decrement $\Omega_h^*$ by $\alpha_{h,i}$; decrement $d_i^*$ by $\alpha_{h,i}$; decrement $M_h^*$ by $M_i$; increment $L_i^*$ by 1.

Step I

1. Sort the tasks in decreasing order of $d_i^*/L_i^*$, subordinated by decreasing order of $C_{h,i}$.

2. For each task $t$ in order:
   a. Sort the hosts with $M_h^* > M_i$ by flow space $\Omega_h^*$ (largest first), breaking ties by $C_h$ (smallest first) and designate $h(i)$ as the $i$th host in order.
   b. Set $i = 1$ //allocate first to host $h(1)$
   c. if $d_i^* > 0$
      i. Allocate $(t, \min(d_i^*, \Omega_h^*(i)), h(i))$
      ii. If $d_i^* > 0$ and I is not empty, increment $i$ and repeat from Step 2.c, else exit with error “not enough available hosts”
   d. if $d_i^* = 0$, move some replica of task $t$ to a cheaper host (smaller $C_h$) if possible, or move some processing to a new replica on a cheaper host, if a license is available.
   e. if I is not empty, increment $i$.

Step II

For each host $i$ which is not selected in Step I:

1. For each host $j$ which has been selected to host one or more tasks:
   - If $\Omega_h^*(j) > \Omega_h^*(j)$ and $M_h^* > M_h^*(j)$ and $C_{h(i)} > C_{h(j)}$ then:
     Move all tasks from host $j$ to host $i$.

Heuristic Packing with MIP (HMIP) Algorithm:

1. Heuristic Packing (HP) performed as above.
2. Using only the hosts selected in Step 1, construct and solve a MIP as in Section 4.

The feasible solution cost found by HP ($C_{hp}$) is used as an incumbent by the MIP solver to reduce the optimization time. It is added as a constraint:

$$\Sigma h C_h \alpha_{h,i} + \Sigma L_i C_{h,i} + \Sigma h S_h C_h \leq C_{hp}$$

To include estimates of contention, step 2 uses MIP+C in place of MIP, with the iteration between the MIP and the performance model, as described above.

VII. EXAMPLE: USE OF HMIP+C

The first example considers deployment of the application shown in Figure 2, with QoS requirement $RT_{UserClass} \leq 29$ms, and a memory requirement for each task of 1 unit. The hosts for this (and later) experiments are equal numbers of five types with the relative speeds, memory and cost factors shown in Table 3. The ratios between $C_h$ and $C_{hp}$ are based on the experimental results given in [2][15], ranging from 0.15 to 1. The number of hosts was adjusted to make the average host utilization about 0.7 in the solution.
Table 1. Host Information

<table>
<thead>
<tr>
<th>Type</th>
<th>Speed</th>
<th>Memory</th>
<th>(C_v)</th>
<th>(C_f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.8</td>
<td>2</td>
<td>0.5</td>
<td>0.6</td>
</tr>
<tr>
<td>B</td>
<td>2.4</td>
<td>4</td>
<td>0.45</td>
<td>0.81</td>
</tr>
<tr>
<td>C</td>
<td>2.8</td>
<td>8</td>
<td>0.4</td>
<td>1.12</td>
</tr>
<tr>
<td>D</td>
<td>3.2</td>
<td>12</td>
<td>0.35</td>
<td>1.4</td>
</tr>
<tr>
<td>E</td>
<td>3.6</td>
<td>16</td>
<td>0.3</td>
<td>1.62</td>
</tr>
</tbody>
</table>

Table 2. Solution Properties by Iteration

<table>
<thead>
<tr>
<th>Iteration</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Time (Goal: 29ms)</td>
<td>34.80 ms</td>
<td>32.47 ms</td>
<td>27.21 ms</td>
</tr>
<tr>
<td>Variable Cost</td>
<td>4.93</td>
<td>4.83</td>
<td>5.07</td>
</tr>
<tr>
<td>Fixed Cost</td>
<td>6.58</td>
<td>6.86</td>
<td>6.88</td>
</tr>
<tr>
<td>Total Cost</td>
<td>11.51</td>
<td>11.69</td>
<td>11.93</td>
</tr>
<tr>
<td>Solution Time</td>
<td>0.73 s</td>
<td>0.55 s</td>
<td>0.59 s</td>
</tr>
</tbody>
</table>

Table 2 shows that a feasible solution with \(RT = 27.21\)ms was found in three iterations, with 13 replicas deployed on 6 hosts. Table 3 shows that the resource usage is balanced across the hosts. The solution takes 1.87 sec.

Table 3 CPU and Memory Utilization in Hosts

Table 4. Comparison of HP alone, MIP alone, and HMIP (Without Iterating to Include Contention)

<table>
<thead>
<tr>
<th>Stress rate</th>
<th>High (0.7±0.05)</th>
<th>Medium (0.5±0.05)</th>
<th>Low (0.25±0.05)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HP MIP HMIP</td>
<td>HP MIP HMIP</td>
<td>HP MIP HMIP</td>
</tr>
<tr>
<td>10 app, 100 tasks</td>
<td>23 hosts</td>
<td>32 hosts</td>
<td>57 hosts</td>
</tr>
<tr>
<td>objective</td>
<td>34.38</td>
<td>32.82</td>
<td>33.14</td>
</tr>
<tr>
<td>Solution time (sec)</td>
<td>0.016</td>
<td>6.969</td>
<td>0.422</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>3313</td>
<td>2326</td>
</tr>
<tr>
<td>20 app, 200 tasks</td>
<td>33 hosts</td>
<td>46 hosts</td>
<td>86 hosts</td>
</tr>
<tr>
<td>objective</td>
<td>56.52</td>
<td>52</td>
<td>51.99</td>
</tr>
<tr>
<td>Solution time (sec)</td>
<td>0.031</td>
<td>4.281</td>
<td>3.203</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>9413</td>
<td>6884</td>
</tr>
<tr>
<td>30 app, 300 tasks</td>
<td>53 hosts</td>
<td>75 hosts</td>
<td>139 hosts</td>
</tr>
<tr>
<td>objective</td>
<td>106.41</td>
<td>84.11</td>
<td>84.06</td>
</tr>
<tr>
<td>Solution time (sec)</td>
<td>0.046</td>
<td>6.266</td>
<td>7.625</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>22523</td>
<td>16629</td>
</tr>
<tr>
<td>40 app, 400 tasks</td>
<td>64 hosts</td>
<td>89 hosts</td>
<td>169 hosts</td>
</tr>
<tr>
<td>objective</td>
<td>107.12</td>
<td>101.71</td>
<td>102.66</td>
</tr>
<tr>
<td>Solution time (sec)</td>
<td>0.063</td>
<td>100.81</td>
<td>13.688</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>36184</td>
<td>26647</td>
</tr>
<tr>
<td>50 app, 500 tasks</td>
<td>81 hosts</td>
<td>113 hosts</td>
<td>216 hosts</td>
</tr>
<tr>
<td>objective</td>
<td>134.86</td>
<td>130.91</td>
<td>130.77</td>
</tr>
<tr>
<td>Solution time (sec)</td>
<td>0.062</td>
<td>Time out</td>
<td>5.437</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>57131</td>
<td>41709</td>
</tr>
</tbody>
</table>

Table 4 shows that a feasible solution was found in three iterations, with 13 replicas deployed on 6 hosts. Table 3 shows that the resource usage is balanced across the hosts. The solution takes 1.87 sec.

The optimization creates replicas for tasks, balances workloads, and optimizes allocations, achieving the required performance with an economical solution.

VIII. EVALUATION OF HP, HMIP and MIP

To focus on the comparison of MIP with both HP and HMIP, experiments were first done without the iterative contention calculations, then with them.

Cases had 10 to 50 applications, each based on the template in Figure 2, with randomly chosen demands for CPU, memory, and license availability for each task. The size of the host pool (made up of the five types A – E in Table 1), was adjusted to give levels of “stress” of about 0.25, 0.5, and 0.7:

\[ \text{stress} = \text{flow required for SLA/total capacity} = \frac{\sum f_i \cdot \Omega_i}{\Omega} \]

At each stress value there is about the same number of hosts of each type, and the same randomly constructed applications were used.
Table 4 compares the cost, solution time and number of MIP variables (continuous and discrete). The smallest value for each is in boldface. It shows:

- MIP usually gives the smallest cost, but it takes longer than HMIP and runs out of memory or time.
- HP occasionally gives considerably higher cost.
- In under 10 sec, HMIP found a solution with 40 applications on 169 hosts, under high “stress”.
- HMIP gives almost as low cost results as MIP.
- HMIP time was smallest for high stress models.
- The costs, hosts and replicas are similar across low and high-stress problems. Excess resources thus are not an advantage.
- Low stress cases have larger MIPs than high stress, but have more reduction in size for HMIP.
- MIP problems with less than 10,000 variables were solved by CPLEX in less than 10 seconds.

Overall, HMIP is very satisfactory. Cost is within 2% of pure MIP and it is always faster, up to an order of magnitude (and increasing) on large problems.

A. Comparison including Contention

The same set of applications with 1, 5, 10 and 20 applications were optimized again with the iterative contention calculation included. The results are shown in Table 5.

If we adopt one minute as a maximum practical optimization time, these cases are practical up 10 applications. This is quite good, since changes to deployment take on the order of minutes even for just a few machines. Since about 60% of the iteration time is devoted to the LQM, this would be improved if the LQM solution time could be reduced.

These experimental results show that CloudOpt can deploy between 100 and 200 heterogeneous tasks in a reasonable time. The performance model is the limiting factor.

### Table 5. Evaluation of HP+C, Pure MIP+C, and HMIP+C

<table>
<thead>
<tr>
<th>Stress rate</th>
<th>High (0.7±0.05)</th>
<th>Medium (0.5±0.05)</th>
<th>Low (0.25±0.05)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 app</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of iterations</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Time for optimization (sec)</td>
<td><strong>0.01</strong></td>
<td>0.109</td>
<td>0.109</td>
</tr>
<tr>
<td>Time for LQNS (sec)</td>
<td><strong>0.094</strong></td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>Total energy and license Cost</td>
<td>3.82</td>
<td><strong>3.23</strong></td>
<td><strong>3.23</strong></td>
</tr>
<tr>
<td># of replicas (task)/ # of hosts</td>
<td>7/2</td>
<td>7/1</td>
<td>7/1</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>5 app</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of iterations</td>
<td>7</td>
<td>18</td>
<td>21</td>
</tr>
<tr>
<td>Time for optimization (sec)</td>
<td><strong>0.012</strong></td>
<td>1.36</td>
<td>1.079</td>
</tr>
<tr>
<td>Time for LQNS (sec)</td>
<td><strong>2.078</strong></td>
<td>4.829</td>
<td>4.875</td>
</tr>
<tr>
<td># of replicas (task)/ # of hosts</td>
<td>58/6</td>
<td>38/5</td>
<td>38/5</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>461</td>
<td>390</td>
</tr>
<tr>
<td>10 app</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of iterations</td>
<td>24</td>
<td>11</td>
<td>24</td>
</tr>
<tr>
<td>Time for optimization (sec)</td>
<td><strong>0.093</strong></td>
<td>6.078</td>
<td>7.488</td>
</tr>
<tr>
<td>Time for LQNS (sec)</td>
<td>32.625</td>
<td><strong>10.954</strong></td>
<td>26.92</td>
</tr>
<tr>
<td>Total energy and license Cost</td>
<td>24.14</td>
<td>21.43</td>
<td><strong>20.63</strong></td>
</tr>
<tr>
<td># of replicas (task)/ # of hosts</td>
<td>129/15</td>
<td>75/11</td>
<td>77/10</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>2185</td>
<td>1480</td>
</tr>
<tr>
<td>20 app</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># of iterations</td>
<td>23</td>
<td>42</td>
<td>22</td>
</tr>
<tr>
<td>Time for optimization (sec)</td>
<td><strong>0.079</strong></td>
<td>197.32</td>
<td>61.62</td>
</tr>
<tr>
<td>Time for LQNS (sec)</td>
<td>202.86</td>
<td>273.66</td>
<td><strong>121.86</strong></td>
</tr>
<tr>
<td>Total energy and license Cost</td>
<td>47.79</td>
<td>42.32</td>
<td><strong>41.42</strong></td>
</tr>
<tr>
<td># of replicas (task)/ # of hosts</td>
<td>244/29</td>
<td>155/21</td>
<td>153/20</td>
</tr>
<tr>
<td># of variables in MIP</td>
<td>-</td>
<td>8289</td>
<td>6884</td>
</tr>
</tbody>
</table>
IX. COMPARISON WITH OTHER APPROACHES

Many existing systems use packing approaches to handle task deployments. HMIP was compared to the Power-minimizing Placement Algorithm (mPP) [2] and a simple greedy approach SGD described as follows:

**Simple Greedy Deployment (SGD):**

T: a collection of tasks to be deployed. 
I: the number of hosts
1. Sort hosts in increasing order of maximum cost $C_{lh} + \Omega_h C_h$, and set $i = 1$
2. For each $i$ in $T$
   2.1. Set $d_i = d_i$
   2.2. If $i \leq I$ then:
      2.2.1. **Allocate** $(t, \min(d_i, \Omega h), h(i))$
      2.2.2. If $d_i > 0$ increment $i$ and repeat from 2.
   2.3. Else (i.e. $i > I$) return error “out of hosts”
   2.4. Next $t$, and increment $i$

Table 6 compares the effectiveness of HMIP, SGD and mPP for the single application used in Section 6.

SGD and mPP do not account for performance, so it is not surprising that their solutions violate the response time constraint. In SGD one task per host increases the number of hosts and reduces host utilization. This results in higher costs and more hosts than HMIP. mPP allows resource sharing, so it uses fewer hosts, but cannot guarantee optimality because of the limitation of the packing strategy. Moreover, both SGD and mPP do not consider the memory and license constraints, but this limitation is not critical in this simple example.

**Table 6. Comparison of HMIP, SGD and mPP**

<table>
<thead>
<tr>
<th></th>
<th>HMIP</th>
<th>SGD</th>
<th>mPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Time (Goal: 29ms)</td>
<td>27.21ms</td>
<td>34.10ms</td>
<td>41.78ms</td>
</tr>
<tr>
<td>Hosts Used</td>
<td>6</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>Total Cost</td>
<td>11.95</td>
<td>14.28</td>
<td>12.34</td>
</tr>
<tr>
<td>Variable Cost</td>
<td>5.05</td>
<td>6.48</td>
<td>6.34</td>
</tr>
<tr>
<td>Fixed Cost</td>
<td>6.88</td>
<td>7.80</td>
<td>5.99</td>
</tr>
<tr>
<td>Average CPU Utilization</td>
<td>0.76</td>
<td>0.55</td>
<td>0.70</td>
</tr>
</tbody>
</table>

X. CONCLUSIONS

CloudOpt has been shown to be an effective and scalable algorithm for optimizing deployments in clouds. It has been shown to be practical (in the sense of providing a solution in under two minutes) for deploying applications totaling 100-200 processes plus some replicas. It honors constraints on user QoS and process memory, and can optimize energy use or financial cost. License cost is also included, showing flexibility in addressing additional concerns.

HMIP is more effective and more efficient when applied to a smaller host pool (which is big enough to carry the workload with an average utilization no more than 70%).

CloudOpt can be made still more scalable with a more efficient performance model solver. It has also been extended to deal with installation of new applications and with other dynamic changes. An extended description of CloudOpt is given in [17].

Acknowledgements

This research was supported by a studentship from the Toronto IBM Centre for Advanced Studies.

REFERENCES


