New Approaches to Optimization and Utility Elicitation in Autonomic Computing

Relu Patrascu and Craig Boutilier
Department of Computer Science
University of Toronto
Toronto, ON, M5S 3H5, Canada
{cebly,relu}@cs.toronto.edu

Rajarshi Das Jeffrey O. Kephart
Gerald Tesauro William E. Walsh
IBM T.J. Watson Research Center
19 Skyline Dr.
Hawthorne, NY 10532, USA
{rajarshi, kephart, gtesauro, wwalsh1}@us.ibm.com

Abstract

Autonomic (self-managing) computing systems face the critical problem of resource allocation to different computing elements. Adopting a recent model, we view the problem of provisioning resources as involving utility elicitation and optimization to allocate resources given imprecise utility information. In this paper, we propose a new algorithm for regret-based optimization that performs significantly faster than that proposed in earlier work. We also explore new regret-based elicitation heuristics that are able to find near-optimal allocations while requiring a very small amount of utility information from the distributed computing elements. Since regret-computation is intensive, we compare these to the more tractable Nelder-Mead optimization technique w.r.t. amount of utility information required.

1 Introduction

The complexity of large, distributed computing systems has provided considerable impetus for research in *autonomic computing* [5]. An important factor in such autonomy is the ability to continuously allocate resources (e.g., application server or CPU time, disk space) to distinct computing elements [1]. Unfortunately, optimal allocation of resources requires knowing the *utility* of different levels of resource to the various computing elements, information which is inherently distributed and in many cases difficult to determine.

Boutilier *et al.* [1] propose a model for resource allocation in autonomic systems in which explicit *utility elicitation* is used to extract relevant information from distributed elements. A central *provisioner* queries elements for samples of their utility functions at various resources levels, and makes allocations based on these samples. With only partial utility information, an optimal allocation cannot be determined, so instead, the notion of *minimax regret* is used to determine a suitable allocation. In this paper, we improve on the methods of [1] in two ways and consider an alternative approach to elicitation. First, we propose an integer programming (IP) formulation of the minimax regret problem that offers great computational benefits over the algorithms of [1]. Though the IP has infinitely many constraints, we derive a tractable constraint generation procedure [4] to effectively solve this IP. Second, we propose several new regret-based elicitation strategies that exploit the anytime nature of our new minimax regret algorithm. Finally, we consider the use of Nelder-Mead optimization [6] as an alternative approach to elicitation and optimization. While Nelder-Mead generally requires far more queries than regret-based elicitation, it is far more tractable, thus proving more suitable in cases where evaluating the utility of resources is relatively inexpensive.

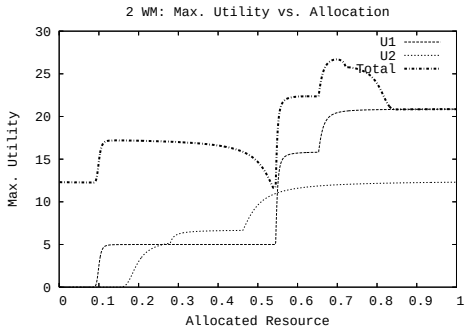

Figure 1: Maximum utility curves: U_1, U_2 show the maximum utility to WM1 and WM2, resp., as a function of the resource level allocated to each. The total system utility as a function of a_1 to WM1 (with $1 - a_1$ to WM2) is also shown.

2 Regret-based Resource Allocation

We begin by reviewing the regret-based resource allocation model of [1]. An automated resource manager, or *provisioner*, must allocate resources to various *workload managers (WMs)* in a data center [7]. Each WM, given a specific allocation of resources, must decide how best to use them to service various client contracts and maintain, say, specific *quality of service (QoS)* levels for each of the transaction classes within these contracts. We assume for simplicity that the WMs use a single, scalar resource type. Given local information (e.g., distribution over transaction type demand), a WM i can compute $u_i(a_i)$, the maximum expected revenue it could obtain with resource level a_i. Fig. 1 shows examples of two such *utility functions*. Unfortunately, these utility functions generally have no convenient closed form, and computation of the

utility $u_i(a_i)$ of a specific allocation level a_i often requires a combination of complex optimization and simulation.

Because the distribution of client demand changes over time, the provisioner will periodically reallocate resources between the WMs (hence offline computation of a fixed allocation will not suffice). When periodically reallocating resources (e.g., as demands change), ideally, the provisioner would solve (assuming n WMs):

$$\arg\max_{\mathbf{a}\in A} \sum_{i\leq n} u_i(a_i) \tag{1}$$

Here A is the set of feasible allocations and we assume the u_i are independent. The provisioner thus maximizes overall organizational utility (e.g., the max of the "total" curve in Fig. 1). However, the provisioner does not have direct access to the functions u_i since it typically lacks relevant internal models and state information about individual WMs (e.g., client demand distributions, dynamic QoS guarantees, etc.). Nor can the u_i be communicated easily by the WMs, since they generally have no closed form.

Fortunately, optimization (or approximation) of Eq. 1 does not generally require full utility information. Boutilier *et al.* [1] exploit this fact by proposing to view the utility demands of optimal resource allocation as a *utility elicitation problem* [10; 3]. In their model, the provisioner asks WMs for the utility of allocation levels at specific "critical" points—those parts of local utility functions that have the most impact on global optimization. Furthermore, trade-offs can be made between local computational expense, number of queries, and (global) decision quality.

A key aspect of this model is the ability to determine an approximately optimal decision given incomplete utility function information. Boutilier *et al.* [1] propose the use of the *minimax regret* decision criterion [9; 2] to allow for robust allocations in the face of such utility function uncertainty; we now formalize the notion.

An *allocation* is a vector $\mathbf{a} = \langle a_1, \ldots, a_n\rangle$ such that $a_i \geq 0$ and $\sum_i a_i \leq 1$ (a_i is the fraction of resources obtained by i). Let A be the set of feasible allocations. We assume each WM's utility function u_i is monotonic non-decreasing. A *utility vector* $\mathbf{u} = \langle u_1, \ldots, u_n\rangle$ is a collection of such utility functions, one per WM. The *value* of an allocation $\mathbf{a}$ given $\mathbf{u}$ is the sum of the WM utilities: $V(\mathbf{a}, \mathbf{u}) = \sum_i u_i(a_i)$.

We assume the provisioner has a collection of samples of each WM's utility function (obtained through utility elicitation as discussed later). Specifically, let

$$0 = \tau_i^0 < \tau_i^1 < \ldots < \tau_i^k = a_i^\top$$

be a collection of $k+1$ thresholds at which samples $u_i(\tau_i^j)$ have been provided ($a_i^\top \leq 1$ is the maximum fraction of resources that WM i can profitably use). This collection of samples defines a set of k *bins* into which allocation a_i might fall (see Fig. 2(a)). Let $[a_i]$ denote the index of the bin in which a_i lies. Let U be the set of feasible utility vectors (those whose components u_i are nondecreasing and consistent with the sampled points). Fig. 2(a) shows bounds on a WM utility function given a set of samples. The vertical lines indicate bin boundaries, and the horizontal lines upper and lower bounds on utility.

Given partial knowledge of WM utility functions in the form of samples, the provisioner can measure the quality of a specific allocation in terms of its *maximum regret*. This gives a bound on the worst-case error associated with an allocation, assuming an adversary can pick the true utility vector from the feasible set U.

Defn. The *maximum regret of allocation* $\mathbf{a}$ *w.r.t.* $\mathbf{a}'$ is

$$MR(\mathbf{a}, \mathbf{a}') = \max_{\mathbf{u}\in U} V(\mathbf{a}', \mathbf{u}) - V(\mathbf{a}, \mathbf{u})$$

The max regret of allocation $\mathbf{a}$ is then

$$MR(\mathbf{a}) = \max_{\mathbf{a}'\in A} MR(\mathbf{a}, \mathbf{a}') \tag{2}$$

An allocation $\mathbf{a}^* \in \arg\min_{\mathbf{a}\in A} MR(\mathbf{a})$ is said to have *minimax regret*. The minimax regret level $MMR(U)$ of feasible utility set U is $MR(\mathbf{a}^*)$.

Minimax regret offers a reasonable method for resource allocation in the face of utility function uncertainty. It minimizes the amount of utility one could potentially sacrifice by acting in the face of such uncertainty.

Boutilier *et al.* [1] compute the minimax optimal allocation iteratively using a search algorithm that enumerates *exhaustive point-wise allocations (EPAs)*; the method calls a *max regret IP* (see below) as a subroutine. Unfortunately, because the number of EPAs grows exponentially with the number of WMs, the algorithm does not scale well (though many EPAs can be pruned through domination testing). Hence, computational results presented in [1] rely on heuristic approximation and even then are limited to 3–4 WMs and roughly 30–40 queries per WM.

The subroutine to compute the maximum regret of an allocation $\mathbf{a}$ (Eq. 2) constitutes an important part of the algorithm. We note that there is a single u_i (for each WM) that supports $MR(\mathbf{a}, \mathbf{a}')$ (w.r.t. any competing allocation $\mathbf{a}'$). Let S be a fixed set of sampled utility points (over all WMs). We define $UBU_i(\mathbf{a})$ to be utility function u_i that assigns a_i its lowest possible utility given S_i, but all other allocations their highest utility consistent with the fact that a_i has its lowest. This *upper bound utility function* can be constructed simply as follows: set the utility over the interval $[\tau_i^{[a_i]-1}, a_i]$ to the lower bound $u_i(\tau_i^{[a_i]-1})$, and the interval $(a_i, \tau_i^{[a_i]}]$ to the upper bound. All other bins b_i^j are set to their maximum values. Fig. 2(b) illustrates the construction of $UBU_i(\mathbf{a})$. The utility vector $UBU(\mathbf{a})$ obtained in this way ensures the following (formalizing the observation of [1]):

Prop. 1 $UBU(\mathbf{a}) \in \arg\max_{\mathbf{u}\in U} V(\mathbf{a}', \mathbf{u}) - V(\mathbf{a}, \mathbf{u}),\ \forall \mathbf{a}'$

As a consequence, given a specific $\mathbf{a}$, we can compute $MR(\mathbf{a})$ without requiring an explicit maximization over U, but rather can set $\mathbf{u}$ to $UBU(\mathbf{a})$ and simply maximize over adversarial allocations $\mathbf{a}'$. This maximization can be formulated as an integer program [1] involving variables that denote the adversarial allocation $\mathbf{a}'$ as well as indicator variables corresponding to the "bins" in Fig. 2(b) (where each bin corresponds to a constant level of the utility function); these variables denote whether $\mathbf{a}'_i$ lies in the corresponding interval. The objective is to maximize the difference in utility (which is fixed

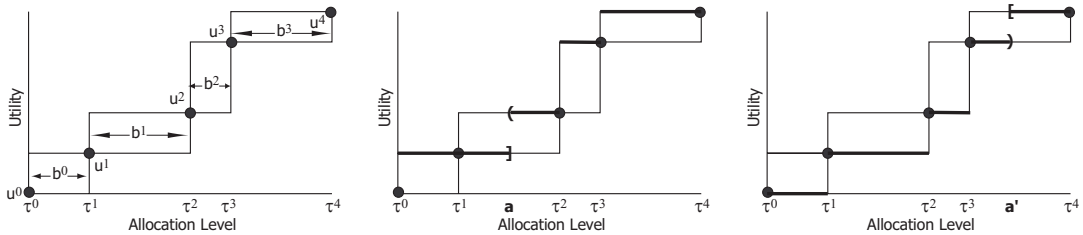

Figure 2: (a) Bounds on feasible utility functions; (b) $UBU(\mathbf{a})$ indicated by bold lines; (c) $LBU(\mathbf{a}')$ indicated by bold lines.

by utility vector $UBU(\mathbf{a})$) between some $\mathbf{a}'$ and $\mathbf{a}$. The solution to the IP produces a *witness* $\mathbf{a}^w$—the adversarial allocation that maximizes regret—as well as the max regret of $\mathbf{a}$: $MR(\mathbf{a}) = MR(\mathbf{a}, \mathbf{a}^w)$.

3 Constraint Generation

To circumvent the computational difficulties facing minimax regret computation, we propose a new formulation of the problem. We begin with the following observation. Given samples S, just as with $UBU(\mathbf{a})$, we can show that there exists a utility vector $LBU(\mathbf{a}')$ that maximizes the pairwise regret against a fixed adversarial allocation $\mathbf{a}'$ for any $\mathbf{a}$. This *lower bound utility function* can be constructed as follows: set the utility over the interval $[\tau_i^{[a'_i]-1}, a'_i)$ to the lower bound $u_i(\tau_i^{[a_i]-1})$, and the interval $[a'_i, \tau_i^{[a'_i]}]$ to its upper bound. All other bins b_i^j are set to their minimum values. Fig. 2(c) illustrates the construction of $LBU_i(\mathbf{a}')$. $LBU(\mathbf{a}')$ obtained in this way ensures the following:

Prop. 2 $LBU(\mathbf{a}') \in \arg\max_{\mathbf{u}\in U} V(\mathbf{a}', \mathbf{u}) - V(\mathbf{a}, \mathbf{u}),\ \forall \mathbf{a}$

To prove this we observe that $LBU_i(\mathbf{a}')$ assigns a_i its greatest possible utility, while assigning all other allocations their least utility with the exception of those allocations $a_i \geq a'_i$ that lie within the same bin as a'_i. But for any such allocation, we have $u_i(a_i) - u_i(a'_i) \geq 0$ by monotonicity no matter how we set u_i, and this utility vector ensures this quantity is 0.

We can formulate $MMR(U)$ as the solution to the following mathematical program:

$$MMR(U) = \min_{\mathbf{a}} \max_{\mathbf{a}'} \max_{\mathbf{u}\in U} [V(\mathbf{a}', \mathbf{u}) - V(\mathbf{a}, \mathbf{u})] \quad (3)$$

$$= \min_{\mathbf{a}} \max_{\mathbf{a}'} [V(\mathbf{a}', LBU(\mathbf{a}')) - V(\mathbf{a}, LBU(\mathbf{a}'))] \quad (4)$$

$$= \min_{\mathbf{a},\delta} \delta \quad \text{subject to} \quad (5)$$

$$\delta \geq V(\mathbf{a}', LBU(\mathbf{a}')) - V(\mathbf{a}, LBU(\mathbf{a}')), \quad \forall \mathbf{a}' \in A$$

The reformulation in Eq. 4 is justified by Prop. 2, while Eq. 5 is a standard transformation of a minimax program into a minimization (thus allowing LP or IP solvers to be used directly). Unfortunately, this conversion leads to an infinite IP, since we have infinitely many constraints (one for each feasible allocation $\mathbf{a}'$) and infinitely many variables (required to represent the "bins" in the different functions $LBU(\mathbf{a}')$).

To circumvent this problem, we use a constraint generation procedure to focus only on relevant constraints (those that will be active in the optimal solution). Intuitively, we solve a relaxed IP with only a subset of all constraints, those corresponding to a small set *Gen* of adversarial allocations $\mathbf{a}'$. This can be viewed as finding the minimax optimal allocation against a "restricted" adversary who can only select allocations in *Gen*. At the purported solution $\mathbf{a}$ (with purported max regret δ), we then compute a *maximally* violated constraint by computing $MR(\mathbf{a})$. If one exists, and its corresponding allocation is the witness $\mathbf{a}^w$, then $MR(\mathbf{a}, \mathbf{a}^w) > \delta$ and we know that the current $\mathbf{a}$ and δ are sub-optimal. Hence we add $\mathbf{a}^w$ to *Gen* and iterate. However, if $MR(\mathbf{a}) = \delta$, then no constraints are violated and $\mathbf{a}$ must be minimax optimal. The procedure can be summarized as follows:

1. Let *Gen* $= \{\mathbf{a}'\}$ for some $\mathbf{a}' \in A$.
2. Solve the relaxed minimax regret IP (Eq. 5) using only constraints for those $\mathbf{a}' \in$ *Gen*. Let $\mathbf{a}^*$ be the IP solution with objective value δ^*.
3. Compute the max regret of $\mathbf{a}^*$ using the max regret IP (Sec. 2), giving max regret r^* and witness $\mathbf{a}^w$. If $r^* > \delta^*$, then add $\mathbf{a}^w$ to *Gen* and repeat from Step 2; otherwise (if $r^* = \delta^*$), terminate with minimax optimal solution $\mathbf{a}^*$ (with regret level δ^*).

By restricting the precision of allowable bins, the procedure is guaranteed to converge in a finite number of iterations, and in practice (as we see below) converges very quickly. We note that the IP gets larger at each iteration not just because of the number of constraints, but also because new variables must be added to reflect the new bins created by the new adversarial $\mathbf{a}'$ added to *Gen* (at most one variable per WM, through clever management). We also note that the procedure lends itself to anytime implementation. First, should we terminate the process before convergence (i.e., before all violated constraints are added to *Gen*), the solution obtained gives us a lower bound on $MMR(U)$; furthermore, the true $MR(\mathbf{a}^*)$ provides a precise measure of the quality of solution $\mathbf{a}^*$ as well as an upper bound on $MMR(U)$. We also envision situations where the time to obtain the minimax optimal allocation is critical, and one readily accepts a timely approximation.[1] Hence, instead of solving Eq. 5 exactly, we allow the mixed integer program solver to return its best feasible solution obtainable within a given period of time. The time limit is enforced whether the exact solution or just an approximation has been reached and the minimax regret or the maximum regret is returned (respectively). We exploit this time-bounded approximation below, and discuss running times in the next section.

[1] Not the tightest possible bound.

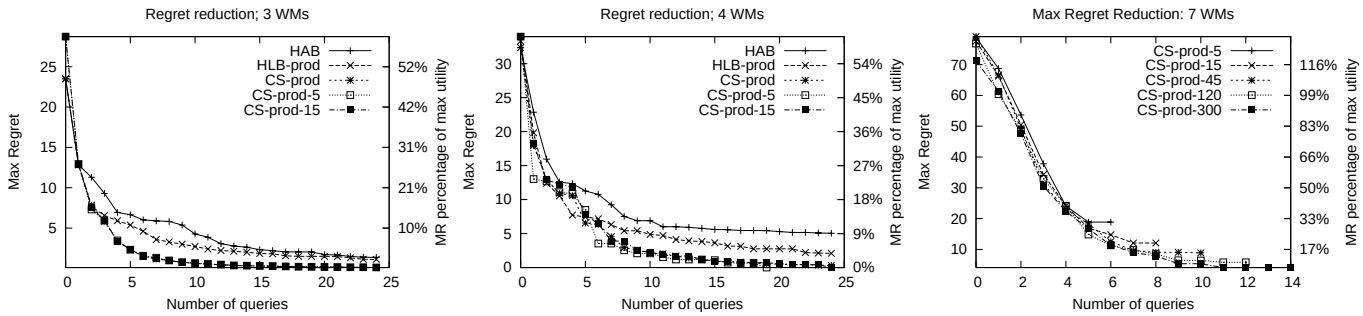

Figure 3: Regret reduction per query: (a) 3 WMs; (b) 4 WMs; and (c) 7 WMs.

It is important to note that the use of $LBU(\mathbf{a}')$ in the constraints is critical to the success of the procedure. Standard constraint generation would suggest computing a pair $\langle \mathbf{a}', \mathbf{u} \rangle$ for which the corresponding constraint is violated. That is precisely what the max regret IP does by using the utility vector $UBU(\mathbf{a})$. However, while the allocation $\mathbf{a}'$ maximizes the regret of $\mathbf{a}$ (specifically at $UBU(\mathbf{a})$), there are (infinitely) many other utility vectors at which regret is maximized by $\mathbf{a}'$ as well. The use of $LBU(\mathbf{a}')$ is the vector that gives the minimax regret IP (Eq. 5) the least flexibility, thus ensuring the most rapid progress. Computational experiments using other choices of utility vector for the constraints verify this observation: the generation procedure does not converge nearly as fast using choices other than $LBU(\mathbf{a}')$.

4 Elicitation Strategies

We now turn to the question of elicitation: how should the provisioner determine which points to sample in order to find a high quality solution? We consider both regret-based methods and a more classic optimization approach.

Regret-based Elicitation

Assume the provisioner has a set S of sampled utility points from the WMs, and has computed a minimax optimal allocation $\mathbf{a}$ (or some approximation thereof). If regret level $MR(\mathbf{a})$ is unacceptably high, it can ask utility queries of any of the WMs to obtain additional sampled utility points. In this section, we consider several variants of the elicitation strategies proposed in [1] and provide systematic experiments. This is made possible only because the constraint generation procedure makes computation of minimax regret feasible.

We briefly describe the two strategies proposed in [1]. *Halve-all-bins (HAB)* is theoretically motivated and proceeds at each iteration by asking each WM for its utility at the sample points that lie midway between the current sampled points (thus it uniformly reduces the size of the "bins" by half at each iteration). A second strategy is the *current solution strategy (CS)* (called "heuristic split" in [1]): this strategy restricts queries to lie in bins containing either the current (minimax optimal) solution $\mathbf{a}$ or the adversarial witness $\mathbf{a}^w$ (that "proves" the max regret of $\mathbf{a}$). Intuitively, this strategy provides great potential to reduce minimax regret by either increasing the lower bound on the utility of $\mathbf{a}$ or decreasing the upper bound on $\mathbf{a}^w$. Rather than querying both bins (since evaluation of a query by a WM is expensive), CS chooses to query WM i either in the bin in which a_i lies, or a_i^w lies, not both. The choice of bin is determined by "size"—whichever has the largest (normalized) sum of length and height (since larger bins are likely to offer a significant change). The chosen bin is queried at its midpoint.

We consider several variants of these methods. One strategy is *halve-largest-bin (HLB)*, in which each WM is queried at the midpoint of the largest utility bin (given the current set of samples). Intuitively, this focuses elicitation effort much more than HAB, but does not have the computational requirements of CS, since one need not compute minimax regret to implement this method. Note, however, that minimax solutions will need to be computed to determine which allocation to offer, and when to stop asking queries. If the computational burden on WMs for evaluating utility at sampled points is severe (as in our motivating examples), savings in regret computation can be damaging if it causes more queries to asked. We consider two variants of HLB: *HLB-sum* uses the sum of (normalized) length and height of a bin, while *HLB-prod* uses the product (i.e., normalized area). We also consider sum and product variants of the CS method. Finally, we examine the effect of approximation on CS. Specifically, we investigate the performance of *CS-sum-k* and *CS-prod-k*, where a time bound of k seconds is imposed on minimax regret computation after each query. While we may approximate minimax regret, we hypothesize that approximate solutions will still provide good guidance for query selection, but faster.

Nelder-Mead Optimization

The Nelder-Mead algorithm [6] is a well-established approach to optimizing continuous functions, and we adopted an implementation similar to that in Sec. 10.4 of [8]. The algorithm forms a simplex in n dimensions using $n+1$ points, with the function evaluated at each point. It gradually expands, contracts, and moves the simplex by selecting new candidate points, as defined by a simple set of rules. Eventually, the simplex may contract around an optimal point. As with hill-climbing methods, Nelder-Mead can converge to plateaus or local optima unless restarting is used.

Nelder-Mead (and indeed any derivative-free algorithm) can be applied to elicitation by distributing function evaluation among WMs. The provisioner selects each candidate point $\mathbf{a}$ for the simplex as a feasible resource allocation, and

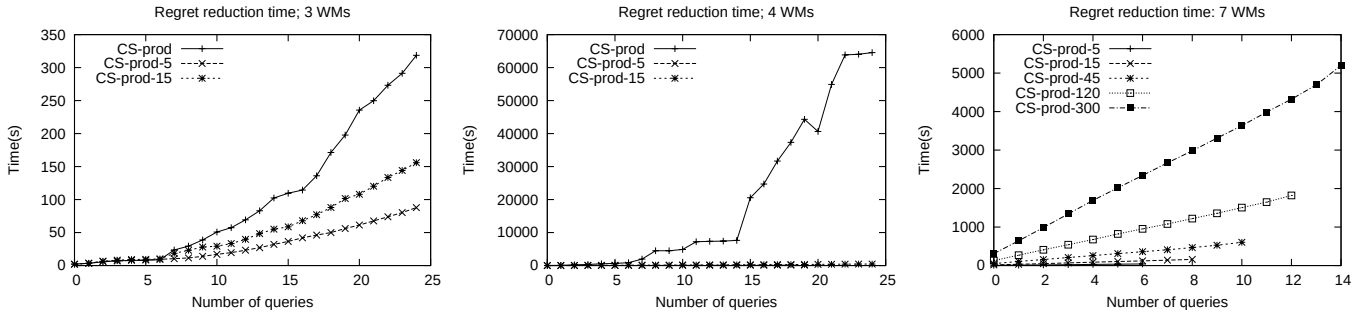

Figure 4: Cumulative computation time: (a) 3 WMs; (b) 4 WMs; and (c) 7 WMs.

to evaluate $\mathbf{a}$, queries each WM for $u_i(a_i)$. The provisioner maintains the highest-value allocation observed as the solution. Although it does know the exact utility of each candidate solution, Nelder-Mead does not compute any bounds, and gives no guarantee on the quality with respect to optimal. In its favor, provisioner computation is much faster than when computing minimax regret. However, there is an interesting trade-off between provisioner and WM cost as Nelder-Mead tends to require significantly more elicitation.

Empirical Results

In this section we describe the results of our elicitation strategies for a data center model with multiple WMs. We studied configurations where each WM handled two transaction classes and the QoS level in each class specified payment as a function of response time. Each of these functions was a smoothed out step function, with high payment for response time below a threshold, and zero payment above the threshold. For a fixed level of resource, a WM controls the response time of each class through the fraction of available resource assigned to that class. Given the constant average class arrival rate, we employed a simple M/M/1 queue to model the average response time. For our simulations, we constrain the resource levels to lie on a discretized grid of 1000 points in the unit interval. This discretization makes it easier to compute an individual WM's maximal utility for a given resource level, and also eliminates floating-point roundoff errors.

We conducted simulations with three, four, and seven workload managers, each having a different utility function. Each experiment started with a randomly chosen known sample utility point, and proceeded to obtain more points through elicitation, until the minimax regret dropped either to zero or some small threshold. Although some of our elicitation procedures do not need to calculate minimax regret, we report it here for ease of comparison. The simulations were repeated ten times and we report the median run. For seven WMs we refrained from calculating the exact minimax regret (due to long computation times) and performed elicitation with various time limits (5s, 15s, 45s, 120s, 300s) to demonstrate that one need not necessarily compute the exact solution in order to determine useful queries to quickly reduce regret.

When eliciting utility information, bin sizes inevitably become smaller; to avoid numerical instabilities we put a lower bound of 10^{-5} on the size of all bins.[2] We use the same 10^{-5} bound as the integer solution tolerance for the MIP solver. All simulations were run on Intel Xeon 2.4GHz machines using CPLEX 9.0 as our solver.

Fig. 3 shows the true max regret of the discovered solution as a function of the number of queries, demonstrating how quickly the various approaches reduce regret.[3] Fig. 4 shows cumulative run times for computing minimax regret (or max regret for methods that do not compute minimax optimal solutions after each query). HAB (tested on 3 and 4 WMs, Fig. 3(a) and (b)), though theoretically motivated, is not able to reduce regret as quickly as HLB or the CS methods; even the severely time-bounded method CS-5 does much better. There was virtually no difference between the *sum* and *prod* variants of either CS of HLB, demonstrating that both forms choose very similar queries (for simplicity we only show the *prod* variant). In the 3 and 4 WM cases, CS (in its various forms) outperforms HLB. Interestingly, the effect of approximation on the CS strategy (in CS-5 and CS-15) is barely noticeable. Despite approximating the solution to the minimax regret problem, the suggested allocations have max regret very near optimal. More importantly, CS-5 and CS-15 direct the choice of queries (which in CS is dictated by the current solution) as well as the unbounded CS. In terms of computation time, obviously the time-bounded methods are the fastest, while the elicitation procedures which compute the exact minimax regret obviously scale much worse (see Fig. 4(a) and (b) for 3 and 4 WM results). Note that HAB and HLB need not compute minimax solutions to determine which query to ask; but they must do it at any iteration in which a solution is to be offered.

Figs. 3(c) and 4(c) show results for the much larger seven-WM problem.[4] One can see that the CS strategy managed to reduce regret quickly with time-bounds as small as five seconds; furthermore, these results are nearly as good as those obtained with much longer computation times. (Solve time scales roughly linearly with the number of queries.)

[2] Even so a few runs had to be terminated due to the propagation of numerical errors. We are currently investigating the cause.

[3] Max regret is plotted when time-bounds were imposed, minimax regret otherwise. In the former case, the best solution is reported (one should save the best solution found after *any* query).

[4] We omit plots for the *sum* variant because they were almost identical to the *prod* based procedures.

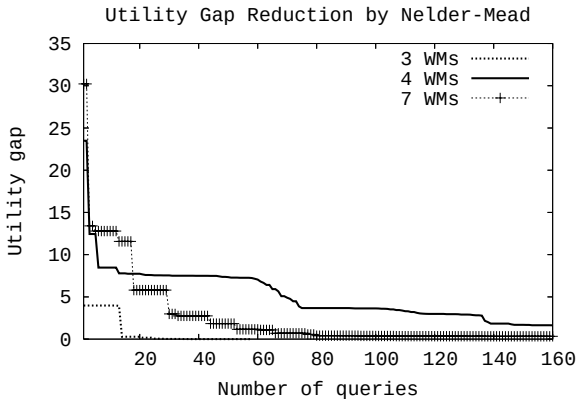

Figure 5: Utility gap reduction by Nelder-Mead.

We note that computationally effective strategies are critical if acceptable solutions are to be achieved with a minimal number of queries. While computation times are not reported in [1], we can see the effect of approximation in that work. Specifically, on the same four-WM problem as tested here, using the CS elicitation strategy, they reach a max regret of roughly 20 after five queries, and 5.5 after ten queries. This is due to the fact that they only approximate minimax regret computation (though they do show exact max regret). In contrast, even with approximation, we find regret levels of roughly 7 to 8.5 (depending on the degree of approximation) after five queries, and 2.0 after ten. The ability to solve the minimax problem exactly, and get good approximations with severe time bounds of 5–15 seconds has a dramatic impact on the ability to propose good solutions and queries.

In Fig. 5 we show the *utility gap* reduction by Nelder-Mead for 3, 4, and 7 WMs. The utility gap is the difference in value between optimal and the best solution found by Nelder-Mead.[5] We ran it without restarts and show the best runs obtained. The regret-based approaches reach zero so quickly that they cannot be visibly plotted against Nelder-Mead. For 3 WMs, Nelder-Mead achieves a gap of under 0.3 after 14 queries and under 0.05 after 25 queries, and CS-5 performs similarly. For 4 WMs, Nelder-Mead is at roughly 1.6 after 160 queries; for 7 WMs, it reaches 0.5 after 66 queries and 0.4 after 95 queries. CS-5 reaches zero (to within tolerance) after just 13 queries for 4 WMs and 7 queries for 7 WMs.

By contrast, a provisioner using Nelder-Mead requires insignificant time (under 1 microsecond) to select the next candidate allocation. This offers the opportunity to explore the provisioner/WM time trade-off compared with CS-5; e.g., we might view the best approach as that with the lowest total runtime. Since the best Nelder-Mead run requires about the same number of queries as CS-5 in the 3 WM case, its total runtime is faster than CS-5. However, for 4 and 7 WMs, Nelder-Mead would be competitive only if queries take under 1s. of WM computation. Indeed, this analysis is biased in favor of Nelder-Mead in that we chose the best runs, while using median runs for CS-5. These results suggest that there is a clear advantage to our regret-based approach—except perhaps for small problems—when elicitation is indeed costly.

[5]Recall that Nelder-Mead does not compute regret.

5 Concluding Remarks

We have provided a new computational procedure for computing minimax regret and determining robust allocations of resources in distributed autonomic systems. The use of a direct IP formulation and constraint generation allows allocations to be determined significantly faster than earlier approaches, and lends itself to approximation due to its anytime nature. We have also shown that this approximation has a negligible effect on the choice of good queries in the utility elicitation in which a provisioner must engage. This is critical since our aim is to minimize the number of (expensive) utility evaluations WMs must perform. We observed a provisioner/WM time trade-off between our approach and the Nelder-Mead optimization method, with the regret-based approach running faster overall when WM queries are costly.

Future research directions include further development and study of approximation techniques and new elicitation heuristics. While our model generalizes to multidimensional utility when multiple resources are at stake, we must explore the impact on our computational and elicitation methods. We have begun exploration of a sequential model of resource allocation in which demands on WMs change over time, and resources can be reallocated. If reallocation costs or delays can be incurred, optimal resource allocation must be based on sequential policies. Finally, we are exploring elicitation strategies for Bayesian optimization criteria.

References

[1] C. Boutilier, R. Das, J. O. Kephart, G. Tesauro, and W. E. Walsh. Cooperative negotiation in autonomic systems using incremental utility elicitation. *19th Conf. on Uncertainty in AI*, pp.89–97, Acapulco, 2003.

[2] C. Boutilier, R. Patrascu, P. Poupart, and D. Schuurmans. Constraint-based optimization with the minimax decision criterion. *Ninth Intl. Conf. on Principles and Practice of Constraint Programming*, pp.168–182, Kinsale, Ireland, 2003.

[3] U. Chajewska, D. Koller, and R. Parr. Making rational decisions using adaptive utility elicitation. *Proc. Seventeenth National Conf. on AI*, pp.363–369, Austin, 2000.

[4] B. Dantzig, R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

[5] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–52, 2003.

[6] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[7] D. Pescovitz. Autonomic computing: Helping computers help themselves. *IEEE Spectrum*, 39(9):49–53, 2002.

[8] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge Univ. Press, 1992.

[9] T. Wang and C. Boutilier. Incremental utility elicitation with the minimax regret decision criterion. *Proc. 18th Intl. Joint Conf. on AI*, pp.309–316, Acapulco, 2003.

[10] C. C. White, III, A. P. Sage, and S. Dozono. A model of multiattribute decisionmaking and trade-off weight determination under uncertainty. *IEEE Transactions on Systems, Man and Cybernetics*, 14(2):223–229, 1984.