Utility-Function-Driven Resource Allocation in Autonomic Systems

Gerald Tesauro, Rajarshi Das, and William E. Walsh and Jeffrey O. Kephart
IBM TJ Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532, USA

Abstract

We study autonomic resource allocation among multiple applications based on optimizing the sum of utility for each application. We compare two methodologies for estimating the utility of resources: a queuing-theoretic performance model and model-free reinforcement learning. We evaluate them empirically in a distributed prototype data center and highlight tradeoffs between the two methods.

1. Computing Resource Value

Traditionally, massive overprovisioning of resources has been used to protect computing systems from spikes in demand. Dynamic resource allocation allows an autonomic computing system to self-optimize its resource usage and avoid the inefficiencies of over provisioning. In order to dynamically allocate resources among multiple applications, it is necessary to accurately and automatically compute the value of resources for each application. To motivate this problem, consider a realistic, distributed prototype data center, described previously in [4]. The data center consists of multiple, logically separated *Application Environments* (AEs), each providing a distinct application service using a dedicated, but dynamically allocated, pool of servers. Each environment has its own service-level utility function $U_i(\mathbf{T}_i)$, specifying the utility to the data center (evaluated on a common scale, such as money) from environment i as a function of service metrics $\mathbf{T}_i$.

Detailed control within an AE is handled by an associated *Application Manager* (AM). At the higher level, a global *Resource Arbiter* allocates resources across different AEs. Each AM sends to the arbiter a *resource-level utility function* $V_i(R_i)$ that specifies the value to the AE of obtaining each possible number R_i of servers. Given the current functions, the Arbiter computes maximizes systemwide serice-level utility as by computing allocation $R^* = \arg\max_R \sum_i V_i(R_i)$ s.t. $\sum_i R_i = \bar{R}$, where $\bar{R}$, where $\bar{R}$ is the total number of servers. Each server is allocated as a whole unit and cannot be be shared among AEs.

Our prototype data center is implemented on a cluster of identical IBM eServer xSeries 335 machines running Redhat Linux. We run two kinds of applications in separate AEs. The "Trade3" application is a realistic simulation of an electronic trading platform, which we drive with demand λ generated by an open-loop Poisson mode with variable mean adjusted according to [2]. We define $\mathbf{T}_i$ as the average response time τ_i. The "Batch" application handles a long-running batch workload and we define $\mathbf{T}_i = R_i$ for it.

A key challenge is for the AM to dynamically compute $V(R)$ for Trade3 in response to changing demand and relevant internal state. A well-established methodology to controlling systems is *model-based approach* based on queuing theory [1]. In our system, we model a Trade3 AE with R servers as R parallel M/M/1 queues. To account for the transient nature of the workload as well as the variance in the performance of servers due to garbage collection and thread management in Java, the AM updates the estimated parameters of the model at periodic intervals. With known τ^t and R^t in time period t and possible R^{t+1} servers in time period $t+1$, we have $\hat{\tau}^{t+1} = 1/\hat{\mu}^t - \lambda^t/R^{t+1}$, where $\hat{\mu}^t = 0.5\,\hat{\mu}^{t-1} + (1-0.5)\,\mu^t$ and $\mu^t = 1/\tau^t + \lambda^t/R^t$. We introduced the μ to $\hat{\mu}$ smoothing transformation to avoid excessive fluctuations. With these equations, the AM computes $V(R^t) = U(\hat{\tau}^t)$ for each possible value of R^t.

In the model-free reinforcement learning (RL) approach, the Trade3 application manager uses an algorithm known as Sarsa(0) to learn a value function $Q(s,R)$ estimating the long-range expected value of an allocation of R servers in local state s. The learning rule has the form $\Delta Q(s^t,R^t) = \alpha[U^t + \gamma Q(s^{t+1},R^{t+1}) - Q(s^t,R^t)]$, where U^t is the immediate "reward" i.e. the instantaneous service-level utility in the current state, (s^{t+1},R^{t+1}) denotes the state and number of servers at time $(t+1)$, the constant $\gamma = 0.5$ is a "discount parameter" expressing the present value of expected future reward, and α is a "learning rate" parameter which is initialized to 0.2 and decays over time.

Potentially many sensor readings may be needed to accurately represent the application's state, but for simplicity we use only current demand, i.e., $s^t = \lambda^t$. The $Q(s,R)$ function is represented as a two-dimensional grid, with demand discretized into 130 intervals of size 2.5 over the range $0-325$. The number of servers R ranges from 1 to 5, so the total size of the value function table is 650. The cell values are initialized using a simple heuristic which decreases linearly with demand per server: $Q_0 = 200 - 1.2\lambda/R$.

RL procedures also require an "exploration" mechanism to ensure that all value table cells are visited sufficiently often. We find that a simple rule of having the arbiter occasionally choose (with probability 0.1) a random allocation suffices to give RL its needed exploration, while incurring only negligible loss of system utility. To further improve the valuations of infrequently visited cells, we use soft monotonicity constraints that encourage cell values to be monotone increasing in R and monotone decreasing in D (both reasonable assumptions in our system).

We note that standard RL convergence proofs do not apply in our system, as the arbiter's allocation policy is neither stationary, nor does it optimize any individual application. However, good empirical evidence for approximate convergence is found in our experiments. The issue of whether localized RL converges in such systems is addressed in [3].

2. Results and Discussion

Figure 1(a) compares the performance of RL and queuing model approaches in the standard two-application scenario (Trade3+Batch), using identical demand generation in each run. Performance is measured over the entire run in terms of average total system utility earned per arbiter allocation decision. We also compare with two inferior allocation strategies: "UniRand" does uniform random allocations; and "Static" denotes the best static allocation (three servers to Trade3 and two to Batch). Also shown is a dashed line indicating an analytical upper bound on the best possible performance that can be obtained in this system using the observable information.

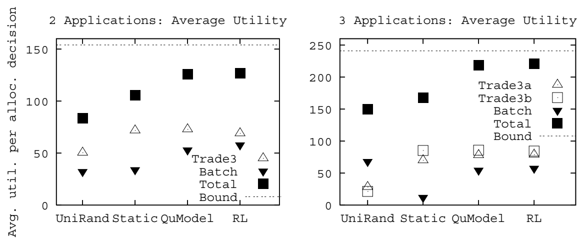

Figure 1. Results using 2 and 3 applications.

Note that RL performance includes all learning and exploration penalties, which are not incurred in the other approaches. The model-based and model-free approaches are virtually identical, despite their radically different natures, and both are substantially better than static or random allocations. They are also reasonably close to the maximum performance bound.

We have also studied a more complex scenario containing three applications: one Batch plus two separate Trade3 environments, each with an independent demand model. The scenario is more challenging for RL in that there are now multiple interacting RL modules, each of which induces non-stationarity in the other's environment. However, using the same initialization as before, we observe no qualitative difference in RL training times, apparent convergence, or quality of policy compared to previous results. Performance results in this scenario are plotted in Figure 1(b). Once again RL performance is comparable to the queuing model approach, and both are quite close to the maximum possible performance. While this does not establish scalability to an arbitrary number of applications, the results are encouraging regarding our general methodology.

Whether such success with RL will continue as we examine progressively more complex systems is of course an interesting open research question. We fully expect that nonlinear function approximators, such as neural networks or support vector machines, will eventually be required to represent RL value functions. We also expect to need explicit techniques for obtaining acceptable performance levels during training, and for intelligent exploration of non-greedy actions. A particularly promising technique, is hybrid training, in which the policy decisions in the initial phases of learning are model-based. This can provide not only a strong initial policy, but can also provide safety bounds on exploration.

In addition to performance, we identify at least two additional aspects of importance to autonomic computing systems. One is the amount of systems knowledge required for success with each method. RL appears to have the initial edge over queuing models in this regard, and it will be interesting to see if this persists in subsequent work. Secondly, there is the issue of brittleness of performance models and learned value functions under various forms of environment or system changes. Certainly some form of system changes, such as changing the service-level utility function, will not require any queuing model changes, whereas RL might need to be retrained from scratch. On the other hand, certain changes in user behavior may lead to gradual "model drift" in which the queuing model progressively becomes less accurate. If this drift is slow, RL might be able to continually adapt the value estimates so that they maintain their accuracy as conditions change.

References

[1] D. Menasce, V. Almedia, and L. Dowdy. *Performance by design: Computer Capacity Planning by Example*. Prentice Hall, Upper Saddle River, NJ, 2004.

[2] M. S. Squillante, D. D. Yao, and L. Zhang. Internet traffic: Periodicity, tail behavior and performance implications. In E. Gelenbe, editor, *System Performance Evaluation: Methodologies and Applications*. CRC Press, 1999.

[3] G. Tesauro. Decompositional reinforcement learning and workload management. Submitted for publication, 2005.

[4] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *1st IEEE International Conference on Autonomic Computing*, pages 70–77, 2004.