Deployment of Services in a Cloud Subject to Memory and License Constraints

Jim (Zhanwen) Li, John Chinneck, Murray Woodside
Dept. of Systems and Computer Engineering,
Carleton University,
Ottawa, Canada
{zwli | chinneck | cmw}@sce.carleton.ca

Marin Litoiu
School of Information Technology,
York University,
Toronto, Canada
mlitoiu@yorku.ca

Abstract

When deploying services in a cloud, a balance must be found between performance and capacity of the service, and the memory available on nodes. This is further complicated if the number of replicas of an application is limited, for instance by the available number of licenses. The analysis of interference between services must scale to large numbers of host nodes, applications, replicas of applications, and classes of users. This paper combines a multi-dimensional packing heuristic and network flow optimization to satisfy simultaneous constraints on throughputs, processor utilizations, memory availability and license availability, at a minimum cost and with a minimum of host processors.

1. INTRODUCTION

Applications are increasingly hosted in large processing complexes sometimes called *clouds* [14][15], which share physical resources among many applications. Clouds support flexible deployment as an application's needs change, hide management details from the user and the service provider, and require payment only for resources used. Clouds use virtualization to achieve controlled sharing of resources, rapid redeployment of application images, and isolation of different applications and instances from each other (when they share a host). Applications may include web applications, legacy client-server applications, platforms (i.e. PaaS [7]), infrastructure (i.e. IaaS [16]), and information services of all kinds.

An economic driver for clouds is the efficiencies achieved by sharing resources among applications, beginning with efficient deployment of applications on the hosts of the cloud. A deployment method must scale up to thousands of services running on thousands of hosts, and be cheap enough to re-run frequently as loads and requirements change. This paper assumes an integrated management viewpoint in which deployment should consider the overall cost of the hosts used, and the performance and QoS goals of each application.

To enhance sharing, one node may host more than one application; however memory constraints on each host must be respected. In previous work [19], [20], the present authors described a novel optimization algorithm for large deployment problems. The present paper extends the method to account for host memory and license constraints.

Figure 1 illustrates the deployment of application processes in our experimental cloud for CERAS [6]. The virtual machine monitors control the rate of processing provided to each VM. Deployment issues include (i) the number of replicas of each service, (ii) the selection of processors, (iii) computing power consolidation, (iv) allocation of service replicas, and (v) workload balancing and distribution. The deployment should meet performance targets described in service contracts, (e.g. response time, number of users, capacity given as arrival rates), and economic targets (e.g. cost budgets, power constraints, profit targets) in the presence of constraints (host processing capacity, host memory, licenses per type of application). The deployment solution will be based on data about the execution demands of each application to be deployed, which may be obtained from performance tests or from operating data. Using operating data, changing demands can be tracked.

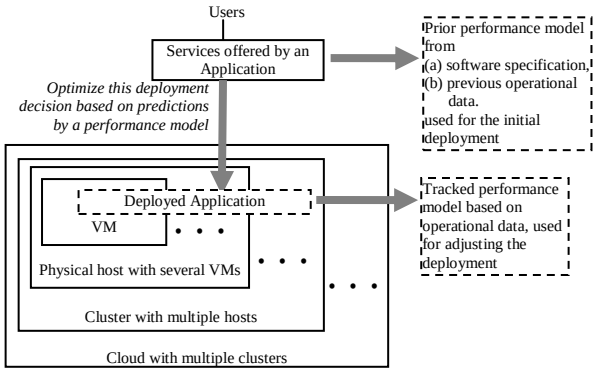

Figure 1. Application processes in a Cloud

We consider that the Cloud uses a flexible licensing model. It owns a number of concurrent licenses (which means the owner can run a specified number of application instances at the same time) for each type of application. In this paper, the concurrent licenses are treated as constraints. When that number is exceeded the Cloud acquires additional pay per use licenses. These additional licenses are reflected in the optimization cost function. Therefore, in the remainder of the

paper, the licenses will be taken into account as soft constraints, with a violation penalty in a cost function.

There is a rich literature on optimal deployment and deployment to satisfy constraints. One approach was to minimize the amount of communications between physically distributed hosts, as described by Bokhari and others (e.g. [2][3][4]). Bin-packing has been applied to pack execution requirements [10], execution and communications requirements [23] and memory; these have been combined in multidimensional bin-packing [8].

Recently Tang et al described a combination of network optimization with heuristic rules that satisfies total demand for each application (as here), and tends to level load and memory requirements [22]. They generate incremental changes to accommodate load changes or new applications, and attempt also to minimize the number of deployment changes in responding to the increment. Because they focus on incremental changes, their placements could drift towards higher costs (e.g. by using more hosts than necessary) through a sequence of decisions; a global re-optimization should be used periodically as a reality check. Another solution for global optimization of this problem, or incremental optimization after a load change, solves a Multiple Knapsack Problem [18]. Neither [18] nor [22] address the question of limiting the number of replicas because of a license constraint.

The present authors combined a contention model with network optimization to optimize deployment subject to response time constraints that take account of resource queueing, including logical resources modeled by extended queueing [20]. However, that work did not account for memory requirements of tasks. This was extended to find a minimal change to accommodate one new application, in [19]. No work we know of addresses the flexible license model proposed here, where concurrent and pay per use licenses coexist. .

This paper is quite different from [19], [20], in that it applies a heuristic allocator motivated by multi-dimensional bin-packing, with one dimension for packing memory on host processors, a second for packing the execution requirements of deployed tasks into a permitted threshold on a host, and a third for packing task allocations into a permitted license-based limit per task. For cost minimization this is augmented with a network flow model similar to [19], [20] but adjusted to minimize the sum of execution cost and a penalty for excess licenses. The resulting algorithm can deploy thousands of tasks on thousands of processors, and is capable of further scale-up.

2. SERVICE SYSTEMS AND DEPLOYMENT

We consider the general service meta-model shown in Figure 2, in which a user class requests a set of services according to its usage profile, and these services may in turn request other services.

UserClasses request services which in turn request other services (exploiting the concepts of Service-Oriented Architecture), forming a web of inter-service traffic. Services are implemented by Applications which run as system tasks or thread pools (ServerTasks), which may have limited capacity. UserClasses have throughput and delay requirements expressed by their SLAs. Resources, such as ServerTasks and Hosts make up the Cloud and are shared among the running Applications. Hosts have flow constraints due to limited processing capacity. ServerTasks can have license constraints. For example, an Application can run its ServerTasks within commercial Data Base Management Systems or in Application Servers and there are upper limits on how many instances of those can be deployed.

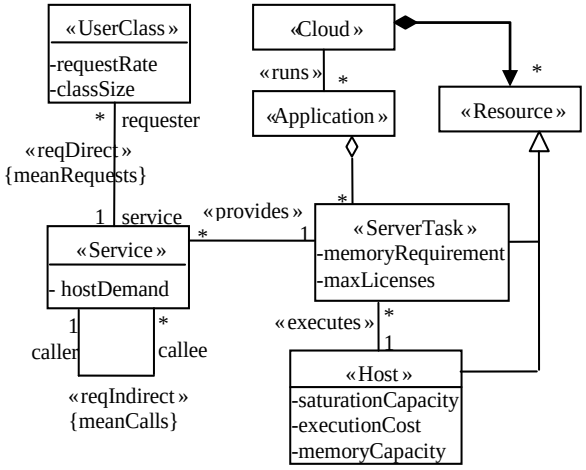

Figure 2. Service system metamodel

Attributes are attached to the stereotypes. Some of the attributes are provided by the service provider, some by the cloud administrator and some are computed or estimated at run time using optimal filters [24]. The attributes will be referred to in the sequel by the following notation:

- $f_{User,c}$ = requestRate of UserClass *c*
- $f_{User,c,SLA}$ = the minimum required value of $f_{User,c}$
- $S_{User,c}$ = the set of services used by class *c*,
- $S_{Task,t}$ be the set of services provided by task *t.*
- Y_{cs} = meanRequests by UserClass *c* to Service *s*, during one response to a user request. This includes the effect of indirect calls to other services at rate meanCalls.
- d_s = mean hostDemand of Service *s*, per invocation, in CPU-sec.
- M_t = memoryRequirement of task *t*, in order to execute (assumed the same for all nodes)
- L_t = maxLicenses available for task *t* (these are the concurrent licenses)
- CL_t = pay-per-use cost of extra licenses for task t, beyond L_t
- C_h = executionCost of host h, a cost factor for a unit of execution on this host. In a homogeneous cloud these are all equal.
- M_h = memoryCapacity of host *h*, memory available for application tasks

Attributes which do not have specific notation are:

- Ω_h = the capacity limit of host h, given as a processing rate relative to a "standard" host used to calibrate execution demands of services. The saturationCapacity of host h is given in the same units, and the utilizat to provide a margin
- U_h = utilization of host h = Ω_h/saturationCapacity of h

2.1. An Example

Figure 3 shows an example with two user classes and six services/applications, labeled by the request rates from one service to another. Tasks are labeled t1 - t6. Other labels, such as the size of the user classes and the host demands of the services, are not shown here.

In the present analysis, the total host demand for each application (task) is computed from the total direct and indirect requests for each of its services, due to requests from each class of users. For example, from Users1 to DB2Serv it is:

$Y_{Users1,\ DB2Serv}$ = 1 x 0.3 x (0.7 + 0.3 x 1.4) = 0.336 requests/user request

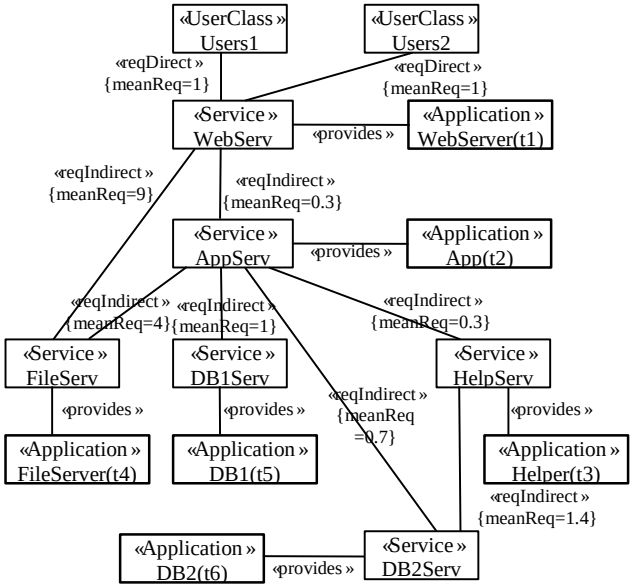

Figure 3. An example of a web application

3. NOTATION FOR DEPLOYMENT

At the required throughputs, the CPU demand d_t, in seconds per second of real time, for task t is:

$$d_t = \Sigma_C\, f_{User,c,SLA}\, \Sigma_{S \in \mathcal{S}_{User,c} \cap \mathcal{S}_{Task,t}} (Y_{cs}\, d_s) \qquad (1)$$

Values for d_s can be obtained by measurement on the reference processor type. The demand d_t is the sum of the processor utilizations for task t, (on hosts of the "standard" rate of processing, required to supply the classes with their required throughputs.

For the example above, suppose (for the sake of illustration) that the execution demands d_s for the six services offered by the six tasks are all unity except for task 4 which is 0.01, and the user request rates $f_{User,c,SLA}$ are 20/sec in class 1 and 30/sec in class 2. Then the task demand rates are:

Task t	1	2	3	4	5	6
Demand d_t (sec/sec)	50	15	4.5	5.1	15	16.8

If a host is faster than the "standard" host, its saturationCapacity and capacity limit are proportionately greater; the demand figures are not changed.

If we define the part of d_t which is executed by host h as α_{ht}, then the analysis will determine the α_{ht} to satisfy

$$\Sigma_t\, \alpha_{ht} = f_{Host,h} \le \Omega_h, \text{ for all } h \qquad (2)$$

$$\Sigma_h\, \alpha_{ht} = d_t \text{ for all } t \qquad (3)$$

The execution rate α_{ht} is provided in practice by controlling the share of the processor given to task t by the virtual machine monitor.

The cost of execution, per unit time, may be different for each host. The cost of executing a unit of demand on host h is defined as C_h, and the total cost is:

$$\text{COST} = \Sigma_h\, C_h\, f_{Host,h} = \Sigma_{h,t}\, C_h\, \alpha_{ht} \qquad (4)$$

The fact that a node executes part of the demand of a task is conveyed by an allocation variable A_{ht},

$$A_{ht} = 1 \text{ if } \alpha_{ht} > 0,\ A_{ht} = 0 \text{ otherwise} \qquad (5)$$

Memory and licenses provide additional constraints. Each task has a memory requirement of M_t bytes, and each host has a memory available of M_h bytes. Using the allocation variables,

$$\text{Memory:}\quad \Sigma_t\, A_{ht}\, M_t \le M_h \quad \text{for each } h \qquad (6)$$

$$\text{Licenses:}\quad \Sigma_h\, A_{ht} \le L_t \quad \text{for each } t \qquad (7)$$

The cost of execution has a contribution $C_h\alpha_{ht}$ for each host-task combination, and a term for the pay-per-use licenses of each task. For the licenses, a cost $CL_h\alpha_{ht}$ is applied to the smallest flows, as follows: For a task t with $\Sigma_h\, A_{ht} > L_t$, order the flows α_{ht} from largest to smallest. For the first L_t of these flows, set $C^*_{ht} = C_h$ (no penalty). For the remainder, which are treated as the excessive allocations, set $C^*_{ht} = (C_h + CL_t)$. Then the cost is

$$\text{COST}^* = \Sigma_{h,t}\, C^*_{ht}\, \alpha_{ht} \qquad (8)$$

3.1. Node Utilization and Quality of Service

Quality of Service is often stated in terms of response delay or throughput[25]. However, these metrics are replaced here by constraints on the host utilizations. Here is the rationale behind this substitution:

- host utilization = assigned demand /saturation capacity
- maximum utilization = capacity limit/saturation capacity

inequality (2) states that every node has a utilization less than the maximum utilization. As in [22] and other works, we assume that if node utilization is kept low enough, say below 60%, then contention will be moderate. Response times then

will be limited to a low multiple of the execution demands to satisfy the request, which should be reasonable.

3.2. Approach

A bin-packing type algorithm HP minimizes the number of hosts required, subject to a *looseness factor* which inflates the demands, in order to slightly over-allocate. Then within this number of processors, the cost is minimized by a network flow calculation *Solve-NFM*. Some final optional adjustments are suggested which can reduce the effect of looseness, or reduce the extra licenses, at the expense of some additional cost.

4. HEURISTIC PACKING (HP)

In the simplest version of bin-packing, a set of items, each having a single dimension of "size" (e.g. length) is given, and the task is to pack all of the items into the smallest number of identical bins, each of which can accommodate a maximum total of item sizes. This problem is known to be NP-complete (e.g. [13]), hence it is usually solved in practice by heuristics which return approximate solutions. Some heuristics provide tight bounds on the optimal solution. See Coffman et al [11] for a survey of bin-packing methods.

In *multi-dimensional bin-packing*, items and bins have more than one dimension, e.g. height, width, and depth. Multidimensional bin-packing was applied to find task-to-processor assignments in digital signal processing systems [8]. This paper considers the problem of packing tasks into the smallest number of hosts while respecting limits on the processing capacity and the memory capacity of the processors. Here, the memory requirement of a task is a fixed amount which must be packed into the memory of each host, and the number of licenses is a discrete limit on the number of processors ("bins") into which a particular task can be packed. Since processing capacity can be subdivided into parts of any size for allocation, this dimension is not a standard bin-packing problem. However we derive a packing heuristic for this dimension motivated by classical static bin-packing methods. HP includes a tendency to respect the limit on the number of licenses of a task, also.

HP uses a packing heuristic suggested by the basic *best-fit* one-dimensional bin-packing algorithm [15]. In the static version of the best-fit algorithm, the items are first sorted from largest to smallest. Items are taken from this list in order and packed into the bin that leaves the smallest amount of unused space. In this problem the primary dimension, execution demand, is infinitely divisible, however the license constraint suggests an upper limit on the number of chunks into which it can be divided.

4.1. Heuristic Packing Formulation of Service Deployment

Packing determines the allocation variables A_{ht} defined above ($A_{ht} = 1$ if $\alpha_{ht} > 0$, 0 otherwise), with the goal

$$\min_A \Sigma_h \max_t A_{ht} = \text{min number of hosts used} \quad (9)$$

subject to the constraints (2),(3),(6),(7). The license constraint (7) is treated as a soft constraint (desirable but not enforced) in this allocation algorithm.

The packing process goes through a sequence of states in which tasks are partially allocated, in which:

M^+_h = remaining memory space of host h,

Ω^+_h = remaining execution demand space of host h,

Ω^{++}_h = available part of Ω^+_h for allocation to a single task, $\Omega^{++}_h = \min(\Omega^+_h, (\textit{reservation factor}) \times \Omega_h)$

d^+_t = remaining execution demand of task t

The control parameter *reservation factor* ≤ 1 forces tasks to be spread across multiple processors, and is examined in Section 6.4.2 below. Initially,

$$M^+_h = M_h,\ \Omega^+_h = \Omega_h,\ d^+_t = d_t \quad (10)$$

Algorithm HP

1. Sort tasks by L_t in an increasing order. (Break ties by sorting by the CPU demands d^+_t, largest first)
2. For each task t in order:

   2(a) Sort the hosts that have M^+_h greater than M_t, by their CPU execution demand space Ω^{++}_h (largest first)

   2(b) If $d^+_t \leq \Omega^{++}_{h(1)}$ for the first host $h(1)$, then the remaining demand of task t will fit into one host. Then:

   2(b*i*) Find the best fit for d^+_t, i.e. the host h with the smallest $\Omega^{++}_h \geq d^+_t$.

   2(b*ii*) Allocate all of d^+_t to host h, that is set $\alpha_{ht} = d^+_t$, decrement Ω^+_h by α_{ht}, and set $d^+_t = 0$.

   2(b*iii*) Proceed to allocate the next task

   2(c) Else if $d^+_t > \Omega^{++}_{h(1)}$ for the first host *h(1)*, the remaining demand will not fit into one host. Then:

   2(c*i*) Allocate as much of d^+_t as possible to $h(1)$, that is set $\alpha_{h(1)t} = \Omega^{++}_{h(1)}$, decrement $\Omega^+_{h(1)}$ by $\alpha_{h(1)t}$, and decrement d^+_t by $\alpha_{h(1)t}$.

   2(c*ii*) Repeat Step 2 for the same task *t*.

3. When execution of all tasks is allocated, compute the cost as COST* given by Eq. (8).

4.2. Effectiveness of the Packing Heuristic

A scalable example conforming to the service metamodel of Figure 2 was defined as follows.

- a list of 2000 tasks was created, each with parameters chosen randomly in an interval: CPU demands d_t in interval [10 , 70] ms., memory requirement M_t in [5, 25] units, and licenses L_t in [1, 5].

- a list of 1000 host processors was created, with capacities (i.e. speed factors) Ω_h in [80 , 120], memory M_h in [50,100] units, and host cost factors C_h in [1, 3].

To evaluate the algorithms at different scales of problem, configurations were constructed from the first n tasks and first m hosts in these lists, for values of the product $m \times n$ ranging from 200 to 2,440,000. This product is the number of allocation variables A_{ht}, which indicates the scale of the problem.

The packing heuristic alone gave the results shown in Table 1. The times are modest. The cost (COST*) per task is roughly constant at about 0.06 units, across the entire table. Cost is not minimized by HP.

TABLE I. Heuristic packing alone: results for cost and execution time over problem scales

tasks n	20	45	75	115	160	240	300	400	1000	2200
hosts m	10	25	40	70	95	125	166	250	600	1200
arcs mn	200	1125	3000	8050	15200	30000	49800	100000	600000	2640000
Cost x1000	1.17	2.59	4.51	7.25	10.03	14.8	18.2	23.0	61.8	133
Time (sec)	0.016	0.016	0.015	0.031	0.047	0.078	0.125	0.235	1.47	14.2

5. THE NETWORK FLOW MODEL (NFM)

Costs arre minimized by a Network Flow Model (NFM) [20]. This is a graph with arcs that carry flows (representing execution demands) and nodes which operate on the flows, as illustrated in Figure 4. Figure 4 shows one representative node of each type. The flow into a host node is its total execution demand $f_{HOST,h}$, divided between its tasks, then between their entries, and finally between the user classes that request it. In the NFM the nodes balance the flow between their input and output arcs (total input = total output), so the model enforces the allocation of each task demand rate (as demanded by the user request rates) to some host.

There is an arc from host h to each task t which is permitted to be deployed on h, with flow α_{ht} (the demand rate executed on host h, to satisfy the needs of task t). If multiple replicas of a task are deployed, each has a flow from its processor.

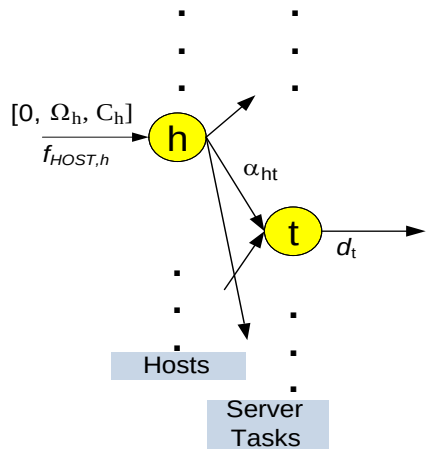

Figure 4. The network flow model notation for this problem

Each arc is labeled with a triple of parameters [l,u,c]: the lower flow bound l (default 0), the upper flow bound u (default infinity), and the cost per unit of flow c (default 0). The parameters are not shown where all take the default values [0, ∞, 0]. The input arcs on the left in Figure 4 are labeled by [0, Ω_h, C_h] meaning that

- $f_{HOST,h} \geq 0$,
- the host capacity limit is flow $f_{HOST,h} \leq \Omega_h$,
- the cost is C_h per unit of flow.

For a homogeneous set of processors all of "standard" processor type, the capacities are all 1.0.

A general description of NFMs is given in [9].

To restrict the allocation of demand from a task to a subset of hosts, the arcs to other hosts are simply removed from the graph.

Procedure Solve-NFM

It is a standard linear programming (LP) problem to minimize the cost of the host-to-task flows α_{ht}, expressed as COST* defined by (8), subject to the constraints (2),(3).

This procedure however does not scale well. There are $n \times m$ arc flows to be determined by an LP with nm variables, which may quickly reach into the millions [1]. To avoid this blow-upwe will apply HP first to reduce the number of arcs.

6. THE COMBINED ALGORITHMS

Because HP alone does not minimize cost and may not succeed in satisfying the license constraints, it was combined with an NFM solution, in different ways:

- HP-NFM: An initial allocation is found and the host-to-task arcs are pruned to minimize the size of the subsequent NFM. The NFM then determines the processing rate on each allocated host. To provide some freedom at the NFM step, all demands are multiplied by a looseness factor ≥ 1 at the HP step. The looseness is removed at the NFM step.
- HP-NFM-HP: When HP-NFM failed to satisfy the license constraints, a final packing step succeeded in re-imposing them (in our experiments). There are two variants, described below.
- In the order NFM-HP, the NFM first minimizes cost and then HP tries to satisfy the memory and license constraints. It works well but does not scale, and to conserve space it is not reported here. We report on the successful HP-NFM variant below.

6.1. Combined Algorithm HP-NFM

A small number of processors satisfying the memory constraint and if possible the license constraint is first found by HP (for the "loosened" problem), then the minimum-cost rates of processing the tasks on those processors are found by the NFM (with the loosening removed). A looseness factor of 1.1 was used in our experiments.

In this heuristic packing step, every host also has a *reservation factor* less than one, which is an upper bound on the capacity fraction that can be reserved by any single task; this prevents a host from being reserved by a single task, which could limit the reasonable choices for the subsequent NFM.

Algorithm HP-NFM

1. Apply Algorithm HP to the problem, after augmenting the execution demands d_t of all tasks by the looseness factor.

2. Construct a pruned NFM for the problem with

   (a) nodes for the tasks, and for the hosts that have been used in Step 1,

   (b) the task demands d_t without the looseness factor, and

   (c) arcs from each host to the tasks allocated to it by HP, with cost C^*_{ht} on arc (h,t) (including penalties for exceeding the license limit).

3. Apply Solve-NFM to minimize COST*

6.2. Algorithms HP-NFM-HP1 and HP-NFM-HP2

The HP-NFM algorithm may spread the allocations out over more processors than necessary, due to the initial looseness factor. Therefore a final HP step without looseness was introduced to reduce the number of allocated processors if possible. Two approaches were used:

- To counter underutilized hosts, an *idleness threshold* for hosts was chosen. Following Step 3 above, hosts with an execution demand space greater than this, i.e. with

  $\Omega^+_h \geq$ (idleness threshold for host h)

  are de-allocated, and their task execution demands are repacked using the HP algorithm.

- To counter license violations, tasks which exceed their license limit had the excess replicas de-allocated.

There are two variants of the algorithm. In HP-NFM-HP1 both criteria are applied; in HP-NFM-HP2 only the license criterion is applied. They are described together.

Algorithm HP-NFM-HPx

1 - 3 execute Algorithm HP-NFM. At the end, the remaining execution demand for all tasks is $d^+_t = 0$.

4. In the HP1 variant only: unpack underutilized hosts. For each host h with $\Omega^+_h \geq$ idleness threshold,

   (4a) For each task t, increment d^+_t by α_{ht} .

   (4b) Set $\Omega^+_h = \Omega_h$

5. In both variants: unpack for license violations. For each task t which violates its license constraint (i.e. has $\Sigma_h A_{ht} - L_t = V_t > 0$),

   5(a) Sort the demand allocations α_{ht} of task t in increasing order,

   5(b) For the first V_t of these demand allocations, increment both d^+_t and Ω^+_h by α_{ht}.

6. Apply Algorithm HP (with no looseness factor) to pack the remaining execution of the set of tasks with $d^+_t > 0$, into the processors with $M^+_h > 0$.

6.3. Computational Experience and Scalability

1) Comparison on Random Cases

The algorithms are compared on how they process the same scalable example as was described for HP alone, but now for 100 replications at each scale with randomly generated parameters. The running times, the final value of COST*, and the license cost are shown in TABLE II. .

TABLE II. Results over 100 replications of three algorithms

Size					
tasks n	240	300	400	1000	1200
hosts m	125	166	250	600	800
allocations n x m	30000	49800	100000	600000	960000
COST* x 1000: average and (standard deviation as %)					
HP	14.6 (±8.18%)	18.2 (±6.20%)	24.2 (±5.42%)	60.3 (±3.14%)	72.3 (±3.10%)
HP-NFM	12.4 (±8.18%)	15.5 (±7.59%)	18.4 (±6.31%)	45.7 (±4.40%)	51.4 (±4.02%)
HP–NFM-HP1	13.1 (±8.18%)	15.9 (±8.22%)	19.2 (±7.39%)	51.19 (±5.62%)	58.8 (±5.50%)
HP–NFM-HP2	12.7 (±8.49%)	15.6 (±7.46%)	18.93 (±6.91%)	47.6 (±4.70%)	53.12 (±4.16%)
Average Solution Time (sec)					
HP	0.048	0.078	0.171	1.132	1.633
HP-NFM	0.075	0.111	0.189	1.210	1.513
HP–NFM-HP1	0.091	0.139	0.235	1.283	2.200
HP–NFM-HP2	0.084	0.118	0.191	1.212	1.686
Average Number of Hosts in Use					
HP	103.34	130.17	175.43	435.04	528.50
HP-NFM	116.20	149.68	210.00	487.48	613.80
HP–NFM-HP1	100.93	127.01	172.27	425.00	518.55
HP–NFM-HP2	116.16	149.52	210.00	487.46	613.79
Average License Cost (average extra licenses per case) [cases which meet license constraints]					
HP	0	0	0	0	0
HP-NFM	28.84 (0.14) [75/100]	44.16 (0.22) [69/100]	10.531 (0.08) [93/100]	13.346 (0.11) [92/100]	7.54 (0.05) [95/100]

HP–NFM-HP1	0	0	0	0	0
HP–NFM-HP2	0	0	0	0	0

The values of control parameters used were:

- maximum capacity: 80% of saturation capacity,
- license violation penalty: 10 times the number of license violations (where this may be fractional),
- looseness factor: 1.1,
- reservation factor: 20%,
- idleness threshold: 20% of the maximum capacity.

The results in Table II show

- HP alone is competitive in every category except COST*. This is not surprising since it ignores this factor; in any case it is worse by no more than 20% at the scales tested here (but the gap grows with scale).
- The lowest COST* and fastest solution is always obtained with HP-NFM, since it terminates with optimization. However it also terminates with residual license violations (whose cost is included in the lowest cost) and uses more hosts. If either of these factors is of high importance to the management decisions, then this algorithm is inferior.
- HP-NFM-HP1 and 2 achieved zero license violations. Of these:
    - The smaller number of hosts is always obtained by HP-NFM-HP1. This difference is as high as 20% in the rightmost column, and grows with the system scale, left to right.
    - The lower solution time and COST* are always obtained by HP-NFM-HP2.

So a user may select the appropriate heuristic according to which goals are most important: HP-NFM or HP-NFM-HP2 for cost, HP-NFM-HP1 for hosts.

2) Looseness and Reservation Factors

The impact of the looseness and reservation factors used in Algorithm HP-NFM is explored in Table III. This table shows the final cost after applying HP-NFM to a single randomly generated case with 1000 tasks and 600 hosts.

TABLE III. Impact of looseness and reservation factors(case with 1000 tasks and 600 hosts)

Reservation factor	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Looseness factor	COST* x 1000								
1	46.0	50.0	54.4	59.2	61.5	62.1	61.7	61.8	61.8
1.1	45.6	48.7	53.3	57.0	60.0	61.5	62.0	61.8	61.8
1.2	nil	47.6	51.0	54.7	56.9	61.0	60.9	61.0	61.0
1.3	nil	46.5	49.6	52.4	54.6	58.4	60.5	60.8	60.7
1.4	nil	45.9	47.0	50.9	52.0	55.3	58.3	60.3	60.6

Reducing the reservation factor reduces the cost, which validates the use of the reservation factor to limit how much processing for a single task is permitted on a single host. Increasing the looseness factor also reduces the cost only a little, indicating that the freedom it provides to the later NFM step is not very valuable.

The solution time varies somewhat with these factors but in no very clear pattern. The number of hosts used in the final allocation increases monotonically with the looseness factor, which is not surprising since the looseness factor inflates the amount of demand to be executed. As the reservation factor increases across the table, the number of hosts first increases and then decreases, because it is also subject to which hosts are chosen rather than only the number of hosts selected.

From these experiments and observations, we conclude that combining a large looseness factor and a small reservation factor is helpful in finding a better solution, in general.

TABLE IV. Comparing HP and HP-NFM on the scalable example of Section 4.2 and Table I.

Cases, with their Size Parameters										
tasks *n*	20	45	75	115	160	240	300	400	1000	2200
hosts *m*	10	25	40	70	95	125	166	250	600	1200
Arcs x1000	0.2	1	3	8	15	30	50	100	600	2640
COST* x 1000										
HP	0.884	2.20	3.94	6.17	7.92	11.76	15.11	20.30	43.0	99.0
HP-NFM	0.746	1.63	3.16	4.83	5.74	8.90	11.03	12.75	29.9	67.0
Fractional Cost Improvement by HP-NFM										
HP-NFM over HP	0.156	0.261	0.20	0.22	0.28	0.24	0.27	0.37	0.33	0.32
Solution Time (Seconds)										
HP	0.016	0.031	0.109	0.188	0.906	0.297	0.266	0.39	2.31	14.6
HP-NFM	0.125	0.187	0.344	0.329	0.359	0.313	0.453	0.844	2.91	16.5

6.4. Recommended Approach: HP-NFM-HPx

HP alone only minimizes processors subject to the memory constraints, NFM only minimizes costs. However the combination of the two in HP-NFM provides a reasonable solution that takes memory requirements, license costs, CPU demands, host CPUs and memory capacities etc. into account.

The optional HP-NFM-HPx variants give some advantages. Of the final step HP1 or HP2, HP1 is significantly better for the

number of deployed hosts, while HP2 is better for solution time and total COST*.

7. CONCLUSIONS

This paper presents an effective algorithm for the heuristic optimization of task deployments in Clouds or in any large system shared by many applications. The algorithm combines the advantages of a packing heuristic with linear programming. A key achievement is the ability to include many realistic constraints in the model, including scalability, CPU and memory requirements, and host capacities, while also addressing goals of low economic costs and a small number of hosts in use.

Limitation of the number of licenses is a secondary objective in HP (it is sought only after meeting the primary objectives) and is encouraged indirectly in NFM by the pay-per-use costs. Thus it is a soft constraint in this work.

A new heuristic packing algorithm HP is defined to pack execution demand (which is infinitely divisible, but with the number of divisions limited by the license constraint) into hosts. HP is inspired by a classic bin-packing approach, but required significant adaptation.

Numerical experiments show that one variant of the combined HP/network optimization approach, called HP-NFM-HPx is preferred. It comes in two flavours, the first favouring a reduced number of hosts in use, and the second favouring lower economic costs and faster solutions. Other experiments explore the search parameters introduced in the heuristic parts of the algorithm.

The tests described here went as high as two and half million candidate allocations, of over 2000 tasks (each with several replicas) on over 1000 hosts. The mechanics of the algorithm continue to scale above that size, perhaps approaching a million hosts. The problem complexity appears to be dominated by the successive sorting steps applied to lists of tasks and hosts in the applications of HP (heuristic packing).

This work is a step in our research on developing advanced management tools for Cloud computing. In future work, these algorithms will be deployed in Websphere XD and Tivoli in the virtualization environment of the CERAS Cloud.

ACKNOWLEDGMENTS

This research was supported by OCE, the Ontario Centres of Excellence, and by the IBM Toronto Centre for Advanced Studies, as part of the program of the Centre for Research in Adaptive Systems (CERAS).

REFERENCES

[1] MS Bazaraa, JJ Jarvis, HD Sherali, “Linear Programming and Network Flows”, John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[2] M.J. Berger, S.H. Bokhari, "A Partitioning Strategy for Nonuniform Problems on Multiprocessors". *IEEE Trans. Comput,*. v 36 n 5 , pp 570-580, 1987

[3] S. H. Bokhari, "On the mapping problem," IEEE Transactions on Computers, vol. C-30, no. 3 pp. 207–214, 1981.

[4] S.H Bokhari, "Partitioning Problems in Parallel, Pipeline, and Distributed Computing". *IEEE Trans. Comput.* v 37 n 1, pp 48-57, 1988.

[5] N. Bobroff, A. Kochut, and K. Beatty. "Dynamic placement of virtual machines for managing SLA violations”. In *Proc. Integrated Management 2007*, pp 119-128, Munich, May 2007.

[6] CERAS (Centre of Excellence for Research in Adaptive Systems) https://www.cs.uwaterloo.ca/twiki/view/CERAS

[7] M. Chang, J. He, E. Castro-Leon. "Service-Orientation in the Computing Infrastructure ", *Proc. 2nd IEEE Int. Symp. on Service-Oriented System Engineering (SOSE'06)*, 2006.

[8] 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

[9] J.W. Chinneck, "Processing Network Models of Energy/Environment Systems", *Computers and Industrial Engineering*, vol. 28, no. 1, pp. 179-189. 1995

[10] E. Coffman, M. Garey, D. Johnson, "An application of bin-packing to multiprocessor scheduling", *SIAM J. Computing*, vol. 7, pp. 1-17, Feb. 1978

[11] E.G. Coffman, M.R. Garey, D.S. Johnson, “Approximation Algorithms for Bin Packing: a Survey”, in *Approximation Algorithms For NP-Hard Problems*, D. S. Hochbaum, Ed. PWS Publishing Co., Boston, MA, pp 46-93, 1997.

[12] 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.

[13] M.R. Garey, D.S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman and Company; 1979.

[14] IBM Corp., "From Cloud Computing to the New Enterprise Data Center", http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/CloudComputingNEDC_wp_28May.pdf, 2008.

[15] IBM Corp., “Cloud Computing”, http://www.ibm.com/ibm/cloud/ibm_cloud/, 2009

[16] IBM Corp., “Dynamic Infrastructure” , http://www-03.ibm.com/systems/dynamicinfrastructure/,2009

[17] D.S. Johnson. “Fast algorithms for bin packing”, *Journal of Computer and System Sciences*, v 8, pp 272–314, 1974.

[18] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, A. Tantawi, "Dynamic placement for clustered web applications", *Proc. 15th Int. Conf. on the World Wide Web* May 2006. ACM, New York.

[19] Z. Li, J. Chinneck, M. Woodside, M. Litoiu, 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.

[20] Z. Li, J. Chinneck, M. Woodside, and 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.

[21] M. Steinder, I. Whalley, D. Carrera, I. Gaweda D. Chess, "Server virtualization in autonomic management of heterogeneous workloads ". *Proc. Integrated Management (IM 2007)*, Munich, May 2007.

[22] C. Tang, M. Steinder, M. Spreitzer, G. Pacifici, "A scalable application placement controller for enterprise data centers ", *Proc. 16th Int. Conf. on the World Wide Web (WWW '07)*. ACM, pp 331-340, 2007

[23] 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.

[24] 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, 2008.

[25] Salesforce.com, Quality of Services, http://trust.salesforce.com/trust/status/, May 31, 2009.