HIDRA: HISTORY BASED DYNAMIC RESOURCE ALLOCATION FOR SERVER CLUSTERS

Jayanth Gummaraju[1] and Yoshio Turner[2]

[1]Computer Systems Lab., Stanford University, Stanford, CA, USA
jayanth@stanford.edu
[2]Hewlett-Packard Labs., Palo Alto, CA, USA
yoshiotu@hpl.hp.com

ABSTRACT

Internet services can experience large time-variations in client demand. We propose Hidra, an online approach for dynamically determining the minimum number of servers needed to meet a service's high-level performance objectives. Hidra does not depend on knowledge of system architecture details. Thus, it can be applied to a wide variety of services, even if their execution paths or workload mix change frequently. Hidra incrementally constructs a history-based model to characterize the system's performance and predict resource requirements. To incorporate new performance measurements and discard obsolete information Hidra uses two metrics, ***freshness*** *and* **confidence***. We present preliminary results, based on simulation of single- and multi-tier systems, to demonstrate that Hidra can achieve nearly optimal levels of resource allocation, even for services that exhibit distributed caching effects and overheads due to communication.*

1. INTRODUCTION

The level of client demand for an Internet or Enterprise service typically has large time-variations with a high peak to average ratio [1]. Dynamic resource allocation has been proposed to exploit this property to reduce the cost of supporting services [2][3]. In a shared infrastructure, resources that are not needed by one service can be reassigned to other services or deactivated to save operational costs such as electric power costs or software license fees.

A key challenge for dynamic resource allocation is to determine on the fly the minimum amount of resources needed to meet a service's performance requirements as client demand changes. This problem is particularly challenging given frequent updates to a service and its implementation (e.g., the service's static and dynamic content, middleware and OS configuration and patches, and shifts in client interest). Thus, it is infeasible to base resource allocation on static capacity planning models or on detailed a priori models of the workload, software, and hardware components such as CPUs, memory, disks, and network interfaces.

We propose Hidra, a system for automatically and dynamically determining the number of servers needed to meet a service's performance requirements. Our focus is on resource allocation for services in which one or more tiers (e.g., web, application server, or database) is implemented on a scalable cluster of commodity machines. We restrict our attention to services large enough to require multiple machines in the cluster to meet their performance goals. As a result, Hidra currently determines resource allocation levels at the granularity of machines.

Hidra uses a "black-box" approach that monitors only changes in service-level performance metrics that are visible externally to the system. It incrementally builds a simple empirical history-based performance model of the system to predict performance with sufficient accuracy

for dynamic allocation without the need to model detailed aspects of a service's implementation. The model's generality enables practical deployment of Hidra for a wide variety of services.

This paper makes two main contributions. First, we present Hidra for single-tier services (e.g., a cluster of web servers, or a cluster of J2EE application servers). Current and past measurements of system performance are used to predict the future performance characteristics and a desirable resource allocation to meet future client demand. To improve prediction accuracy, collected history information is weighted using two attributes we introduce: *freshness* (related to the age of a performance measurement) and *confidence* (the degree to which a history record agrees with current performance measurements). Since the model may have incomplete information when an allocation decision is needed, we devise extrapolation techniques to determine a desirable resource allocation. These techniques exploit knowledge of the general shape of a response time function to identify the most useful history information. We evaluate the scheme on a cluster simulation environment that provides the control and observability needed for detailed analysis of the scheme's behavior. In particular, we evaluate Hidra for services exhibiting distributed caching effects and overheads due to communication and synchronization.

The second contribution is an extension of the single-tier solution to multi-tier services. The multi-tier scheme determines the appropriate number of servers in each tier (e.g., web and application service tiers). We expect our scheme could also be applied to scalable clustered database tiers [4].

2. Related Work

Dynamic resource allocation schemes differ in several dimensions including monitoring metrics, a priori knowledge of the system behavior, allocation granularity, and control policies. In this section we describe the unique position of Hidra in comparison to previous research efforts. Hidra was designed to avoid system assumptions and to base predictions on measurements of performance metrics that are visible externally to the system: request rate, reply rate, response time, and error rates. The scheme does not need to know, for example, which low-level resource (e.g., CPU, disk bandwidth, or logical resources such as IP ports) is the bottleneck that limits performance [5].

In contrast, some previous schemes allocate resources by controlling the utilization of a particular low-level resource that is assumed to be the bottleneck resource [2][6] The utilization level of this resource must be translated to the actual performance metrics that are of interest to the user (i.e., throughput and response time). However, this can be challenging especially if the mapping is subject to change over time or if a different resource becomes the bottleneck.

Hidra incrementally builds an empirical, high-level model of system performance and uses it in an online feedback control loop to determine resource allocation requirements. Hidra can be applied even to services that undergo frequent changes in implementation and workload. In contrast, several previous schemes rely on service-specific models (e.g., queueing models) of performance or workload constructed in advance of service deployment [7][8][9][3][10]. To expand the generality of model-based approaches, it may be possible to use online measurements to calibrate the models [11]. Hidra's model is simple to understand and can easily be extended by system administrators to incorporate information from service-specific models to improve prediction accuracy. Hidra allocates resources at the granularity of individual "machines". We assume the machines in a service tier all equivalent (e.g., homogeneous blades within a blade server). Allocation at machine granularity matches our focus on large services and trends towards better price-performance for clusters of small commodity machines than for single large machines. Also, it naturally meets the practical need to provide isolation of performance, security, and faults between disparate services at low cost.

Several previous efforts have allowed allocation at finer granularity than a machine [11][8][12][7][10]. To get better isolation between services that share a machine would require using OS mechanisms for resource partitioning [13] or a virtual machine (VM) monitor that can partition a physical machine's resources among multiple VMs hosting different services. Although Hidra does not prescribe how to change the physical resource allocation of a VM, it could allocate at the granularity of VMs which are each given equivalent portions of the resources of a physical machine. However, high performance overheads of VMs can preclude their use for deployed services on small physical machines.

In addition to dynamic resource allocation, admission control can be used to react to variations in demand [14][15]. Admission control denies some transactions admission to a service when the level of demand approaches an estimate of the system's capacity. In contrast, dynamic resource allocation adjusts the resources to match fluctuations in workload without limiting demand. When dynamic resource allocation cannot find free resources to allocate to a service prevent poor performance for clients that are accepted into the system. Finally, policies are needed to make allocation decisions based on the relative importance of services.

Service importance can be quantified with utility functions that describe economic factors such as service revenue and the cost of violating service agreements [8][10]. In addition, services may compete for resources that are priced dynamically in a bidding process [2].

3. SINGLE-TIER HISTORY-BASED RESOURCE ALLOCATION

Hidra estimates the minimum resource allocation that would keep mean server response time below a user-specified threshold. This is a critical piece of a complete dynamic resource allocation solution which would additionally include automated mechanisms for changing the resource allocation by reconfiguring and loading software onto servers and other components [16][17], monitoring performance and client demand (e.g., by monitoring server logs), and predicting the client request rate in the near future based on current trends [11][2].

Hidra builds a model that predicts the response time for different client request rates and resource allocations. To match our focus on server allocation, the targeted performance metric is the response time of the servers, excluding the network latencies between clients and the server site. The model is updated periodically using measurement of the client request rate and mean response time over a sampling interval of a few minutes. Hidra uses the model, together with a prediction of the near future level of applied load, to determine a new level of resource allocation. The real response time function for a system can change slowly over time, for example as new content is gradually added to a service. In addition, at relatively infrequent times the function can change suddenly. It may rapidly shift to a new steady state, for example after a major software upgrade, or it can have transient deviations caused by temporary factors such as a software glitch or a short series of unusual client requests. At time instants in which the function is changing slowly, and at time instants just after infrequent shifts and transients, the current system behavior is a good predictor of the immediate future behavior. This motivates Hidra's use of recent history information to construct its model.

3.1. History-Based Model

Hidra's model for a single-tier system consists of a graph of response time as a function of λ, the per-machine applied load, which is calculated as the service's total client request rate (R requests/sec) divided by the resource allocation (N machines). To determine the resource allocation, Hidra first finds the desired operating point as the maximum value of λ which meets the response time requirement. Using the predicted client request rate R, Hidra converts λ to the corresponding resource allocation $N = R/\lambda$. An advantage of basing the model on the average

applied load across the allocated machines is that the scheme works without modification for both uniform and non-uniform (e.g., content-aware) distribution of requests across the cluster. Although in this paper we focus on mean response times from the servers, Hidra's model could easily be extended to describe additional functions of λ, such as the 95-percentile tail of the response time probability distribution by monitoring the corresponding performance metrics.

Hidra maintains a collection of history records which describe a set of points on the graph of response time versus λ. For each sampling interval, the measured mean response time is used to update the history record that corresponds to the applied load λ over the sampling interval. Hidra could record points of additional functions of λ in the history, such as the reply rate and the rate of errors that were observed. Typical shapes for these curves are illustrated in Fig. 1. At low levels of applied load, the reply rate equals the request rate; the response time is short and lengthens only slowly with the applied load. As the applied load increases further, the response time rises steeply as the system approaches its capacity. Above saturation some requests result in errors such as connection refused, and as the error rate increases, the reply rate can level off or even decrease as system resources are increasingly wasted processing error cases rather than generating successful replies. In saturation, the response time for successfully handled requests can continue increasing or level off as queue depths reach their maximum limits.

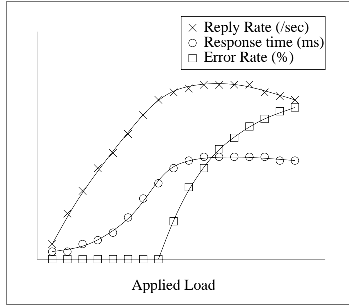
3.2. History Update: Freshness and Confidence

Updating the history requires balancing a trade-off between accuracy and responsiveness. New information must be incorporated effectively into the history records or else the model will fail to reflect the current state as the system undergoes gradual changes. However, if history information is removed too easily, transient glitches in the performance characteristics can pollute the model. Our approach is to weight history information with the aid of two factors we term *freshness* and *confidence*. *Freshness* quantifies the notion that the point value stored in a history record likely becomes less accurate as time passes without the value getting updated. Thus the freshness value for a point is computed according to a decreasing function of the time that has elapsed since the history point was last updated.

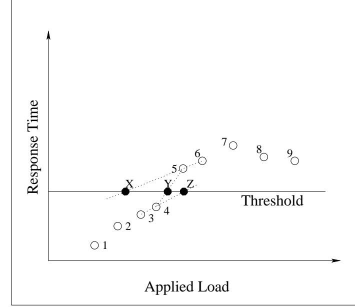
Confidence is a measure of how consistently a history record seems to characterize the current behavior of the system. A point in the history may represent an obsolete system behavior even if it was updated recently and therefore has high freshness. This can occur when the history record reflects measurements taken prior to a recent shift to a new steady state or during a transient operating condition that is no longer in effect. It is desirable to quickly discount such history and replace it with new values. Therefore, the confidence value for a history record increases (up to a maximum bound) as new measurements provide confirmation of the stored value, and it decreases when new measurements show substantial difference from the stored value. Each new response time sample is compared with the average response time recorded in the corresponding history record. Confidence is increased if the new sample is close to the recorded value (within 10% in our experiments), and is decreased otherwise. For a history point we define a weight α that ranges from 0 to 1 and is a composition of the point's freshness and confidence values.

A history point is updated to the weighted average of the point's stored value and the new value that is reported by performance monitoring: stored_value = α * stored_ value + (1 - α) * new_value. The value of α is computed according to a function which decreases over time since the point was updated (freshness) and which decreases more rapidly for points with low confidence value than for points with high confidence value. For our experiments, we chose a definition for α that results in history points computed as exponentially weighted moving averages with a decay rate that slows down as the confidence value increases.

3.3. Determining Resource Allocation



(a) Typical System Characteristics



(b) Computing an operating point

Figure 1: Extrapolation approach

Hidra must extrapolate the partial information in the history to estimate the desirable operating point (λ,y) on the response time curve. Hidra attempts to use only history points that reflect the current state of the system by ignoring any point with an α value below a threshold (since α is a measure of the importance of a history point). In addition, as described below, Hidra uses only points that are consistent with the shape of the typical response time curve (Fig. 1(a)).

Fig. 1(b) depicts how Hidra computes the operating point in different scenarios. Various example history points are shown along with the user-specified response time threshold. The desired operating point is the extrapolated point that intersects the threshold. To find this point, the algorithm first finds a set of points forming a positive slope, matching the expected shape of the response time curve. To determine this set uniquely, the algorithm favors keeping points with higher α values which are more likely to represent the current system state. Piecewise linear interpolation is used to find the intersection with the threshold. For example, Point X would be determined by extrapolation if only points above the threshold were available, whereas the result would be Y with all points available, and Z if only points below the threshold were available. Point pairs such as points 7 and 8 are never considered together in extrapolation since they form a negative slope. When the history has only one valid point, which corresponds to the current operating point, a simple heuristic reminiscent of control theory is used. The heuristic adjusts the resource allocation by an amount proportional to the difference between the response time for the current operating point and the response time threshold.

Finally, we impose a limit on the amount by which the resource allocation can change in consecutive resource allocation periods. This is motivated by the decreased accuracy of extrapolation to operating points that are far away from the points in the history. For each consecutive time interval in which the resource allocation is increased and the change limit has to be enforced, the change limit is increased to allow a larger resource change for the subsequent time interval. This enables resources to be added more rapidly to satisfy a rapid increase in resource requirements.

4. Experimental Evaluation: Single-Tier Resource Prediction

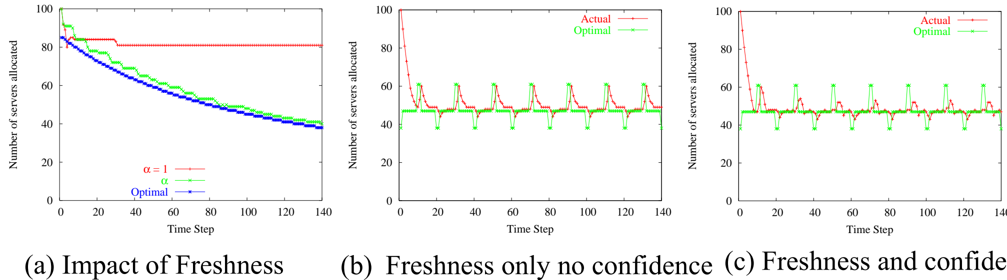
To gain insight into the potential of the Hidra approach, we developed a simulator which allows us to easily control and vary the behavior of a simulated cluster. The simulation is used to evaluate the contributions of each key element of Hidra to its overall performance.

4.1. Effectiveness of Freshness and Confidence

We applied Hidra to a simulated cluster to study the effects of *freshness* and *confidence*. Hidra determines the number of servers to be allocated during each interval of time, and the simulator

reports back to Hidra the performance that resulted from the resource allocation. At any time, the simulator models a server's response time as an increasing function of the applied load (as in Fig. 1(a)), and to model changes in system behavior and workload it changes this function throughout the experiment. For simplicity, the particular function we use corresponds to the response time of an M/M/1 queue with request rate λ and service rate μ. Although more complex queueing or other models could be used, they too would produce a response time function resembling Fig. 1(a) and our conclusions about Hidra's ability to estimate this function over time are likely to be the same. For the experiments in this section, we set a response time objective of 60 ms and the level of aggregate client demand to a constant value of 2000 req/sec.

To test the effectiveness of *freshness*, we conducted several experiments in which the cluster undergoes continuous gradual change in performance characteristics. We show one representative example in which each server has a capacity μ which increases steadily over time from 40 to 70 requests/sec. This could correspond, for example, to a gradual shift in the types of requests that are received by the service toward requests that can be processed more rapidly. For this simulated system, Fig. 2 shows the resource allocations that result when Hidra uses *freshness* to varying degrees. In addition, the figure shows the optimal resource allocation, which is determined through an exhaustive search in the simulation at each point in time.



(a) Impact of Freshness (b) Freshness only no confidence (c) Freshness and confidence.

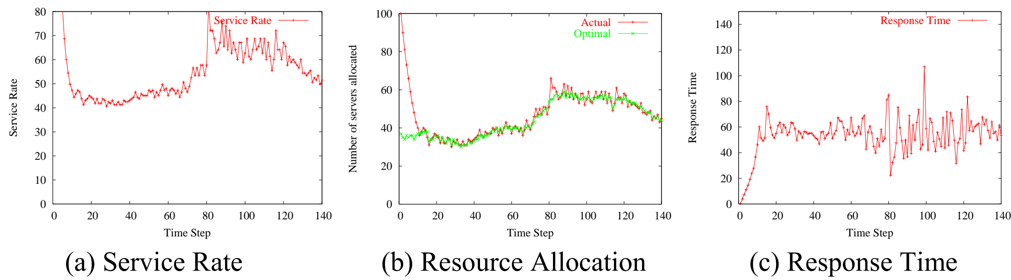
Fig. 2: Effectiveness of Freshness and Confidence. (a) Service rate increases with time. (b) & (c) Service rate kept constant (60) except for transient glitches every 10 time-steps.

The resource allocation when freshness is not used is shown in the curve α=1 in Fig. 2(a). Fixing α=1 causes each history record to be assigned a value only the first time the corresponding operating point is encountered, and operating point extrapolation uses all records, no matter how old. The result shows that without freshness, the scheme primarily uses data from the beginning portion of the experiment when the servers are slow (small μ), resulting in over-allocation throughout the experiment. The curve labeled α shows the effect of using freshness for both updating the history and for operating point extrapolation. In this case, the impact of obsolete information is reduced and allocation quickly converges toward the optimal value.

Confidence is useful when the system experiences a transient change in demand characteristics such as request type distribution. We simulate this effect by setting μ to be constant with time except for transient glitches every few time steps. As shown in Fig. 2(b), if confidence is not used, Hidra over-allocates by large amount immediately after each glitch. A transient glitch is not distinguished from a persistent change in system demand because transient values that enter the history records persist there just as long as values that reflect the normal behavior of the system, and even longer due to averaging effects for values with large deviation from the norm. By using confidence (Fig. 2(c)) the system preserves the more commonly observed history for a longer duration and discounts transients. Fig. 2(c) shows that the resulting resource allocation has a smaller reaction to transient glitches compared to the result in Fig. 2(b).

4.2. Non-Linear Cluster Scaling

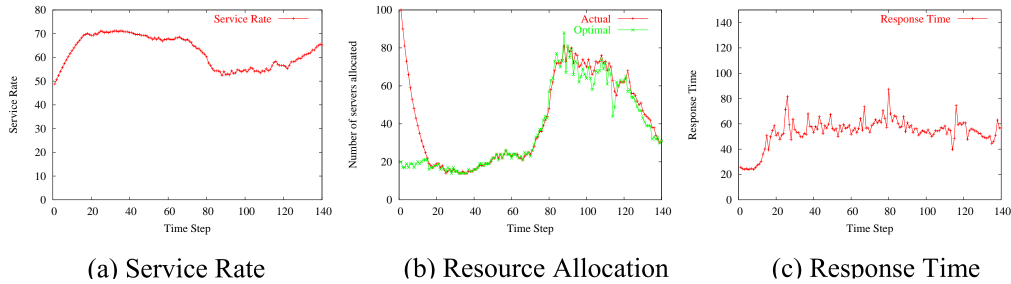
Hidra models response time as a function of server applied load. A server's behavior, as described by its response time as a function of its request rate, may be independent of changes in the resource allocation. However, in some scenarios the response time function for a server can depend significantly on number of servers that are allocated. This requires Hidra to adjust its model as the allocation varies. In this section, we examine two such situations. The first situation concerns content caching. With cooperative caching, in which machines pool their individual memories to create a single logical cache, the effective cache capacity increases as the cluster scales, leading to higher hit ratios and lower per-server response time. The second situation concerns communication and synchronization overhead. If machines in a cluster participate in distributed operations that involve locks and synchronization, the communication overhead and waiting times to acquire a lock can get worse as the cluster size scales. This in turn can cause the performance on a per-server basis to be reduced as the cluster scales. In the following experiments we vary the client request rate according to logs that were collected from the hp.com web service over a 24-hour period.



(a) Service Rate (b) Resource Allocation (c) Response Time

Fig. 3: Experiment incorporating caching effect.

We simulate a case where caching effectiveness increases with cluster size. Thus, Hidra faces the challenge of adapting to huge changes in a server's response time function throughout the experiment as the caching effect causes the capacity of each server (Fig. 3(a)) to vary by a factor of two. The simulator computes the hit ratio as a function H(N) which we define as a linear increasing function of N, the number of machines that are allocated. The simulator then calculates the value of μ according to the following formula: $1/\mu = H(N) * HitTime + (1 - H(N)) * MissTime$. The results in Fig. 3 show that even in this case where server behavior changes greatly with allocation, Hidra determines resource allocation (Fig. 3(b)) within 2% of optimal, and the response time (Fig. 3(c)) is close to the threshold value. This shows that Hidra's recent history records are usually adequate approximations of the server's current behavior, even if those records were updated when the system had a different resource allocation.



(a) Service Rate (b) Resource Allocation (c) Response Time

Fig. 4: Experiment incorporating communication effect.

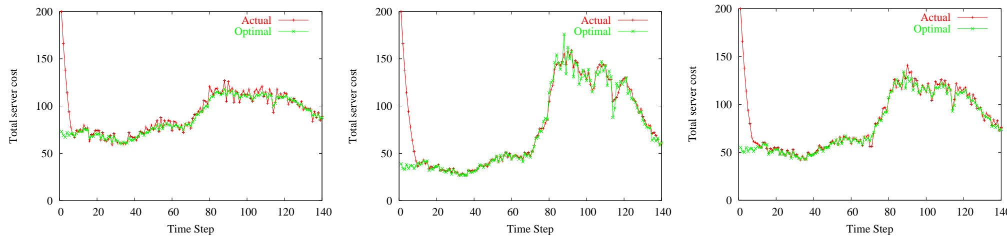
We next show an example of the cluster communication effect in which the server's response time function again changes drastically, but unlike for the caching effect each server becomes

less efficient with increasing allocation. In particular, we cause each server's capacity to vary by a factor of close to 1.5 by increasing the service time (1/μ) linearly with the number of servers allocated. The results in Fig. 4 show that the number of resources allocated is within 3% of the optimal resource allocation, and response time again stays close to the threshold. The average number of servers allocated (42) is one half the minimum adequate static allocation (88 servers).

5. MULTI-TIER RESOURCE PREDICTION AND EVALUATION

We extend Hidra to multi-tier systems where a client request may require execution in multiple tiers to generate a response. For example, a request to a front end web tier can trigger secondary requests to application servers in a second tier which must be processed before the front end can respond to the client. In some cases multiple logical tiers can be run on a common set of machines in a single physical tier, but it is often preferred to use separate physical tiers to simplify systems management and to secure confidential data by placing firewalls between tiers. The average response time for client requests is the sum of the average contribution from each tier, which depends on the path of execution for a request through the system and the resource allocation at each tier. Hidra should find the per-tier allocations that satisfy a user-specified response time threshold with minimized total cost of resources over all tiers. This problem is more complex than the single-tier case since various breakdowns of response time among the tiers can lead to similar total response time but very different total cost.

We assume that the system has an instrumentation or monitoring mechanism that reports the average contribution of each tier to response time. Suitable instrumentation mechanisms exist commercially for enterprise services [18]. In the future, it will likely be feasible to estimate response time contributions using tools that can infer execution paths and performance behavior of complex systems without using intrusive, detailed instrumentation [19][20].



(a) Caching effect both tiers (b) Comm. effect both tiers (c) Caching (tier 1) Comm. (tier 2)

Fig. 5: Multi-tier resource allocation. Costs of servers in both tiers kept same.

The proposed approach is to build for each tier a history-based model which tries to characterize the tier's contribution to total response time as a function of the tier's resource allocation. The algorithm adjusts the various tiers' target contributions to search for a configuration that would minimize total cost while meeting the overall response time objective. The history-based model for each tier constructs a graph that characterizes the behavior of a machine. This is similar to the single-tier case (Section 3) except that the metric is not the response time to requests that it receives, since these requests can be secondary requests that depend on the execution paths of client requests. Instead, each tier's model is a graph of the tier's contribution to the overall response time of client requests as a function of the load on each machine in the tier. We can express the load on a machine as the ratio of the client request rate divided by the number of machines in the tier. For example, suppose a two-tier system consists of a web tier front end and an application server tier, and that on average a client request arrives to the web tier which generates two secondary requests to the application server tier. Although the request rate to the application server tier is twice the request rate to the web tier, we consider each tier to be processing the same client request rate.

On each time step, the algorithm considers various ways to break down the desired total response time across the tiers. It slightly increases the target response time contribution from one tier and decreases the target response time contribution from other tiers by the same amount, such that the sum continues to equal the threshold for overall response time. As an example, for a two-tier system, the resource allocation is determined as follows. An array max_resp[2] is maintained to record the target response time contribution from each tier. The array is initialized such that the response time threshold max_resp_time is divided equally across the tiers, i.e., max_resp[i] = max_resp_time/2. For each time interval, the algorithm considers the resource predictions for alternative breakdowns of max_resp_time across the two tiers: 1) max_resp[0] and max_resp[1]; 2) max_resp[0]+ delta and max_resp[1] – delta; and 3) max_resp[0] - delta and max_resp[1] + delta, where delta is a small increment (we use delta = 10 milliseconds in our experiments).

The algorithm selects the alternative that predicts the lowest cost resource allocation. After each time interval, the algorithm obtains the measured or inferred response time contributions for each tier and updates the history table for each tier. We next evaluate the effectiveness of Hidra for the case where the two tiers exhibit similar scaling behavior in the form of the caching and communication effects that were introduced in Section 4.2. Fig. 5(a) shows the results when the caching effect is present in both tiers, and Fig. 5(b) shows the results when the communication effect is present in both tiers instead of the caching effect. The results show that each graph is similar to the corresponding single tier case from Section 4.2. This occurs because the two tiers are similar in both performance and per-machine cost. Multi-tier Hidra is able to rapidly identify that in this case the optimal operating point is where each tier contributes equally to response time. We next examine a more complex system where the two tiers have dissimilar scaling behavior. As before, the per-machine cost is equal for the two tiers. Fig. 5(c) shows the results when the caching effect is present in the first tier and the communication effect is present in the second tier. The graph resembles an "averaging" of the cases of caching effect (Fig. 5(a)) and communication effect (Fig. 5(b)). Hidra successfully adapts to the different behavior of each tier in the multi-tier system by finding the partitioning of response time that has the lowest cost. Finally, we also experimented with cases in which the tiers had different per-machine costs and different scaling behavior. The results, omitted due to space limitations, showed that Hidra continues to provide close to optimal resource allocation in this challenging case.

6. CONCLUSIONS

In this paper, we presented Hidra, a scheme for history-based dynamic allocation of servers for services implemented with single-tier and multi-tier scalable clusters. Hidra constructs an empirical model of system behavior based on service-level metrics of client request rate and response time. Hidra uses the *freshness* and *confidence* level of the collected history information along with the knowledge of the general shape of the response time versus applied load curve to predict the resource requirements. We believe this approach is well suited to managing services that exhibit changes over time in a variety of properties such as software versions, service functionality, and request type distribution. Currently, Hidra has been tested against a large simulated server environment for both a single and multi-tier environment. These experiments indicate that the scheme allocates resources within a few percent of the optimal allocation. In addition, the results show that the scheme can handle scenarios in which server performance characteristics change with the number of servers in a tier because of factors such as changing communication overhead or caching effects. Our simulation environment was not designed to model any particular system faithfully but is instead intended only as a vehicle for exercising and understanding the behavior of Hidra. In the future, we intend to deploy and further refine Hidra using a large cluster of machines running a multi-tier Enterprise service workload.

REFERENCES

[1] M. F. Arlitt and C. L. Williamson, "Internet web servers: Workload characterization and performance implications," IEEE/ACM Tr. Networking, vol. 5, no. 5, pp. 631-645, 1997.

[2] J. Chase et al, "Managing energy and server resources in hosting centers," in Symposium on Operating System Principles (SOSP), 2002.

[3] K. Appleby et al, "Oceano - SLA based management of a computing utility," in IFIP/IEEE Int'l Symp.. on Integrated Network Mgmt., 2001.

[4] T. Lahiri et al, "Cache fusion: Extending shared-disk clusters with shared caches," in Int'l Conf on Very Large Databases (VLDB), 2001.

[5] I. Cohen et al, "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in Operating Systems Design and Implementation (OSDI), 2004.

[6] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "QoS-driver server migration for Internet data centers," in Int'l Workshop on Quality of Service (WQoS), 2002.

[7] R. Doyle et al, "Model-based resource provisioning in a web service utility," in USENIX Symp. on Internet Technologies and Systems, Mar 2003.

[8] R. Levy et al, "Performance management for cluster based web services," in Int'l Symp on Integrated Network Mgmt., Mar 2003.

[9] T. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: a control-theoretical approach," IEEE Tr. Parallel and Dist. Computing, vol. 13, no. 1, 2002.

[10] H. Zhu, H. Tang, and T. Yang, "Demand-driven service differentiation in cluster-based network servers," in IEEE INFOCOM, 2001.

[11] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in Int'l Workshop on Quality of Service (IWQoS), 2003.

[12] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated resource management for cluster-based Internet services," in Operating Systems Design and Implementation (OSDI), 2002.

[13] G. Banga, J. C. Mogul, and P. Druschel, "Resource containers: a new facility for resource management in server systems," in Operating System Design and Implementation (OSDI), 1999.

[14] S. Parekh et al, "Using control theory to achieve service level objectives in performance management," Tech. Rep. RC 21844 (98315), IBM Research Labs, Sept 2000.

[15] P. Phaal, "Session-based admission control: A mechanism for improving the performance of an overloaded web server," Tech. Rep. HPL-98-119, HP Labs, 1998.

[16] Intel, "Wired for management," http://www.intel.com/labs/manage/wfm/.

[17] VMware, "VirtualCenter white paper," http://www.vmware.com/pdf/vc_wp.pdf.

[18] Hewlett Packard Company, "OpenView web transaction observer," http://www.openview.hp.com/products/wto/.

[19] M. Aguilera et al, "Performance debugging for distributed systems of black boxes," in Symp. on Operating System Principles (SOSP), Oct 2003.

[20] M. Chen, E. Kiciman, E. Brewer, and A. Fox, "Pinpoint: Problem determination in large, dynamic internet services," in Dependable Systems and Networks (DSN), Jun 2002.