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

Jim ZW Li[1], Murray Woodside[1], John Chinneck[1], Marin Litoiu[2]
{zwli | cmw | chinneck}@sce.carleton.ca,; mlitoiu@yorku.ca
[1]Carleton University, Ottawa, Canada; [2]York 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 (involving 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 through-put 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 (NFMs) to find deployments that satisfy QoS conditions

[19]. 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]).

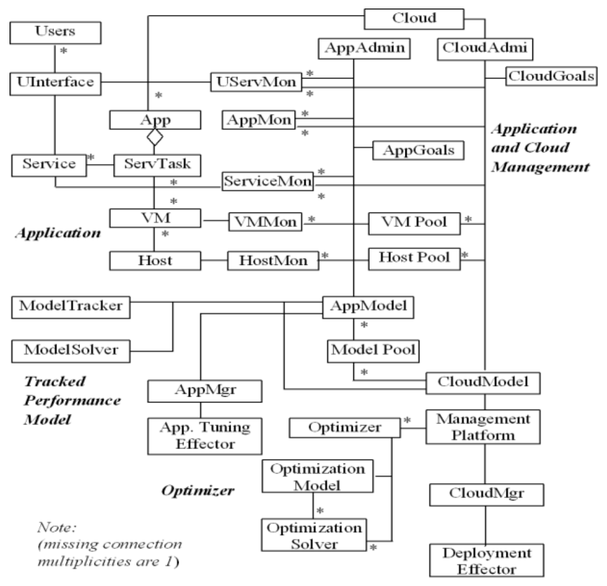

Figure 1. Cloud Management with Optimization

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

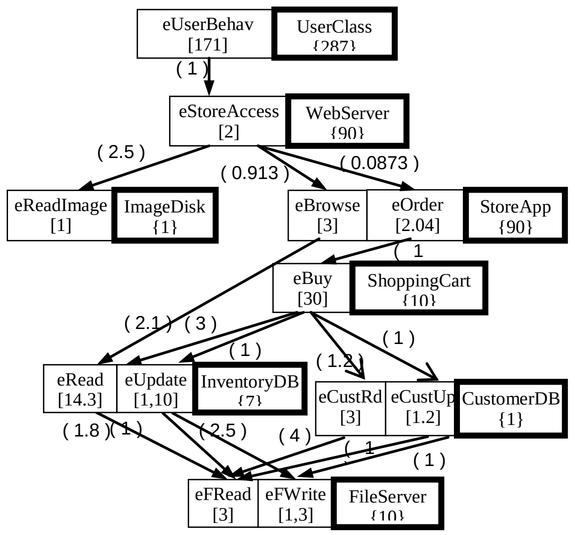

Figure 2. A Service System as an Application LQM

The performance model is a layered queueing 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 = mean response time of user class c, subject to the QoS constraint $RT_c \leq RT_{c,SLA}$
- f_c = 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,

$$f_{c,SLA} = N_c/(RT_{c,SLA} + Z_c) \qquad (1)$$

- $\boldsymbol{S}_{user,c}$ = the set of all services used by class c,
- Y_{cs} = the mean total requests to service $s \in \boldsymbol{S}_{user,c}$ 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, $Y_{UserClass,eRead}$ = 0.913*2.1+0.0873*1*3.

If Z_c is unknown the worst-case value of zero can be taken. If both $f_{c,SLA}$ and $RT_{c,SLA}$ are specified for a finite N_c, $f_{c,SLA}$ is replaced by $\max(f_{c,SLA}, N_c/(RT_{c,SLA} + Z_c)$. If N_c is infinite and the SLA specifies open arrivals and values for both $f_{c,SLA}$ and $RT_{c,SLA}$ 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 Ω_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, Ω_h = (relative CPU speed)×(m_h = number of cores or CPUs)×(φ_h = 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:

α_{ht} = work rate on host h for task t ,

β_{ts} = total work rate by task t for service s,

γ_{sc} = total work rate by service s 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 α_{ht} >0); the total flow into service s represents the total work rate for that service for all user classes; and the flow γ_{sc} is the work rate for class c, i.e. for $Y_{cs}f_c$ service requests/sec. The cpu demand of one request for service s is defined for a "standard" host as d_s . Then

$$\gamma_{sc} = (Y_{cs}\, d_s)\, f_c = d_{sc}\, f_c \qquad (2)$$

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:

$$\text{Cost}_h = C_{fh\,:\,\Sigma_t\alpha_{ht} > 0} + C_h\, \Sigma_t\, \alpha_{ht} \qquad (3)$$

The fixed part C_{fh} is incurred if $\Sigma_t\alpha_{ht} > 0$ (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 $P_h = a + b{*}ratio$ for some constants a and b. Further we suppose that *ratio* is chosen to give a total host flow rate of Ω_h. Then at host h,

$$ratio = flow/\Omega_h\ ,$$

$$P_h = a + b{*}\ flow/\Omega_h$$

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 ($s \in \boldsymbol{S}_{Task,t}$), and for pairs ($s$,$c$) for which user class c uses service s directly or indirectly ($s \in \boldsymbol{S}_{User,c}$). The flows on arcs have labels [min flow, max flow, cost per unit flow] with default values (where not shown in the figure) of [0, ∞, 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 Ω_h. The flow out of node c on the right of Figure 3 is the rate of user transaction requests, at rate f_c/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 γ_{sc} into transaction flow rates f_c, using fixed proportionalities given by Eq. (2).

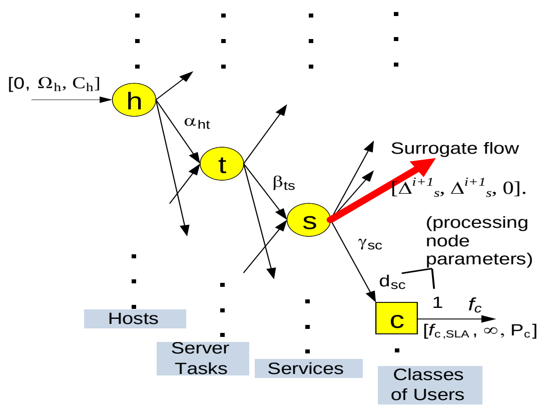

Figure 3. Network Flow Model

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 queueing at the hosts or the server tasks. Contention delays may make the LQM user flows $f_{c,LQN}$ smaller than the required flows $f_{c,SLA}$, even if the NFM flows meet the requirement $f_c \geq f_{c,SLA}$. 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

$$\delta_s = \Sigma_c\, (\textit{difference at node } s)$$
$$= \Sigma_c\, d_{sc}\, (f_{c,SLA} - f_{c,LQN})$$

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 (δ^j_s at iteration j) is:

$$\Delta^{i+1}_s = \Sigma^i_{j=1}\, \delta^j_s$$

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 α_{ht} >0, else 0.

Then a deployment must satisfy $\Sigma_t\, A_{ht} m_t < M_t$.

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, $\Sigma_t\, A_{ht} - L_t$) = additional licenses,
- C_{Lt} = 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 α_{ht}>0 for some t, else 0,
- $\mathbf{T}(h)$ = set of tasks with α_{ht} >0,
- $BigC$ = a large positive number.

Optimization Model: MIP Model

Objective:

$$\text{Minimize } \Sigma_h\, S_h C_{fh} + \Sigma_{ht}\, C_h \alpha_{ht} + \Sigma_t\, L_t'\, C_{Lt} \qquad (4a)$$

over A, L', S, α, β, $\gamma \geq 0$ with A, S in {0,1}, L' integer, and subject to constraints:

○ capacity for each host h:	$\Sigma_t \alpha_{ht} \leq \Omega_h$	(4b)
○ for each task t:	$\Sigma_h \alpha_{ht} = \Sigma_s \beta_{ts}$	(4c)
○ for each service, add surrogate flows at node s:	$\Sigma_t \beta_{ts} = \Sigma_c \gamma_{sc} + \Sigma_S \Delta^i_s$	(4d)
○ for each class c and service s:	$\gamma_{sc} = f_c d_{sc}$	(4e)
○ force A_{ht}=1 for arcs with positive flow:	$\alpha_{ht} \leq A_{ht}.\ BigC$	(4f)
○ memory space at host h:	$\Sigma_t A_{ht} m_t < M_h$	(4g)
○ license constraint:	$\Sigma_h\, A_{ht} \leq L_t + L_t'$	(4h)
○ Set S_h = 1 if any $A_{ht} \geq 0$	$A_{ht} \leq S_h$ over all t	(4i)
○ SLA constraint	$f_c \geq f_{c,SLA}$	(4j)
○ non-negative flows	$\alpha, \beta, \gamma, f \geq 0$	(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.

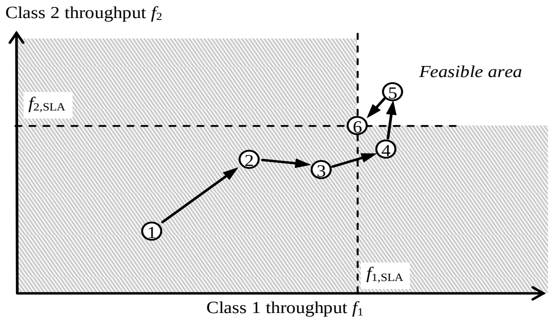

Figure 4. Notional Progress of Iterations of MIP+C

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 Packing with MIP (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:

$$Cost_h = \max(M_t/M_{\mathrm{h}},\ d/\Omega_h\ )\ C_{fh} + \ d\cdot C_h \qquad (5)$$

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

Ω^+_h	remaining execution demand space of host h
Ω^*_h	the used capacity of host h, $\Omega^*_h = \Omega_h - \Omega^+_h$
M^+_h	remaining memory space of host h
M^*_h	used memory space of host h, $M^*_h = M_h - M^+_h$
d^+_t	remaining execution demand of task t
L^+_t	Remaining licenses available for use
L^*_t	Total number of licenses in use, $L^*_t = \Sigma_h A_{ht}$

Heuristic Packing (HP) Algorithm:

```
Allocate (t, d, h )   // allocation function
//allocate demand d for task t to host h, and adjust the remaining demand $d^+_{lt}$ and available memory $M^+_h$
    Set $\alpha_{ht} = d$; decrement $\Omega^+_h$ by $\alpha_{ht}$; decrement $d^+_t$ by $\alpha_{ht}$; decrement $M^+_h$ by $M_t$; increment $L^*_t$ by 1.

Step I
1   Sort the tasks in decreasing order of $d^+_t / L^+_t$, subordered by decreasing order of $C_{Lt}$.
    //This gives priority to tasks with the most demand and fewest licenses.
2   For each task t in order:
    a.  Sort the hosts with $M^+_h > M_t$ by flow space $\Omega^+_h$ (largest first), breaking ties by $C_h$ (smallest first) and designate $h(i)$ as the ith host in order. Define the sorted hosts as set I.
    b.  Set $i = 1$    //allocate first to host $h(1)$)
    c.  if $d^+_t > 0$
        i    Allocate (t, min($d^+_t$, $\Omega^+_{h(i)}$ ), $h(i)$)
        ii   If $d^+_t > 0$ and I is not empty, increment i and repeat from Step 2.c, else exit with error "not enough available hosts"
    d.  if $d^+_t = 0$, 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:
        For each host j which has been selected to host one or more tasks:
            If $\Omega^+_{h(i)} > \Omega^*_{h(j)}$ and $M^+_{h(i)} > M^*_{h(j)}$ and $C_{fh(i)}+C_{h(i)}\Omega^*_{h(j)} < C_{fh(j)}+C_{h(j)}\ \Omega^*_{h(j)}$ then:
                Move all tasks from host j to host i.
                //swap to a cheaper host if feasible
```

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:

$$\Sigma_{ht}\ C_h\alpha_{ht}\ + \Sigma_t\ L_t'C_{Lt} + \Sigma_h\ S_hC_{fh} \leq \ C_{HP} \qquad (6)$$

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_{fh} 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

Type	Speed	Memory	Variable cost coefficient (C_h)	Fixed cost coefficient (C_{fh})
A	1.8	2	0.5	0.6
B	2.4	4	0.45	0.81
C	2.8	8	0.4	1.12
D	3.2	12	0.35	1.4
E	3.6	16	0.3	1.62

CPLEX [13] is used as the MIP solver with aggressive probing and strong branching, stopping when it has found a feasible solution within one percent of the current bound, or at 350 seconds. No additional licenses were required in this case.

Table 2 shows that a feasible solution with *RT* = 27.21ms 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 2. Solution Properties by Iteration

Iteration	**1**	**2**	**3**
Response Time (Goal: 29ms)	34.80 ms	32.47 ms	27.21 ms
Variable Cost	4.93	4.83	5.07
Fixed Cost	6.58	6.86	6.88
Total Cost	11.51	11.69	11.93
Solution Time	0.73 s	0.55 s	0.59 s

Table 3 CPU and Memory Utilization in Hosts

Host Name	**CPU Utilization**	**Memory Utilization**
pHostB_505	76.19%	29.45%
pHostB_510	76.15%	25.95%
pHostC_895	78.21%	18.48%
pHostC_899	70.91%	13.79%
pHostD_307	78.38%	13.35%
pHostE_774	78.44%	5.94%

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:

$$\text{stress} = \text{required flow for SLA/total capacity}$$
$$= \Sigma_c f_{c,\text{SLA}} \Sigma_s d_{sc} / \Sigma_h \Omega_h$$

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

Stress rate	**High (0.7±0.05)**			**Medium (0.5±0.05)**			**Low (0.25±0.05)**		
	HP	MIP	HMIP	HP	MIP	HMIP	HP	MIP	HMIP
10 app, 100 tasks	**23 hosts**			**32 hosts**			**57 hosts**		
objective	34.38	**32.82**	33.14	46.79	**32.65**	32.85	35.2	**32.2**	**32.2**
Solution time (sec)	**0.016**	6.969	0.422	**0.015**	6.344	1.734	**0.031**	0.891	0.422
# of variables in MIP	-	3313	2326	-	4582	2326	-	8107	2044
20 app, 200 tasks	**33 hosts**			**46 hosts**			**86 hosts**		
objective	56.52	52	**51.99**	68.69	**51.3**	51.3	52.35	50.98	**50.88**
Solution time (sec)	**0.031**	4.281	3.203	**0.031**	25.953	7.735	**0.062**	68.66	2.562
# of variables in MIP	-	9413	6884	-	13066	7165	-	24306	6322
30 app, 300 tasks	**53 hosts**			**75 hosts**			**139 hosts**		
objective	106.41	84.11	**84.06**	86.1	83.4	**83.3**	84.51	**81.95**	**81.95**
Solution time (sec)	**0.046**	6.266	7.625	**0.078**	51.687	9.672	**0.109**	21.42	4.328
# of variables in MIP	-	22523	16629	-	31785	16629	-	58729	14945
40 app, 400 tasks	**64 hosts**			**89 hosts**			**169 hosts**		
objective	107.12	**101.71**	102.66	104.72	**100.7**	**100.7**	102.13	Out of	**101.0**
Solution time (sec)	**0.063**	100.81	13.688	**0.094**	165.83	27.093	**0.141**	Memory	9.047
# of variables in MIP	-	36184	26647	-	50209	26647	-	95205	23842
50 app, 500 tasks	**81 hosts**			**113 hosts**			**216 hosts**		
objective	134.86	130.91	**130.77**	135.2	**130.28**	130.3	130.58	Out of	**128.6**
Solution time (sec)	**0.062**	Time out	54.547	**0.094**	Time out	57.3	**0.187**	Memory	19.359
# of variables in MIP	-	57131	41709	-	79563	42410	-	152103	37503

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

Stress rate	**High (0.7±0.05)**			**Medium (0.5±0.05)**			**Low (0.25±0.05)**		
1 app	**HP**	**MIP+C**	**HMIP+C**	**HP**	**MIP+C**	**HMIP+C**	**HP**	**MIP+C**	**HMIP+C**
# of iterations	1	1	1	1	1	1	1	1	1
Time for optimization (sec)	**0.01**	0.109	0.109	**0.01**	0.109	0.109	**0.01**	0.109	0.109
Time for LQNS (sec)	**0.094**	0.11	0.11	**0.109**	0.11	0.11	0.109	**0.094**	0.109
Total energy and license Cost	3.82	**2.32**	**2.32**	3.82	**2.32**	**2.32**	3.85	**2.22**	**2.22**
# of replicas (task)/ # of hosts	7/2	7/1	7/1	7/2	7/1	7/1	7/3	7/1	7/1
# of variables in MIP	-	37	37	-	37	37	-	82	52
5 app									
# of iterations	7	18	21	1	8	8	8	3	9
Time for optimization (sec)	**0.012**	1.36	1.079	**0.016**	1.11	0.765	**0.031**	1.03	0.953
Time for LQNS (sec)	**2.078**	4.829	4.875	**0.313**	1.907	2.031	2.811	**0.789**	2.155
Total energy and license Cost	10.25	9.765	**9.532**	9.82	9.52	**9.37**	13.13	**9.30**	9.41
# of replicas (task)/ # of hosts	58/6	38/5	38/5	37/5	37/5	38/5	48/10	37/4	38/5
# of variables in MIP	-	461	390	-	674	390	-	1242	674
10 app									
# of iterations	24	11	24	23	21	25	19	9	20
Time for optimization (sec)	**0.093**	6.078	7.485	**0.015**	17.859	11.174	**0.109**	28.141	6.425
Time for LQNS (sec)	32.625	**10.954**	26.92	32.093	**20.70**	24.886	31.031	**11.189**	25.890
Total energy and license Cost	24.14	21.43	**20.63**	29.71	**20.19**	20.4	24.77	**19.49**	20.08
# of replicas (task)/ # of hosts	129/15	75/11	77/10	179/20	76/10	78/10	112/15	76/8	77/9
# of variables in MIP	-	2185	1480	-	2890	1762	-	5710	2044
20 app									
# of iterations	23	42	22	32		28	17		21
Time for optimization (sec)	**0.079**	197.32	61.62	**0.156**		64.32	**0.295**		29.766
Time for LQNS (sec)	202.86	273.66	**121.86**	304.67	Out of Memory	**271.4**	**178.13**	Out of Memory	191.74
Total energy and license Cost	47.79	42.32	**41.42**	58.26		**41.75**	54.63		39.29
# of replicas (task)/ # of hosts	244/29	155/21	153/20	294/38		158/21	250/35		152/16
# of variables in MIP	-	8289	6884	-		8008	-		5479

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 $C_{fh}+\Omega_h C_h$, and set $i = 1$
2. For each $t$ in T
   2.1. Set $d^+_t = d_t$
   2.2. If $i \le$ I then:
      2.2.1. Allocate ($t$, min($d^+_t$, $\Omega_{h(i)}$), $h(i)$)
      2.2.2. If $d^+_t > 0$ 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

	HMIP	**SGD**	**mPP**
Response Time (Goal: 29ms)	**27.21ms**	34.10ms	41.78ms
Hosts Used	**6**	13	10
Total Cost	**11.95**	14.28	12.34
Variable Cost	**5.05**	6.48	6.34
Fixed Cost	**6.88**	7.80	5.99
Average CPU Utilization	**0.76**	0.55	0.70

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

[1] Amazon Web Services, http://aws.amazon.com/, March. 2011.
[2] A. Verma, P. Ahuja, A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems". In Proc 9th ACM/IFIP/ USENIX Intl Conf on Middleware (Middleware '08), Springer-Verlag, pp 243-264.
[3] S. H. Bokhari, "Partitioning Problems in Parallel, Pipeline, and Distributed Computing". IEEE Trans. Comput. 37, 1 48-57, Jan. 1988.
[4] D. Carrera,, "Adaptive Execution Environments for Application Servers." PhD dissertation, Universitat Politècnica de Catalunya , Barcelona, 2008
[5] CERAS project, https://www.cs.uwaterloo.ca/twiki /view/CERAS, Jan 15, 2010.
[6] S. Chao, J.W. Chinneck, R.A. Goubran, "Assigning Service Requests in Voice-over-Internet Gateway Multiprocessors" Computers and Operations Research, v. 31, pp 2419-2437, 2004.
[7] S. Chaisiri, B. Lee, D. Niyato, "Optimal Virtual Machine Placement Across Multiple Cloud Providers", Proc Asia-Pacific Services Computing Conf, (APSCC 2009), 2009, pp 103 – 110.
[8] J.W. Chinneck, "Processing Network Models of Energy/Environment Systems", Computers and Industrial Engineering, vol. 28, no. 1, pp. 179-189. 1995.
[9] E.G Coffman, M.R. Garey, D.S. Johnson, "An Application Of Bin-Packing to Multiprocessor Scheduling", SIAM J. Computing, vol. 7, pp. 1-17, Feb. 1978.
[10] EU's FP7 RESERVOIR project, http://www.reservoir-fp7.eu/, Jan 15, 2010..
[11] G. Franks, T. Al-Omari, M. Woodside, O. Das, S. Derisavi, "Enhanced Modeling and Solution of Layered Queueing Networks", IEEE Trans. on Software Eng. Aug. 2008.
[12] IBM Corp., "From Cloud Computing to the New Enterprise Data Center", 2008.
[13] IBM Corp, Users Manual for CPLEX, 2009.
[14] A. Karve, T. Kimbrel, Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., and Tantawi, A., "Dynamic Placement for Clustered Web Applications". Proc. 15th Int Conf. on World Wide Web (WWW '06), pp 595-604. Edinburgh, 2006.
[15] D. Kusic,; J.O. Kephart, J.E. Hanson, N. Kandasamy, G. Jiang; "Power and Performance Management of Virtualized Computing Environments Via Lookahead Control," ICAC'08, pp 3 – 12, June 2008.
[16] A. Lenk , M. Klems , J. Nimis , S. Tai , T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud Landscape", Proc. 2009 ICSE Workshop on Software Eng Challenges of Cloud Computing, p.23-31, May 2009.
[17] J. Li, "Fast Optimization for Scalable Application Deployments in Large Service Centers" , PhD thesis, Carleton University, Ottawa, Feb, 2011.
[18] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai, "Performance Model Driven QoS Guarantees and Optimization in Clouds", Proc. Workshop on Software Engineering Challenges in Cloud Computing, ICSE 2009, Vancouver, May 2009.

[19] J. Li, J. Chinneck, M. Woodside, M. Litoiu, "Fast Scalable Optimization to Configure Service Systems having Cost and Quality of Service Constraints," Proc. Int. Conf. on Autonomic Computing (ICAC09), Barcelona, June 2009.

[20] J. Li, J. Chinneck, M. Woodside, M. Litoiu, "Deployment of Services in a Cloud Subject to Memory and License Constraints", Proc the 2nd Int. Conf. on Cloud Computing, IEEE, Bangalore, India, Sept 2009.

[21] M. Litoiu, J. Rolia, and G. Serazzi, "Designing Process Replication and Activation: A Quantitative Approach", IEEE Transactions on Software Engineering, vol. 26, no. 12, pp. 1168-1178, Dec. 2000.

[22] L. Bertini, J.C.B. Leite, D. Moss, :Power optimization for dynamic configuration in heterogeneous web server clusters", J. Syst. Softw. 83, 4 (April 2010), 585-598.

[23] D. A. Menascé, E. Casalicchio, V. Dubey, "A Heuristic Approach to Optimal Service Selection in Service Oriented Architectures", Proc 7th Int. Workshop on Software and Performance, WOSP '08, pp. 13-24, New York, 2008.

[24] Salesforce Cloud Computing Platform, http://www.salesforce.com/platform/, Jan 15, 2011.

[25] M. Steinder, I. Whalley, D. Carrera, and D. Chess, "Server Virtualization in Autonomic Management of Heterogeneous Workloads". Proc. Integrated Management (IM 2007), Munich, May 2007.

[26] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A Scalable Application Placement Controller for Enterprise Data Centers", Proc. 16th Int. Conf. on the World Wide Web (WWW '07). pp 331-340, 2007.

[27] Toktay and Uzsoy, "A Capacity Allocation Problem with Integer Side Constraints". European Journal of Operational Research, v109 pp 170-182, 1998.

[28] Computing and Networking (e-Energy '10), ACM, 225-233.

[29] H. Wada, J. Suzuki, K. Oba, "Queuing Theoretic and Evolutionary Deployment Optimization with Probabilistic SLAs for Service Oriented Clouds", 2009 World Conf. on Services, pp 661–669, July 2009.

[30] C.M. Woodside, G.G. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems", IEEE Trans. on Parallel and Distributed Systems, v. 4, n. 2, pp. 164-174, 1993.

[31] X. Dutreilh, A. Moreau, J. Malenfant, N. Riviere, and I. Truck, "From Data Center Resource Allocation to Control Theory and Back." Proc 3rd Int Conf on Cloud Computing (CLOUD '10). Washington, pp 410-417.

[32] T. Zheng, "Model-based Dynamic Resource Management for Multi Tier Information Systems", PhD thesis, Carleton University, Ottawa, August 2007.

[33] T. Zheng, M. Woodside, M. Litoiu, "Performance Model Estimation and Tracking using Optimal Filters", IEEE Trans. Software Engineering, V 34 , no. 3, pp 391-406. May 2008.