High Performance Resource Allocation and Request Redirection Algorithms for Web Clusters

Supranamaya Ranjan, *Member, IEEE,* and Edward Knightly, *Senior Member, IEEE*

***Abstract*— With increasing richness in features such as personalization of content, web applications are becoming more complex and hence compute intensive. Traditional approaches towards the design of web cluster architectures have targeted static content such as images that are usually network intensive. However, these methods are not applicable to dynamic content applications which are more compute intensive than static content. This paper proposes a suite of algorithms which optimize the performance of dynamic content applications by considering both server CPU loads and network latencies. The goal of the algorithms is to reduce client access times while also minimizing the resource utilization. The algorithms are designed for a hosting architecture comprising a grid of clusters inter-connected via high bandwidth links and each cluster hosting a complete application replica. A server migration algorithm allocates servers on-demand within a cluster while a server selection mechanism enables statistical multiplexing of resources across clusters by redirecting requests away from overloaded clusters. The paper also proposes a cluster decision algorithm which decides between serving a request locally after migrating-in additional servers at the local cluster or remotely by redirecting the request to a remote server that can serve the request earliest. Through a combination of analytical modeling, simulation over traces from large e-commerce sites and testbed implementation, we explore the performance savings achieved by the proposed algorithms.**

***Index Terms*— Web Servers, Multitier systems, Wide-Area Networks, Request Redirection, Resource Allocation**

I. Introduction

THE increasing reliance on the WWW as a ubiquitous medium becomes ever more apparent whenever there is a disruption in the availability of a certain web service. Furthermore, due to the much higher access network bandwidths today than a decade ago, clients of web services have much higher expectations with the service quality and hence are less tolerant to degradation in throughput or access times. The disruptions and degradations in a web service can be accounted for by one of the following two overload conditions: (1) *Time-of-Day* effect which is the diurnal variation in traffic observed at most web sites and reported to vary between a factor of 2 and 20 throughout the day [10][11] and; (2) *Flash Crowd* arrivals which is a sudden surge in users at a web site at an unexpected time or of unexpectedly high magnitude.

Typically, web content is distributed across multiple clusters world-wide so that the content can be brought closer to the users accessing it thereby decreasing the user perceived latencies. Distributed cluster architectures enhance the user perceived Quality-of-Service (QoS) since for *static content* such as image files, network latency and hence transmission time is a critical component. However, with the latest trend towards personalizing users' browsing experience, content providers are using an increasing amount of *dynamic content* in their web sites. Such content is generated dynamically depending on the user's location, past history or the nature of the request itself e.g., latest stock quote or latest auction bid. In such cases, a typical web page would comprise both static and dynamic fragments. However, dynamic content places different demands on the resources involved as compared to static content. Since processing dynamic content involves either forking a new process (e.g., CGI scripts) or accessing database tables (PHP scripts), it is more compute intensive than static content. This implies that the server selection mechanisms as designed for static content may not be optimal for dynamic content. While for static content, forwarding a request to the closest server makes sense, for dynamic content, the optimal server selection mechanism must take both the network latencies and server loads in to account. While under certain conditions it may make sense to forward a dynamic content request to the closest server, under different conditions it may be better to forward it to the server furthest away, if it has the lowest sum total of network latency and expected server processing time.

Moreover, current web cluster architectures are manually configured and cannot automatically adapt to changing workloads such as those caused by time-of-day effects or flash crowds. When these architectures are provisioned to handle the average workload, they suffer significant performance degradation when loads exceed capacity. In contrast, if these architectures are provisioned to handle the peak workload, they result in poor resource utilization for most of the time. On the other hand, web sites continue to grow larger and popular sites such as Google are known to contain as many as 100,000 servers spread across 60 clusters (source: CBS's 60 minute interview on Google in January 2005). An increase in the size and scale of clusters imposes severe demands on the operational costs, especially the power usage. According to a recent study [33] based on information collected from data centers, the total power used on servers and their cooling infrastructure represented 1.2% of the total power usage within the United States. Data centers are increasingly being architected with solutions such as HP's *Dynamic Smart Cooling* [4] that involve monitoring and adjusting air flow within a data center to reduce power usage. Thus, dynamically allocating resources within a data center would have the added advantage that redundant servers can be powered off, leading to significant power cost savings. Alternatively, redundant

Subsets of this work appear in [38],[37] and [36]. The research of Ranjan and Knightly was supported by NSF ITR grant CNS-0331620 and a grant from HP Laboratories.

servers can be *migrated* in to the cluster of another web site with a greater need for the extra resources. However, the challenge in designing a dynamic resource allocation policy is to minimize server resources without affecting the user perceived QoS.

Building on the two observations above, this paper proposes a suite of algorithms for *dynamic content* web sites such that performance as characterized by *client access time* and *resource utilization* is optimized even during overload conditions. We design the algorithms for an architecture comprising of a global-scale grid of clusters with each cluster hosting the same content and inter-connected to other clusters through custom high-bandwidth links. This architecture is representative of the large-scale e-commerce companies such as Yahoo, Google and Amazon, which host a single application across multiple data centers for performance and fault-tolerance reasons[1]. Each cluster manages its resources through an infrastructure-on-demand architecture similar to the ones proposed in references [9], [39], [38], with server resources being allocated only when needed. The paper proposes the following two mechanisms that jointly optimize cluster performance: (1) *Qu*ality-of-Service for *I*nfrastructure-on-*D*emand (QuID), a mechanism to optimize performance within a cluster by adapting the number of servers according to client demands. Through QuID, a hosted application experiencing a demand surge can *migrate* servers from a shared pool or away from underloaded clusters co-located within the same data center; (2) *W*ide *A*rea *ReDirection* (WARD), a mechanism that views the cluster grid as a single resource pool and multiplexes resources across the grid. This is done via a server selection mechanism to redirect requests to the best server, which could be either in the local cluster or a cluster remote to the user.

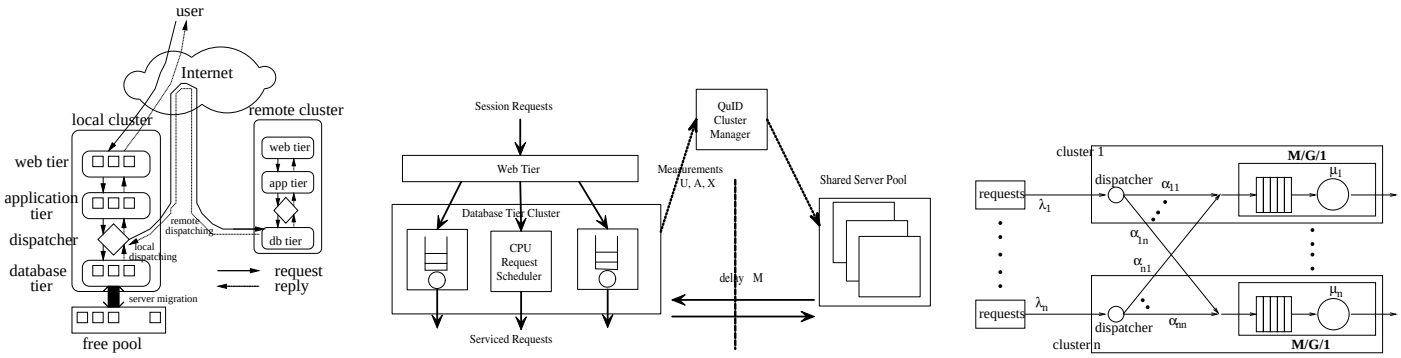
Specifically, we introduce the following algorithms: (1) *QuID-online*, a dynamic resource allocation algorithm that exploits long term variations in traffic to allocate resources within a cluster such that the hosted application's QoS is still met while using a lower number of servers compared to static allocation; (2) *WARD per-request (or per-query) redirection* algorithm which redirects requests (or queries) away from an overloaded cluster to a server that minimizes the total networking plus server processing delay and; (3) *cluster decision* algorithm that combines the two algorithms above to decide which one should be used under which workload conditions. We motivate the cluster decision algorithm via an example. Suppose a cluster is overloaded due to sudden popularity of the application in a geographic region and additional servers are available locally, either from a pool of unused servers or from an underloaded application's cluster. If the workload burst is expected to be sustained for a duration longer than migration time, defined as time needed to allocate a new server, then allocating new server(s) via QuID would achieve a better performance. However, if the workload burst is too small, then new servers if allocated, may have to be returned back to the pool immediately and hence server allocation via QuID may be prohibitive. In such a scenario, it would be of greater benefit to redirect the requests away to a remote cluster via WARD.

In order to quantify the performance gains achievable by the QuID and WARD mechanisms, we also introduce analytical models for each of them. Specifically, we formulate the dynamic server allocation problem as a constrained optimization problem, namely *QuID-optimal*, solution to which yields the optimal number of resources needed for an application's arrival and demand sequence. While QuID-optimal is not realizable in practice, it serves an important purpose by providing a bound on the resource savings that can be achieved by any dynamic allocation policy subject to satisfying an application's QoS requirements. Next, we formulate the server selection problem by assuming a single bottleneck tier in each multi-tiered cluster and modeling it as a M/G/1 queue and by considering inter-cluster latencies as the cost of redirection. Solution to the analytical framework for WARD yields the optimal percentage of requests that should be redirected to a remote cluster replica under given server and network characteristics. The WARD analytical model allows us to perform a systematic performance evaluation of the benefits afforded by WARD. An example finding by this model is that for dynamic content applications, a server selection mechanism *must* obtain fine-grained server load information owing to a much lower tolerance to errors in server loads compared to network latencies. This vindicates our hypothesis that for dynamic content applications, a server-side redirection policy can achieve a better performance than client-side redirection policies which can not obtain server load information at the same granularity and similar overheads as the server-side policies.

Finally, via trace-driven simulations as well as implementation of a dynamic content application (online bookstore), we provide a proof-of-concept demonstration of the performance benefits due to QuID and WARD. We implement a testbed that realizes the cluster grid architecture over which we replicate the online bookstore application based on the TPC-W benchmark [7]. Each cluster is implemented with Apache as web tier, PHP as application tier and MySQL as database tier while the inter-cluster link delays are emulated via Nistnet [6]. Since the database tier bottlenecks first, we implement QuID and WARD algorithms at this tier. We use this experimental testbed for three purposes. First, we compare WARD's redirection algorithm against strawman algorithms to prove that combining both server loads and latencies in the redirection decision yields significant benefits. Second, we validate that the WARD analytical model based on M/G/1 queues does model the implementation fairly accurately despite its simplifying assumption of Poisson arrivals and hence the performance implications derived from the model can be expected to hold in practice. Third, we validate our cluster decision algorithm by examining the workload conditions under which we can achieve better performance by redirecting via WARD versus migrating new servers via QuID.

The rest of this paper is organized as follows. Section II introduces the cluster grid architecture and Section III introduces the QuID-online, WARD per-request and probabilistic, and the cluster decision algorithms. Section IV formulates the analytical models for QuID and WARD. Next, Section V quantifies the performance benefits afforded by QuID and

[1]Applications are hosted on such a cluster grid architecture with either full- or partial-replication i.e., the content is replicated either completely or partially on another cluster instance. In this paper, we assume full replication.



(a) Cluster grid (b) QuID model (c) WARD model

Fig. 1. Cluster grid architecture with QuID for improving intra-cluster performance and WARD for inter-cluster resource sharing.

WARD and also studies the tolerance of WARD to errors. Section VI describes our testbed implementation and also provides a systematic performance evaluation of QuID and WARD against strawman algorithms as well as a validation of the cluster decision algorithm. Section VII provides an overview of related work and we summarize our conclusions in Section VIII. Finally, the Appendix provides details on the optimization problem, QuID-optimal.

II. Cluster Grid Architecture

We present our web hosting architecture in which an application is replicated across a grid of clusters that are inter-connected by custom high-bandwidth links as shown in Fig. 1(a).

To illustrate this architecture, let's consider the requests of an e-commerce session. First, a client request arrives at the initial cluster, where the selection of the initial cluster is via client-side redirection policies such as DNS round robin or more sophisticated policies such as [25], [21], [3]. On arriving at the initial cluster, a dispatcher dispatches the request to a server on the web-tier using either round robin or sophisticated policies as in reference [13]. If the request is for a static page, it is served by the web server which also forms the response and returns it to the user. If the request is for dynamically generated content such as those involving purchase processing or shopping cart, then it is forwarded to a server in the application-tier. The application server then resolves all the dynamic fragments embedded within the client request by generating relevant database queries. The decision of which database server must handle a database query is made by another dispatcher. Finally, the application server collects all the responses to the database queries and returns the page to the web server, which then forwards it back to the client.

The proposed architecture provides a foundation for efficient utilization of server resources: (1) within a cluster via QuID-driven dynamic allocation of server resources and; (2) across clusters via WARD-driven statistical multiplexing of requests. Thus, when a particular cluster becomes a hot-spot due to flash crowds or time-of-day effects, additional servers may be added to the local cluster using QuID. However, if servers are not available locally, requests can be transparently redirected using WARD to other clusters while still ensuring a latency benefit to clients.

The decision of when to migrate additional servers versus redirect them to remote clusters is a policy decision that depends on both quantitative factors such as migration time, inter-cluster latency and predicted duration of the traffic burst as well as qualitative factors such as agreement between cluster operators on leasing servers. For instance, if the burst duration is estimated to be less than migration time, then it may be advantageous to redirect requests to a remote cluster. However, if it's observed that a significant fraction of requests are being redirected, then conditional on server availability locally, additional servers may be added to the local cluster. Since time-scales are an important aspect of the decision process between redirecting a request versus allocating a new server locally, in Section III-C, we propose a *cluster decision algorithm* that takes traffic burstiness in to account.

A. QuID System Model

Fig. 1(b) shows a simplified abstraction of the cluster grid architecture to explore the issue of QoS-driven dynamic server migration. The system model depicts a single tier of servers (database tier) as well as the dispatcher that allocates sessions and requests to servers. For a cluster hosting a single application, the system can be viewed as dynamically migrating servers to and from a shared pool according to the workload and QoS requirements of the hosted application.

Importantly, there is a migration time M associated with migrating a server from the shared pool and into a particular cluster tier. Thus migrating a server in to the database tier would consist of the following components: (1) a constant time, M_{boot} for booting up the operating system and starting the server daemon and; (2) a variable time, M_{query} to restore the new server's state such that it is consistent with the existing servers in the tier. Specifically, with respect to a database server, this can be achieved by re-applying the queries that were made since the last time that this server was part of the database tier. Based on experiments on a real testbed as described later in Section VI, migration time can be expected to be on the order of minutes, depending on the operating system used and the amount of work involved in bringing the server state up-to-date.

Similarly, releasing a server from a cluster also takes time, as even after application servers stop sending new queries to a database-tier server, all queries currently in progress at the server must be completed or timed out. Based on server logs investigated for an E-commerce web site (see Fig. 2), we expect typical "warm-down times" to migrate out servers to be in the range of minutes as well.

B. WARD System Model

Once, a request arrives at a cluster tier, a WARD dispatcher can potentially redirect the request to a remote cluster via a redirection algorithm or decide to service it at the local cluster tier. In multi-tiered clusters, WARD can be implemented on the dispatchers in front of any of the tiers: web, application or database. If implemented before the web-tier, then WARD can be used to redirect complete requests to new clusters. Alternatively, if implemented before the database-tier, WARD can be used for finer-grained dispatching of the database queries to new clusters.

Thus, the objective of the redirection algorithm is to redirect requests or queries only if the savings in the processing time at the remote cluster overwhelm the network latency incurred to traverse the backbone in both the forward and reverse path. In this way, end-to-end client delays can be reduced while requiring changes only to the dispatcher, and leaving other elements unchanged.

III. Algorithms

In this section, we present our algorithms for achieving intra-cluster as well as inter-cluster performance benefits.

A. QuID Algorithm

First, we present *QuID-online*, an algorithm that attains a predictable cluster-wide QoS by maintaining a targeted average CPU utilization by acquiring and releasing servers in response to changes in load. With simple queuing theoretic arguments and simulation experiments, we show that QuID-online's utilization-target provides an effective mechanism for maintaining cluster-wide QoS. Moreover, by time-averaging workload variations and (necessarily) gradually migrating new servers into and out of the cluster, the algorithm minimizes the impact of rapid fluctuations in load and instead responds to long-time-scale trends.

To achieve these goals, the online algorithm requires measurements characterizing the state of the system. Thus, the QuID-online cluster manager depicted in Fig. 1(b) periodically queries the database servers for summary workload information. The measurement interval, which we denote by τ, is typically on the order of minutes in duration. Thus, every τ the cluster manager queries servers for the following measurements: X, the number of request completions in the previous measurement interval; A, the number of request arrivals in the previous measurement interval; and U, the average CPU utilization (over all servers) in the previous measurement interval. Thus, $U = (U_1 + U_2 + \cdots + U_N)/N$ from the reported values of individual CPU utilizations. The average CPU utilizations are generally available from operating system monitors and application level measures such as request arrivals and completions per interval are typically available from server logs. In both cases, this information is summarized at the servers and communicated to the cluster manager.

Given N, the current number of servers, as well as μ, the target utilization, QuID-online computes N', the required number of servers for the next interval as follows:

1) $D = U/X$, compute average demand per completion;
2) $U' = \max(A, X)D$, compute the normalized utilization based on the current number of servers and all arrivals;
3) $N' = \lceil NU'/\mu \rceil$, compute the upper bound on number of servers needed to achieve the target utilization μ.

Note that when the system is under increasingly heavy load, arrivals may exceed completions and the computation of U' takes this into account. This allows QuID-online to react more quickly to increases in load than via the use of X or U alone. Moreover, we use the maximum of A and X to avoid releasing servers too quickly, thereby making the algorithm less sensitive to short-term decreases in load.

QuID-online initiates requests to acquire or release servers whenever $N' \neq N$. However, note that the overhead due to server migration time prohibits rapidly changing the number of servers. There are several aspects of the algorithm which mitigate the effects of this overhead. First, the measurement interval τ can be increased to average out short-term load fluctuations. Second, we note that the migration time itself provides a damping mechanism and we employ an additional policy as follows. If more servers are needed ($N' > N$), servers previously warming down in preparation for migration out of the cluster are reinstated into the cluster in a last-in-first-out order before any requests are made to acquire additional servers. Similarly, if fewer servers are needed ($N' < N$), servers previously warming up in preparation to join the cluster are released in a last-in-first-out order before any requests are made to release servers doing useful work. Finally, we incorporate the overhead of migration time by accounting for servers migrating to and from the cluster in the computations of the average numbers of servers.

QuID online's use of a target CPU utilization μ as a control parameter for QoS is justified as follows. First, assume that servers have a large number of threads so that requests rarely incur software blocking. Moreover, consider that threads are scheduled in a first-in-first-out manner without preemption by their server's CPU. Furthermore, the individual sessions constituting the total workload can be considered to be independent. In this scenario, a server cluster as in Fig. 1(b) can be modeled as a $G/G/N$ system which has a mean response time R under heavy traffic given by [32]: $R = \frac{U}{X} + \frac{\sigma_a^2 + \frac{\sigma_b^2}{N^2}}{2\bar{t}(1-U)}$, where σ_a^2 and σ_b^2 are the variance of inter-arrival and service times respectively, and $\bar{t}$ is the mean inter-arrival time. Since the response time R is determined by the utilization U, we maintain a target utilization μ by controlling the number of servers N.

Regardless, QuID-online does not attempt to relate the mean response time R and average utilization U directly and makes no assumptions about inter-arrival or service time distributions. However, the relationship between response time and target utilization can be determined empirically for a particular workload. We utilize this methodology in Section V and show how a proper setting of μ can control cluster-wide QoS.

B. WARD Algorithms

Within the framework of wide-area redirection of requests or queries to improve cluster performance, we propose two different algorithms.

The first algorithm, namely *per-request* (or, equivalently *per-query*) redirection computes the best server per request (or, per-query) as its name implies. The objective of the redirection algorithm is to minimize the total time to service a request. Namely, if a request arrives at cluster k, then the objective is to dispatch the request to cluster j satisfying

$$\text{argmin}_j \left(2\,\Delta_{kj} + T_j\right) \tag{1}$$

where Δ_{kj} denotes the network delay between cluster k and j and T_j is the request's service time at cluster j.

In practice, the actual service time at each remote cluster T_j cannot be known in advance. Yet, it can be estimated from the average load at cluster j as well as the request type. Thus, we employ a measurement-based algorithm in which the average T_j is estimated from ρ_j, the mean load at cluster j, as well as the request type. In WARD, this is achieved by measuring a mean delay vs. load profile for each request type in an offline workload characterization phase as described later in Section VI-B. Clusters then periodically exchange load information to refine their estimates of each others' processing delays. In contrast, Δ_{kj} remains relatively static among clusters due to their high-speed interconnection links. Thus, on request arrival, the dispatcher uses the measured load at cluster j on the delay vs. load profile corresponding to this request's type to estimate the total service time on cluster j.

We consider a second policy, namely *probabilistic redirection* which does not make a decision on a per request basis but rather computes a fraction of requests to be remotely dispatched given the cluster workloads and inter-cluster latencies. In particular, we show in Section IV that under certain simplifications there is an optimal ratio of requests that should be remotely dispatched in order to minimize the delay of all requests. Once this ratio is known, the dispatcher can simply remotely redirect a request with the computed probability.

C. Cluster Decision Algorithm

Finally, we present the cluster decision algorithm that decides between QuID and WARD under different workload conditions. The intuition behind the algorithm is that if the traffic burst is expected to be sustained for a period larger than the migration time, then new servers are migrated locally via QuID. In contrast, if the traffic burst is expected to last for a time-period smaller than the migration time, then migrating new servers is prohibitive and hence requests are redirected to other clusters via WARD.

The algorithm quantifies traffic burstiness by estimating the workload's autocorrelation for lag equal at the key time-scale of migration time. While calculating the autocorrelation, we consider the number of requests seen at the dispatcher in the last n time intervals. Let the sequence of requests seen at the dispatcher be represented by $Y_1, Y_2, ...Y_n$. For ease of notation, let each time interval be a second and let migration time be m seconds. Let $\overline{Y}$ represent the sample mean for the time series of requests. Autocorrelation coefficient at time n for a lag equal to migration time of m seconds is given by:

$$\hat{A}(n,m) = \frac{\sum_{t=1}^{n-m}[Y_t - \overline{Y}][Y_{t+m} - \overline{Y}]}{\sum_{t=1}^{n}(Y_t - \overline{Y})^2} \tag{2}$$

Note that Equation 2 calculates the normalized autocorrelation coefficient such that the value at lag of zero is 1.0 and the value varies between 0 and 1 always. Intuitively, a high value for $\hat{A}(n,m)$ signifies that the workload is correlated within the last migration time interval and can be expected to remain predictable for the next migration time interval making it easier for QuID to track. In contrast, a low value signifies that the workload is highly bursty and unpredictable and hence difficult for QuID to track.

Thus, the cluster decision uses the autocorrelation coefficient as measured in the last n intervals as an estimate for the future traffic trend. The algorithm dictates the usage of QuID to migrate additional servers to the local cluster when the workload is predictable ($\hat{A}(n,m) \geq \theta$) while requests are redirected to remote clusters via WARD when the workload is unpredictable ($\hat{A}(n,m) < \theta$). In Section VI-E, we estimate the value of the autocorrelation threshold θ experimentally for a testbed implementation of an online bookstore and given values of migration time and mean workload arrivals.

IV. Analytical Models

In this section, we first develop an optimal solution for achieving intra-cluster performance benefits via server migration. Next, we develop an analytical model for achieving inter-cluster benefits via redirection.

A. QuID

In this part of the section, we develop QuID-optimal, an optimal off-line algorithm for dynamic resource allocation via server migration. QuID-optimal provides a benchmark for evaluation of practical algorithms by computing the *minimum* number of servers required by a server migration algorithm such that the application's response time requirement (QoS) is bounded cluster-wide. Consequently, it characterizes the maximal available gain of any dynamic policy as compared to a static policy.

QuID-optimal uses as its inputs a workload trace, i.e., a sequence of session and request arrival times as well as the corresponding server CPU processing time required to service each request. For a particular migration time and maximum response time, the algorithm jointly computes the best sequence of dispatching decisions (allocation of requests to servers), scheduling decisions (when to serve each request), and dynamic server allocation decisions (how many servers are required to ensure all request response times are within the required QoS bound). The solution, while not realizable in practice, provides the "best" decision sequence among all three dimensions in that it provides the minimum server solution subject to the maximum response time QoS requirement.

The problem formulation for QuID-optimal is presented in the Appendix. To solve this problem, we first formulate the dynamic server allocation problem as a constrained optimization problem. Next we transform the non-linear problem into a linear programming problem and show that the solutions of the two problems are equivalent. Finally, we discuss a simplified and more computationally feasible (but still not on-line realizable) bound that relaxes some of the most computationally intense constraints.

B. WARD

Next, we develop an analytical model for wide area redirection. For a given workload, mean and variance of service time, and network latency, we derive an expression for the delay-minimizing fraction of requests that a dispatcher should redirect to remote clusters. Moreover, we compute the average total response time including service- and waiting-times and end-to-end network latency. In Section V, we then perform a systematic performance analysis to estimate the optimal dispatching ratios α^* and to predict the expected average request response time under varying parameters, such as the server load, the end-to-end network latency and the average request service time.

Fig. 1(c) illustrates the system model for WARD. We model request arrivals at cluster i as a Poisson process with rate λ_i and consider a single bottleneck tier modeled by a general service time distribution having mean $\overline{x}_i$ and variance σ_i^2.

We consider a redirection algorithm in which a request is redirected from cluster j to cluster i with probability α_{ji}, i.e., we consider probabilistic redirection. Denote $E[T_i]$ as the expected total delay for servicing a request at cluster i, and denote Δ_{ji} as the one-way end-to-end network latency for a request sent from cluster j to cluster i.

For the general case of a system of n cluster replicas, denote $\mathbf{A} = \{\alpha_{11}, \ldots, \alpha_{ji}, \ldots, \alpha_{nn}\}$ as a matrix of request dispatching fractions, $\mathbf{E}[\mathbf{T}] = \{T_1, \ldots, T_n\}$ as the vector of all total delays at the bottleneck tier at a cluster and $\mathbf{D} = \{2\Delta_{11}, \ldots, \Delta_{ji} + \Delta_{ij}, \ldots, 2\Delta_{nn}\}$ as a matrix of round-trip times from cluster i to cluster j and back. Furthermore, denote $\mathbf{L} = \{\lambda_1, \ldots, \lambda_n\}$ as a vector of request arrival rates at the cluster dispatchers, $\overline{\mathbf{X}} = \{\overline{x}_1, \ldots, \overline{x}_n\}$ as the average service time, $\mathbf{C} = \{c_1, \ldots, c_n\}$ as the vector of coefficient of variation for the service times, with $c_i = \sigma_i / \overline{x}_i$.

Lemma 1: The mean service time for the redirection policy using a dispatching fraction $\mathbf{A}$ is given by:

$$\mathbf{E}[\mathbf{T}] = \mathbf{A} \cdot \overline{\mathbf{X}} + \frac{(\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}}^2(1 + \mathbf{C}^2)}{2(1 - (\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}})} + \mathbf{A} \cdot \mathbf{D} \qquad (3)$$

Proof: The total service time is composed of 3 durations: (i) the network latency of transferring the request to and from the remote cluster (ii) the queuing time at the cluster and (iii) the service time at the cluster.

For symmetry reasons, in the following equations, we attribute the "costs" to the receiving cluster i. First, we assume that the network latency between the dispatcher and a local cluster $\Delta_{ii} = 0$ and hence, network latency is incurred only by requests dispatched to a remote cluster: $\alpha_{ji}(\Delta_{ji} + \Delta_{ij})$.

Second, consider the mean waiting time for a request in an cluster queue before being serviced. In general, the waiting time for for an M/G/1 queue, where $\rho = \lambda\overline{x}$ is: $\frac{\rho\overline{x}(1+c^2)}{2(1-\rho)}$. For any cluster i, the arrival rate λ is the sum of the requests that are dispatched from all cluster j to cluster i, i.e., $\lambda_i = \sum_j \alpha_{ji}\lambda_j$. With this λ, the waiting time for a single cluster i can be rewritten as:

$$\frac{(\sum_j \alpha_{ji}\lambda_j)\overline{x}_i^2(1 + c_i^2)}{2(1 - (\sum_j \alpha_{ji}\lambda_j)\overline{x}_i)}.$$

Finally, service time for a request at cluster i is given by $\alpha_{ji}\overline{x}_i$. The addition of these 3 terms for a set of clusters yields Equation (3). ■

From Equation (3), we can compute the optimal dispatching ratios that minimize the service times over all requests. In particular, let $\mathbf{A} = \{\alpha_{11}^*, \ldots, \alpha_{nn}^*\}$ denote the matrix of optimal request dispatching ratios.

Proposition 1: Using $\mathbf{E}[\mathbf{T}]$ defined in Equation (3), the optimal dispatching ratios $\mathbf{A}^*$ are given by:

$$\frac{\partial}{\partial\alpha}(\mathbf{A} \cdot \overline{\mathbf{X}} + \frac{(\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}}^2(1 + \mathbf{C}^2)}{2(1 - (\mathbf{A} \cdot \mathbf{L})\overline{\mathbf{X}})} + \mathbf{A} \cdot \mathbf{D}) = 0 \qquad (4)$$

To solve Equation (4) for all α_{ji}, we use the following constraints to reduce the number of unknowns. First, we clearly have that $\sum_j \alpha_{ji}^* = 1$. Second, $\lambda_i \geq \lambda_j \implies \alpha_{ji}^* = 0$ i.e., when considering 2 clusters with different λ, under steady-state conditions, no requests will be dispatched from the cluster with a smaller arrival rate to the cluster with a higher arrival rate. Thus, as we show later in Section V, for a system of two clusters, we can reduce the problem to one unknown, α which is the fraction of requests served locally, which can be solved by using Proposition 1.

The optimal dispatching ratios $\mathbf{A}^*$ can be used to predict the average request service time for a system of cluster replicas.

Proposition 2: The expected request service time under optimal dispatching ratios is given by:

$$\mathbf{E}[\mathbf{T}^*] = \mathbf{A}^* \cdot \overline{\mathbf{X}} + \frac{(\mathbf{A}^* \cdot \mathbf{L})\overline{\mathbf{X}}^2(1 + \mathbf{C}^2)}{2(1 - (\mathbf{A}^* \cdot \mathbf{L})\overline{\mathbf{X}})} + \mathbf{A}^* \cdot \mathbf{D} \qquad (5)$$

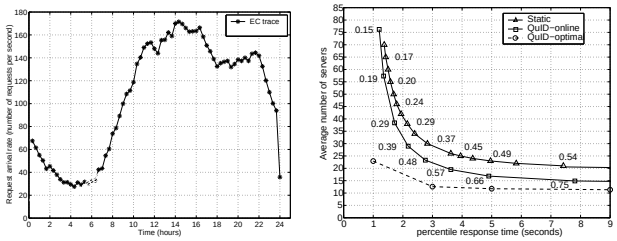
Proof: Equation (5) follows from Lemma 1 and by using the optimal dispatching ratios from Equation (4). ■

V. Numerical Results

This section quantifies the performance benefits achieved by QuID and WARD via an extensive numerical analysis.

A. QuID: Trace Driven Simulation

First, we explore the resource savings and QoS improvements available due to QuID as compared to a static approach. We also compare the performance of the QuID-online algorithm against the optimal offline algorithm, namely QuID-optimal to benchmark the amount of performance benefits exploited by QuID-online.



(a) Trace Request Arrivals (b) Cluster Performance

Fig. 2. Subfigure (a) shows the trace arrival pattern while (b) shows the performance as characterized by number of servers used by a resource allocation algorithm vs. 95%-ile response time.

We use a commercial trace as depicted in Fig. 2(a). The trace, referred to as "EC trace" is a 24-hour trace from a

large-scale E-Commerce site with a multi-tier architecture as illustrated in Fig. 1(b)[2]. In the EC trace [10], each request is classified as either static, dynamic, search or other. Requests for dynamic pages are further classified as those that are uncacheable versus those that were resolved by the cache (cache hit) and those that were a cache miss. The workload consists of 18% static requests, 40% that result in a cache hit, 17% that result in a cache miss, 8% that are uncacheable and 17% requests of other types. Each request type incurs a varying amount of CPU demand on each cluster tier, the details of which can be found in reference [38]. More importantly, the cache miss and uncacheable requests incur 500 and 100 times more CPU demand than a cache hit request. Moreover, the ratio of peak-to-mean requests over the entire trace duration of 24 hours is 1.5 while the trace exhibits a strong autocorrelation value of 0.99 at lags of 5 minutes. The trace is regenerated through our simulator and each request's workload is generated by an exponentially distributed random variable with mean demands per tier as calculated from the original trace.

Through a trace-driven simulator based on YacSim [28], we study the performance of QuID-online and compare it against a static allocation policy. We consider a single bottleneck tier, in particular the application server tier. The simulator models servers according to a FSFS non-preemptive CPU schedule and implements the QuID-online algorithm to migrate servers in and out. In each experiment, the measurement interval, τ = 5 minutes and migration time, M = 1 minute.

Fig. 2(b) depicts the average number of servers against the response times for the QuID-online algorithm along with a upper baseline provided by a static allocation policy and a lower baseline as provided by the optimal policy, QuID-optimal. Each point on the curve represents the result of a simulation for a different target utilization μ, with measured utilization values depicted next to each point. For example, the figure shows that for a utilization of 0.66, QuID-online achieves a 95%-ile response time of less than 5 seconds and requires 17 servers on average. In contrast, static allocation requires 24 servers to achieve the same response time. Hence, for this response-time QoS target, QuID-online has reduced the required number of servers by 29%. An alternative interpretation is that for a fixed number of servers, QuID-online improves QoS as compared to a static approach. For example, on using 20 servers, QuID-online achieves a 95%-ile response time of 3.7 sec vs. 7.4 sec for static allocation, a 50% reduction. Moreover, as data center operators would likely be required to over-provision servers to address unknown and highly variable workloads, the gains of QuID-online as compared to a static approach may be even more pronounced in practice.

Further, note that the performance of QuID-online as compared to the optimal policy reveals that the same response time of 5 seconds can be achieved by the optimal policy by using only 12 servers. Hence, a further resource savings of 21% is available in theory. However, we observe that due to the assumptions made in solving the optimization problem associated with QuID-optimal, it would be impossible to realize the complete gains of QuID-optimal in practice. In particular, QuID-optimal differs from a realistic setup such as our simulation model in the following: (a) it doesn't model session affinity; (b) it optimally schedules CPU time and; (c) it divides an individual request across multiple servers.

[2]This trace is not available in the public domain and for privacy concerns, it has been scaled so that the depicted rates of Fig. 2(a) differ by a constant factor from the actual trace.

B. WARD: Numerical Results

Next, we show that wide area redirection is able to optimize inter-cluster performance characterized by total access delays perceived by clients. Then, we experimentally establish the higher tolerance of WARD to measurement errors in network latencies than to errors in server load measurements. This further verifies our hypothesis that redirection mechanisms must obtain finer-granularity server load measurements, which WARD is better-suited at given that it is implemented on dispatchers that are co-located with local servers and are connected via high-bandwidth links to the remote servers.

Using the system model developed in Section III, we consider a system of 2 clusters with replicas having the same average request service time $\overline{x}$. Furthermore, we assume a symmetric network with wide area latency between the two clusters: $\Delta = \Delta_{12} = \Delta_{21}$. Finally, we set $\lambda_2 = 0$, which satisfies $\lambda_1 > \lambda_2 \implies \alpha_{21} = 0$, and denote $\lambda := \lambda_1$ and $\alpha^* := \alpha_{11}^*$ for simplicity.

The dispatching ratio is computed based on Equation (3):

$$E[T] = \alpha\overline{x} + \frac{\alpha\lambda\overline{x}^2(1+c^2)}{2(1-\alpha\lambda\overline{x})} + (1-\alpha)\overline{x} + \frac{(1-\alpha)\lambda\overline{x}^2(1+c^2)}{2(1-(1-\alpha)\lambda\overline{x})} + 2(1-\alpha)\Delta \quad (6)$$

Equation (6) is solved according to Proposition 1 to obtain the optimal dispatching ratio α^*. Henceforth, we refer to the term $1-\alpha^*$ as the *remote redirection* ratio, i.e. the fraction of requests dispatched remotely. Then, according to Proposition 2, the expected total delay of the cluster system is given by substituting α by the optimal ratio α^* in Equation (6).

If not otherwise stated, we use the following default values: $\overline{x} = 42.9$ msec, $\sigma = 40.1$ msec, where these values were obtained from our testbed and $\Delta = 36$ msec, which corresponds to a speed-of-light latency for two clusters separated by 6 time-zones at 45^o latitude[3]. We will use $\rho = \lambda\overline{x}$ to denote the total load on *all* clusters. For clusters without redirection, ρ corresponds to the server load on the bottleneck tier, whereas WARD can split this load among the local and remote clusters. To obtain a given value of ρ, the arrival rate λ will be scaled, with $\overline{x}$ remaining fixed.

1) The case for wide-area redirection: First, we provide evidence that wide area redirection is able to decrease the user-perceived total delay. We calculate the total delay of WARD using Equations (4) and (5) and compare it to the total delay of a cluster that does not implement redirection. Fig. 3 shows the total delay as a function of the end-to-end latency and different system loads ρ.

For low loads ($\rho = 0.5$), improvements are achieved only when the end-to-end latency $\Delta \leq 25$ msec. For higher

[3]Given the circumference of earth at 45^o latitude as 28335 km and the speed-of-light through optical-fiber as 205 km/sec, the one-way latency across 1 time-zone can be calculated as: $28335/(205*24) \approx 6$ msec.

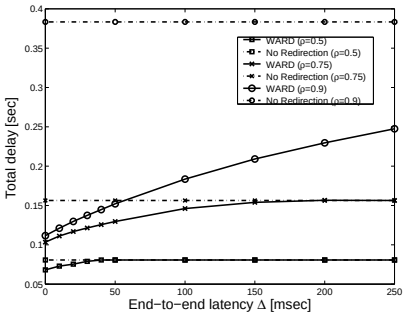
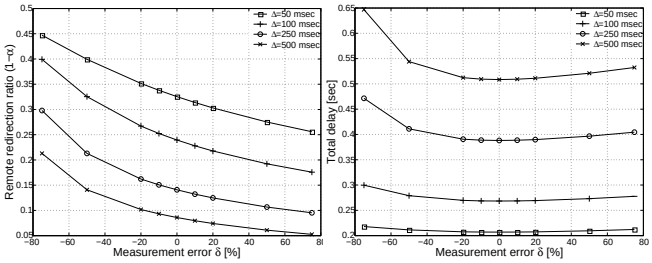


Fig. 3. Comparison of total delay with and without wide area redirection



(a) Remote redirection ratio (b) Total delay

Fig. 4. WARD performance under network measurement errors. A value of 0 on the x-axis corresponds to perfect end-to-end latency information. The server load is set to $\rho = 0.95$.

latencies, the redirection cost exceeds the processing time so that all requests are serviced locally. However, a significant improvement is achievable for higher loads. For a moderate load of $\rho = 0.75$ and $\Delta < 50$ msec, the total delay is reduced from 0.16 sec to 0.13 sec using WARD, an improvement of 18%. For a heavily loaded system with $\rho = 0.9$ and when $\Delta < 50$ msec, the total delay is reduced from 0.38 sec without redirection to 0.15 sec using WARD, an improvement of 60%. Moreover, for loads $\rho > 0.9$, still higher improvements are predicted by the model.

2) Susceptibility to Measurement Errors: Next, we establish the fact that the performance benefits out of a wide-area redirection policy can be exploited only when the server utilization values are available at a very fine granularity. In contrast, network latencies can be quite coarse grained without any performance penalties out of making the wrong decision. For establishing this claim, we study the performance impact due to measurement errors in network latency Δ and server load ρ. We quantify the performance impact in terms of *error tolerance* defined as the percentage error $\pm\epsilon$ that increases the total delay by at most 2%.

First, we study the impact of network latency errors as follows. Let Δ denote the true inter-cluster network latency and $\widehat{\Delta} = \Delta + \delta$ the measured value, and $\widehat{\mathbf{D}}$ the corresponding round-trip time matrix. The dispatcher calculates the dispatching ratios replacing $\mathbf{D}$ by $\widehat{D}$ in Equation (3).

For the calculation of average total delay, Equation (5) is used with the true latency values $\mathbf{D}$. The effects of measurement error in network latency on the remote redirection ratio $(1 - \alpha)$ and the resulting average request response time are shown in Fig. 4. Each curve denotes a different (true) latency Δ, and the x-axis denotes the error δ, in percent of Δ.

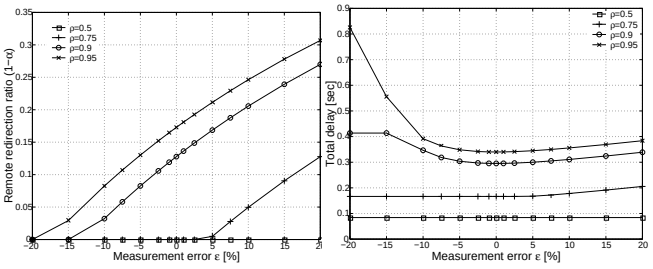
Fig. 4(a) shows that the redirection ratio changes more for negative δ than for the corresponding positive δ. The reason is that the redirection ratio does not grow linearly with the end-to-end latency. As a consequence of the asymmetry, the total delay increases more for negative δ, as shown in Fig. 4(b). Note, however, that the response times are not highly sensitive to latency measurement errors and the the error tolerance is quite high at $\pm 20\%$.

Likewise, we consider a scenario when the dispatcher has inaccurate server load measurements, e.g., due to delays in receiving the measurements. In this scenario, the measured load at the dispatcher is given by $\widehat{\rho} = \widehat{\lambda}\overline{x}$, with $\widehat{\lambda} = \lambda + \epsilon$ (where ϵ is in percent of the correct load ρ) and the corresponding measured arrival rate by $\widehat{\mathbf{L}}$. The network latency is set to $\Delta = 500$ msec.

First, consider the case of measurement error $\epsilon > 0$, when the dispatcher assumes the server load to be higher than what it is and hence it redirects more requests than the optimal. Fig. 5(a) shows that the remote redirection ratio increases with increasing measurement errors. These extra redirections incur additional network latencies and hence the total delay also increases linearly in Fig. 5(b). In particular, for $\rho \geq 0.9$, the error tolerance is $+1.5\%$. Next, consider negative ϵ, when the dispatcher assumes the local server load to be less than the actual value and hence redirects pessimistically. As a result, the load on the local server incurs greater processing times at the local cluster. As expected, Fig. 5(b) shows that at high server loads $\rho \geq 0.9$, the total delay is much more sensitive for negative ϵ with an error tolerance of only -0.5%.

Thus, comparing the impact of latency and server measurement errors, the error tolerance for latency is high at $\pm 20\%$ while that for server load is an order of magnitude lower at $+1.5, -0.5\%$. We thus conclude that greater accuracy is needed in server load measurements than network latency.

Since server-side redirection mechanisms can obtain more fine-grained server load information at lower overheads, this verifies their superiority in efficiently load-balancing requests for dynamic content applications than client-side mechanisms: First, client-side redirection policies when implemented at clients or DNS servers may not have access to high-bandwidth links to the servers and; Second, client-side redirection as implemented at proxies near clients (e.g., Akamai) may have high-bandwidth access links, however, their overhead for obtaining the server load information is much higher, given the much larger number of client-side dispatchers than server-side ones. Consider, the following: Say, the number of clusters is n and thus in WARD, there are n server-side dispatchers. If the total number of servers across all clusters in the tier implementing WARD is M, where $M > n$, then the complexity of information exchanged in WARD is $O(nM)$. In contrast, consider a client-side redirection mechanism with



(a) Remote redirection ratio (b) Total delay

Fig. 5. WARD performance under server load measurement errors.

one dispatcher per client-side proxy for a total of N client-side proxies or dispatchers which yields a complexity of $O(NM)$. Given that the number of client-side proxies is typically much larger than the number of cluster replicas, the complexity is much less for server-side redirection mechanisms[4].

VI. Experimental Evaluation

In this section, we first describe our testbed implementation of a multi-tiered e-commerce site. Using synthetic traces generated to follow the TPC-W benchmark for web workload, we validate the following interesting issues. First, we validate that WARD's per-query redirection algorithm performs better than a set of strawman algorithms. Second, we validate that WARD's analytical model using M/G/1 queues as developed in Section IV does match the implementation closely. Third, we provide a comparative performance study of WARD's per-query redirection algorithm against the probabilistic algorithm. Finally, we validate the cluster decision algorithm by comparing the performance of QuID and WARD under varying workload scenarios.

A. Implementation

We use the experiment set up shown in Fig. 1(b) consisting of two clusters. Traffic arrives at only the local cluster whereas the workload of the remote cluster is solely created by dispatched requests. We expect such zero-load conditions on remote data centers several time-zones away due to the *time-of-day* effects. The number of servers in the web-tier in both the clusters is statically allocated such that this tier doesn't ever become the bottleneck. We implement our wide-area redirection as well as server migration algorithms on the database tier.

The dispatcher maintains consistency across the currently active servers in the database tier by two means: maintaining an identical *total ordering* of writes at all database servers, and a *read-one write-all* dispatching strategy, where the queries that involve updates to the database tables are sent to all the database servers. More details on the consistency model and implementation can be found in references [36] and [8]. Moreover, to bring the database tables of newly migrated servers to the same consistency level, we use the following algorithm: (a) when a server is migrated out of the tier, the server first waits in a "warm down" phase, where all the currently running queries are completed, after which its daemon processes are stopped and the machine halted. The dispatcher then maintains a log of the database writes that happen during this server's absentia; (b) when a server is migrated in to the tier, the dispatcher sends the missing write queries to the server in a "warm up" phase and new queries are sent once the server is consistent with the entire tier.

Each cluster is implemented with Apache as the web tier, PHP as the application tier and MySQL as the database tier. All the servers used in our experiments are Intel Pentium IV 2.0 GHz processor machines running Linux 2.4.18 kernel with 512 MB SDRAM. In all our experiments, we set the measurement interval τ for obtaining the CPU loads and latencies per server as 10 seconds. Using reported data on time needed for booting the Linux operating system as well as that observed on our testbed machines, we set the boot time M_{boot} at 0.5 minutes.

For our experimental workload, we utilize the TPC-W benchmark [7] which represents the workload characterizing an online bookstore site. Specifically, our client emulator generates the *browsing mix* consisting of 95% read queries and 5% writes. We also performed experiments using the other two mixes proposed in TPC-W, the shopping and ordering mixes which have higher percentage of writes and the results followed the same trend as with the browsing mix.

B. WARD Delay-Load Curve

Since the dispatcher is implemented in front of the database tier, we next present the offline technique to configure the redirection policy with information about expected query response time under varying database CPU loads. In an offline experiment, we measure the response time as a function of CPU load, a key input to the per-query redirection policy. We use one cluster with access to one local database server. The execution time for a query depends on the number and type of other queries executing at the same time on the database server, which can be abstracted as the workload entering the system. Hence, we vary the CPU load on the database server by increasing the number of clients. In each case, we measure the mean execution time for each of the 30 read-only MySQL queries. The resulting delay-load curve as illustrated in Fig. 6(a) is then used in the *per-query* redirection policy.

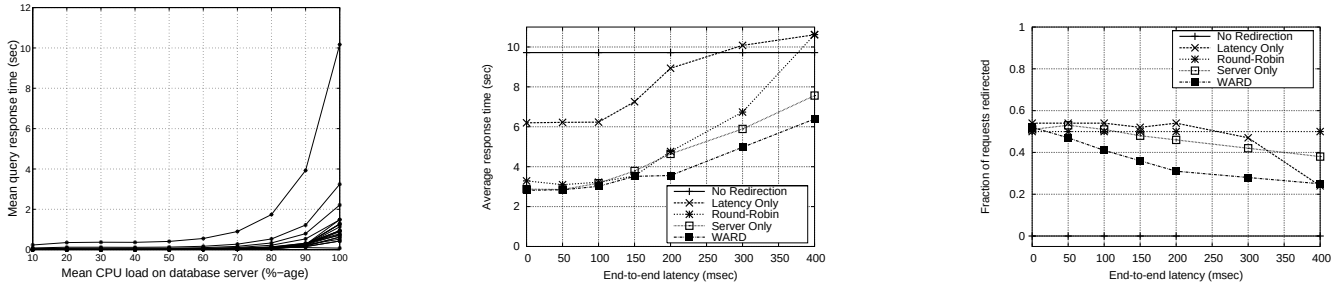
C. WARD Performance Evaluation

Using our experimental test bed, we first validate the efficacy of WARD in improving the cluster performance owing to its ability to perform a per-query redirection while taking both server loads and network latencies in to account. In this experiment, we use a set up in which the local cluster's web tier is over-provisioned with servers such that it never becomes the bottleneck. Both the local and remote clusters have 1 server each at their database tier. Requests only arrive at the local cluster and after being processed at the web tier, their queries may either be processed at the local database tier or redirected to the remote database tier.

We compare the performance of WARD against the following 4 different strawman algorithms: (1) *No Redirection* where all queries are processed locally; (2) *Latency Only* where queries are forwarded to the server with the least round-trip time as measured in the last measurement interval; (3) *Server Only* where queries are forwarded to the server with the least CPU load and hence which can be expected to process the query the fastest; (4) *Round Robin* where queries are forwarded in a round-robin fashion between the local and remote database servers. We refer to the last three algorithms that involve redirection of queries collectively as the redirection strawman algorithms.

Fig. 6(b),(c) shows the performance achieved using a trace generated using the browsing mix of TPC-W consisting of 100

[4]Assuming that a client-side redirection scheme were to install one proxy per Autonomous System in the world, then N can be expected to be around 39,000 [5]. In contrast, typical cluster grids such as Google consist of around 60 clusters.



(a) Delay-Load profile (b) Response time (c) Redirection ratio

Fig. 6. Subfigure (a) shows the WARD delay-load profile and subfigures (b) and (c) show the performance of WARD against strawman algorithms.

client sessions with mean inter-request arrival time per session being 4 seconds. First, Fig. 6(b) shows that the performance of all algorithms that redirect queries away from the local server is better than the No Redirection algorithm. This highlights the fact that for this workload the local database server is heavily loaded and hence it is of benefit to redirect a part of its load to the remote server. However, the performance of WARD is much better compared to all the strawman algorithms, thereby proving its superiority. Infact, as seen from Fig. 6(c), WARD achieves a better performance while redirecting the least fraction of queries to the remote server. This is on account of the better redirection decision made by WARD on a per-query basis by accounting for both the server loads and latencies.

Second, the Latency Only algorithm achieves the worst performance amongst the redirection strawman algorithms. This behavior further validates our hypothesis that for a dynamic content web site such as ours, the server loads are more important for forwarding decisions than the latencies. However, even the Server Only algorithm's performance becomes worse compared to WARD with increasing inter-cluster latencies. This is again an expected behavior since the Server Only algorithm doesn't take the latencies in to account in its redirection decision and hence is unable to make the correct redirection decision per-query. In contrast, WARD is designed to take both the server loads and latencies in to account in its redirection decision.

Third, the performance of all the redirection algorithms (WARD as well as the strawman redirection algorithms) degrades with increasing inter-cluster latencies since the latency overhead of redirection is higher per-query. Eventually, the cost of redirecting even a single query can be expected to outweigh the benefit of having a remote server with lower CPU load. Hence, the performance of the Latency Only (Round Robin) algorithm becomes worse than not redirecting at all once the end-to-end latencies are as high as 270 (375) milliseconds. In contrast, WARD can support having the remote cluster the furthest away than all the strawman algorithms.

D. Comparing WARD Analytical Model with Redirection Algorithms

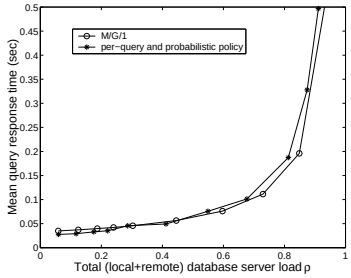
We validate the analytical model developed in Section IV by comparing against the WARD redirection policies on our testbed. Since the bottleneck tier is the database tier, we compare the redirection ratios and response time for processing queries at this tier under the model as well as on our testbed. For the model, we use Equation (6) from Section V with $\overline{x} = 42.9$ msec and $\sigma = 40.1$ msec, as measured on an unloaded database server in our testbed.

First, Fig. 7(a) compares the mean query response time of the model and the implementation on a single cluster as a function of the server load ρ. Observe that the model matches the measured query response time for $\rho < 0.7$ within $\pm 10\%$. Beyond this load, the model deviates from the implementation because: (1) our M/G/1 model makes the simplifying assumption that the arrival process of queries at the database tier is independent which may not hold true given the correlation across queries generated for the same web request; (2) our M/G/1 model doesn't take read-write conflicts into account due to which queries may take longer to process than what the model predicts and; (3) at high loads there are more queries and thereby greater number of conflicts.

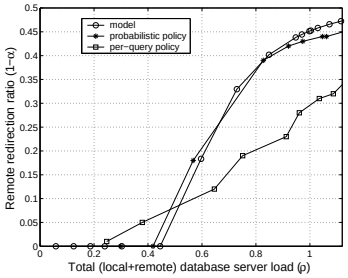
Next, we compare the model with the two implemented redirection policies: (1) *probabilistic*, and (2) *per-query*. The per-query policy receives the CPU load measurements every 5 seconds and we set the inter-cluster latency to be 25 milliseconds in all the experiments.

Fig. 7(b) and (c) compare the remote redirection ratio and query response time as a function of the system load. The redirection ratios of the model and the probabilistic policy are close because this policy bases itself upon the optimal values predicted by the model. On the other hand, the per-query policy begins redirecting earlier and redirects more queries until $\rho < 0.5$ compared to both the model and probabilistic policy. The reason for this behavior is that heavy queries are more sensitive to load as shown in Fig. 6(a), and hence it is of increasing value to redirect them at comparatively lower system loads. Hence the per-query policy performs better and exhibits a lower mean response time for $\rho < 0.5$ in Fig. 7(c). When $\rho > 0.5$, the probabilistic policy redirects more queries than the per-query policy and hence yields lower response times. We attribute this difference to the fact that the the measurement interval of 5 sec is too coarse grained to capture the small oscillations in CPU load. A better response time can be expected for smaller measurement intervals, but would require that an optimal tradeoff be established between measurement accuracy and measurement overhead.

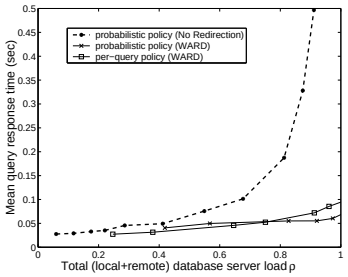
Thus, we derive two important conclusions from this experiment. First, that despite its simplifying assumptions, the M/G/1 model does match the implementation closely and hence the conclusions derived by using the model in Section V should be expected to hold true in real world implementations as well. Second, the WARD per-query redirection algorithm performs



(a) 1-cluster

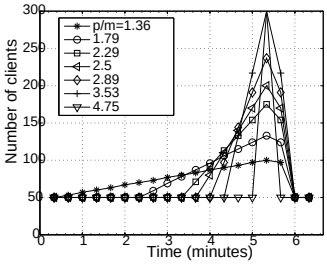


(b) 2-cluster, redirection ratio

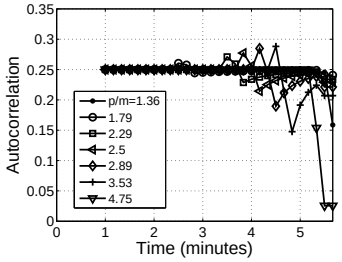


(c) 2-cluster, mean response time

Fig. 7. WARD Analytical Model Validation



(a) Session arrivals



(b) Autocorrelation

Fig. 8. Traces with different peak-to-mean variations.

better than the probabilistic algorithm under low load scenarios and its performance under high load scenarios can be improved by reducing the measurement interval.

E. Justifying Cluster Decision Algorithm

Next, via experiments on our test bed, we study the conditions under which server migration and request redirection achieve their best performance. Infact, we validate experimentally the *cluster decision* algorithm as proposed in Section III.

1) Trace Generation: We generate a set of traces with different burst size and duration by keeping the 'mean' number of sessions the same (80 sessions) while varying the 'peak' session arrivals as shown in Fig. 8(a). This yields traces with different peak-to-mean ratio for arrivals and henceforth, we refer to each trace by its respective peak-to-mean ratio. Each trace is of duration 6 minutes, with a new set of clients arriving after every 20 seconds. Furthermore, in each trace, the inter-request arrival time between two requests of a client session is 4 seconds.

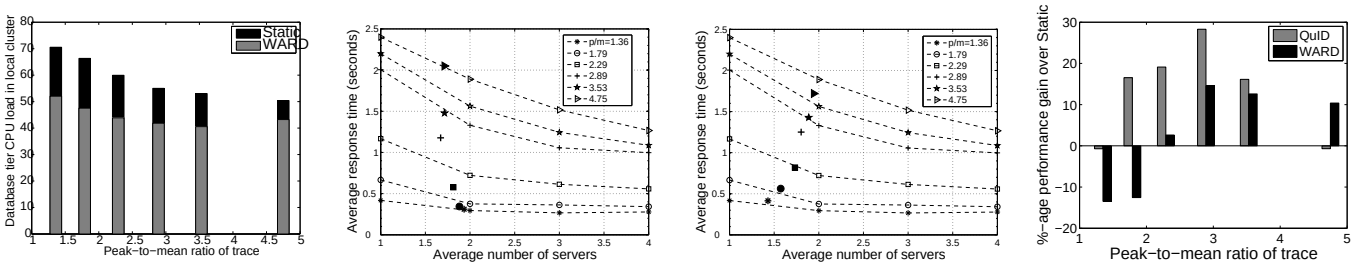
We quantify the burstiness of each trace by its autocorrelation coefficient $\hat{A}(n, m)$ computed by considering the session arrivals over the last n time units at a lag equal to the migration time of m time units. In particular, we use the value of n and m set as 0.5 minutes. Fig. 8(b) plots the autocorrelation coefficient at this lag computed at different points in time for a trace's lifetime. Note that the trace with the lowest peak-to-mean ratio (1.35) has the same value for its autocorrelation coefficient (0.25) throughout the trace. In contrast, traces with higher peak-to-mean ratio begin exhibiting drops in their autocorrelation coefficient around the time of arrival of the arrival burst. For instance, in the trace with peak-to-mean ratio of 2.29 where the burst arrives at 3.3 minutes, the autocorrelation coefficient experiences a drop at 3.6 minutes.

2) Performance Metric: Since the purpose of this experiment is to compare QuID and WARD, we need to define a performance metric to quantify the performance of these algorithms. To assist in defining the performance metric, we select a simple strawman scenario *Static allocation* in which the entire workload is processed locally while statically varying the number of database servers as $[1\text{-}4]$. Thus, we define the performance metric for QuID (or WARD) as the percentage reduction in response time compared to static allocation while using the same number of servers as static allocation. We calculate the effective total number of database servers used by WARD as: $n_{local} + \sum_{i=1}^{n_{remote}} l_i$, where n_{local} and n_{remote} are the number of servers in the local and remote clusters respectively and l_i is the CPU utilization on a remote server i that is on account of serving redirected queries only.

3) Testbed setup: We setup the testbed to consist of two clusters, with the local cluster's database tier consisting of 1 server while the remote cluster's database tier consists of 3 servers. Moreover, the local cluster has access to a free pool consisting of 3 more database servers from which QuID can potentially allocate additional servers at the local database tier. We simulate the inter-cluster latency as 100 milliseconds and the target CPU utilization μ for QuID at 40%. First, we establish the operating regime of our different traces through an experiment in which the number of servers on the local database tier is statically allocated to 1 without any redirection to the remote cluster. Fig. 9(a) shows that the traces with higher peak-to-mean ratios incur lower CPU loads on the database tier on average. However, as expected, the CPU does bottleneck towards the latter half of these traces owing to the sudden arrival of users (figure not shown).

To validate the cluster decision algorithm, we design experiments with different autocorrelation thresholds θ (see Section III-C). In one experiment, we set $\theta = 0.02$ for all traces so that the cluster decision algorithm only uses QuID to migrate additional servers (and never uses WARD) This is on account of the fact that the autocorrelation coefficients for all the traces is more than 0.02 at all times as shown in Fig. 8(b). In another experiment, we set $\theta = 0.29$ so that the cluster decision algorithm only uses WARD to redirect queries remotely. Again this is on account of autocorrelation of all traces being upper bounded by 0.29 at all times. We refer to these experiments as *QuID-always* and *WARD-always* in the rest of this section.

4) Performance: Fig. 9(b) and Fig. 9(c) depict the average response time achieved under QuID-always and WARD-always respectively under the different traces. In both these figures, a line corresponds to the performance of static allocation over a trace when the number of servers is varied. As expected, the average response time decreases on increasing the number



(a) Database CPU (b) QuID-always vs Static (c) WARD-always vs. Static (d) Gains

Fig. 9. Performance comparison of QuID-always and WARD-always.

of servers in the local database-tier. In both these figures, we also plot the point corresponding to the performance achieved by using QuID-always (or WARD-always) over a trace. Note that the points corresponding to QuID or WARD have been made larger and filled to distinguish from those representing static allocation. Thus, if the point corresponding to QuID-always appears below the line for static allocation, this is indicative of QuID achieving a better performance i.e., a lower response time than static allocation while using the same number of servers. Infact, we compute the respective performance gains of QuID-always and WARD-always compared with static allocation as shown in Fig. 9(d).

Note that with an increasing peak-to-mean ratio, QuID is able to extract increased performance gains with the largest gain exhibited for trace with peak-to-mean ratio of 2.89. However, as the traces start varying over shorter burst durations, they are also more difficult for the dynamic algorithm to track due to their more rapidly decaying autocorrelation functions. In particular, for the trace with peak-to-mean ratio of 4.75, the auto-correlation falls very low to 0.02 at the burst arrival time (see Fig. 8(b)), and hence QuID doesn't yield any resource savings. Further, observe that at the other end of the spectrum there is no performance gain when traces such as the one with peak-to-mean ratio of 1.36 don't exhibit enough variation. In such a scenario, the negative resource saving is indicative of the performance overhead paid while migrating servers in. Infact, the performance of QuID does approach that of static allocation, when we reduce the overhead dictated by M_{boot} to values lower than 0.5 minutes.

Thus, if a web cluster experiences demand surges at time scales of minutes, QuID can track the widely varying load provided that the autocorrelation values are still relatively high at lags corresponding to the key time scales of the system, namely the server migration time. Furthermore, in our experiments with QuID, we find that the time needed to apply queries to a newly migrated server M_{query} is less than 1 seconds irrespective of the traces and thus the total migration time is governed mainly by the booting time. Similar conclusions are derived on using the shopping or ordering mix of TPC-W i.e., the booting time (30 seconds) is always much greater than the time needed to re-apply the queries (1-2 seconds). Although, we can expect the total time for applying queries to be higher in a scenario where a server has been taken away from the cluster for a very large time period such that a large number of queries need to be applied to bring it up-to-date. However, in our experiments, a server was never taken out of the cluster for more than a few minutes and hence the low query application time[5].

In contrast, observe that for traces that don't exhibit enough variation (high autocorrelation coefficients), any performance gain out of redirecting queries away are offset heavily by the inter-cluster latencies and thereby redirection via WARD performs worse when compared to server migration via QuID. However, the cluster decision algorithm's hypothesis to use WARD in scenarios with low autocorrelation coefficients is verified on observing that the performance via WARD (+10.38%) is better than QuID (-0.7%) in the trace with peak-to-mean ratio of 4.75 whose autocorrelation coefficient falls to as low as 0.02 at times. Thus, overall QuID seems to perform better for cases when the autocorrelation coefficient at lag equal to migration time is larger than 0.02 at all times. In other words, we experimentally establish the value of the autocorrelation threshold θ to be 0.02 for the workload and setup used in this experiment.

To summarize, this experiment validates the cluster decision algorithm. Dynamic server allocation through QuID achieves better performance gains when the workload has variations over durations on the same order of magnitude as server migration time. In contrast, wide-area redirection through WARD achieves higher performance gains than QuID when the workload has bursts over duration smaller than the migration time-scales.

VII. Related Work

Approaches to minimize web access times can be separated into different groups: resource vs. request management and, for the latter, client-side vs server-side redirection. In lieu of this classification, QuID and WARD provide resource and request management respectively and WARD is a server-side redirection policy.

A number of related approaches have been proposed to address the performance limitation of static server allocation policies in references [9], [22]. Each offers a notion of utility computing where resources can be acquired and released when and where they are needed. Such architectures can be classified as employing *shared server utility* or *full server utility* models. With the shared server utility model, many services share a server at a time, whereas with the full server utility model, each server offers one service at a time.

Shared server utility models [22], [42], [12], [46], [43] consider a fine-grained resource allocation problem where a

[5]While this does lead to an interesting observation that server migration time is also dependent on the time for which the server has been out of the cluster, we consider such questions as out-of-the-scope primarily due to lack of realistic workloads against which we could investigate this further.

server's physical resources of CPU, disk and memory can be divided in to finer virtual resources. These models are geared towards achieving higher performance within a physical server by optimally mapping the application instances to virtual resources. For instance, in MUSE [22] all services are run concurrently on all servers in a cluster, and a pricing/optimization model is used to determine the fraction of CPU resources allocated to each service on each server. However, shared server utility models rely on isolation of the virtual resources from each other. Owing to the lack of such partitioning support in the common operating systems, this problem was addressed by using cluster reserves in reference [12].

While shared server utility models can achieve more fine-grained concurrency, each server can only be used to host processes that are part of the same content provider. This is on account of the security implications involved in sharing content from different providers on the same physical server. This limitation is addressed by full server utility models as proposed in our earlier work [38] and realized in many products such as HP's Utility Data Center [26], IBM's Oceano [9] and Amazon's Elastic Compute Cloud (EC2) [2]. In particular, Amazon EC2 [2] provides a resizeable hosting environment that can be purchased and managed over the web. These products are representative of the changing demands of web content providers who have come to realize the benefits in outsourcing their hosting to data centers instead of managing it in-house, primarily on account of the reduced maintenance costs. On the other hand, data centers can benefit from statistical multiplexing of their resources across multiple applications. While the online and optimal dynamic resource allocation algorithms of QuID were proposed in our earlier work [38], this paper also provides a proof-of-concept implementation of QuID over an e-commerce web site hosted on a linux cluster. Furthermore, we note that the issue of web server QoS has received a great deal of attention in contexts such as web server admission control [17], [30], [34], service differentiation across different types of sessions [45], [35], [40], operating system support [14], [19] and networking support. Such techniques represent mechanisms at the request and session time scale whereas QuID operates at time scales of minutes.

Next, in the context of redirection mechanisms, a significant body of research has focused on *client-side* mechanisms such as request redirection in CDNs [44], [29], server selection techniques [21], [25], caching [31], mirroring, and mirror placement [27], [23]. These techniques are based on the premise that the network is the primary bottleneck. However, we have shown that this assumption is not applicable to dynamic content owing to the higher server processing times and the lower tolerance to server measurement errors exhibited by dynamic content. Thus, while such schemes can be applied to finding the best initial cluster, WARD's server-side redirection is essential to jointly incorporate server and network latencies.

A combination of client-side and server-side redirection is also possible and beneficial if the bottleneck is not clearly identified or varying over time. Such a combined architecture is presented in reference [20]. Their server-side redirection mechanism may redirect entire web requests using HTTP-redirection if the CPU utilization exceeds a certain threshold. While they conclude that server-side redirection should be used selectively, we see server-side redirection as a fundamental mechanism for current and future cluster architectures. Our redirection mechanism is not threshold-based, but is able to optimize cluster response times for all CPU utilization values.

In contrast to approaches in references [44], [29], [21], [25], [31], [27], [23], that are designed for static content, other approaches such as those in references [18] or Akamai's EdgeSuite [1] address server selection for dynamic content via caching of dynamic fragments. Caching can occur at either the client-side with expiration times set using cookies or at the server-side (on a reverse proxy server) with cached pages being expired on receiving database update queries. However, these solutions either result in stale data being served to the clients or add to the complexity of site development and management. Nevertheless, caching is complementary to the solution adopted by WARD. In cases where a request is not resolved from the cache, the request can be forwarded to a server (local or remote) that can process it the earliest.

Though QuID and WARD were proposed in our earlier work [38] and [37] respectively, this paper proposes a novel cluster decision algorithm that combines the two algorithms to jointly optimize cluster performance. Further, we experimentally validate the performance achieved by combining QuID and WARD over a testbed hosting an e-commerce web site.

While QuID's online algorithm and WARD's analytical model are designed for multi-tiered clusters, they model each tier in isolation. In this regards, our work can benefit from models that consider inter-tier effects such as caching of one tier affecting another tier or bottlenecks shifting from one tier to another as proposed in [41]. Moreover, we only consider the resources of server CPU and network bandwidth in our framework. Other approaches [15], [24] have modeled workload variations by considering additional resources such as disk demands along with CPU. Further, this paper solves the problem of resource allocation with respect to a cluster tier when the resources can be obtained from a shared pool. However, in full server utility data centers, it is likely that resources need to be transferred from one application to another. Infact, this problem is addressed in [16] where the authors frame the problem as a global optimization problem.

VIII. Conclusions

In this paper, we propose a suite of algorithms that ensure high performance to dynamic content applications even during overload conditions such as those during time-of-day effects or flash crowd events. The cluster performance as characterized by average response time effected on clients and average server utilization is optimized through two mechanisms. The first mechanism, QuID optimizes performance within a cluster by dynamically allocating servers on-demand while the second, WARD multiplexes resources across clusters by load-balancing requests across the clusters. We also proposed a cluster decision algorithm to decide the conditions under which QuID or WARD must be used to improve the cluster performance. Through a combination of trace-driven simulation and analytical models, we have shown the

performance savings inherent in these algorithms. Moreover, through a testbed implementation of the algorithms on an online-bookstore we also explored the time-scales at which the two algorithms should be used- QuID is well-suited for large time-scale variations that occur over periods larger than a minute while WARD can be used to handle bursts at shorter time-scales.

IX. Acknowledgements

The authors are grateful to Jerry Rolia, Roger Karrer, Willy Zwaenepoel, Cristiana Amza and Huirong Fu for the detailed discussions and the anonymous reviewers for their insightful comments which aided in strengthening this paper immensely.

References

[1] Akamai. http://www.akamai.com.
[2] Amazon Elastic Compute Cloud (EC2). http://developer.amazonwebservices.com/connect/servlet/KbServlet/download/865-102-1397/ec2-dg-2007-%03-01.pdf.
[3] Cisco System: Distributed Director. http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml.
[4] Data center power and cooling. http://www.hpl.hp.com/research/about/power_cooling.html.
[5] Exploring Autonomous System Numbers. http://ispcolumn.isoc.org/2005-08/as1.html.
[6] NISTNET: Network Emulation Package. http://snad.ncsl.nist.gov/itg/nistnet/.
[7] TPC-W: Transaction Processing Council . http://www.tpc.orgi.
[8] C. Amza, A. Cox, and W. Zwaenepoel. Conflict-aware scheduling for dynamic content applications. In *USITS*, Seattle,WA, 2003.
[9] K. Appleby et al. Oceano – SLA based management of a computing utility. In *IFIP/IEEE INM*, 2001.
[10] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *ACM ToIT*, 1(1), 2001.
[11] M. Arlitt and C. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM ToN*, 5(5), 1997.
[12] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *ACM SIGMETRICS*, 2000.
[13] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *USENIX ATC*, 2000.
[14] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *USENIX OSDI*, 1999.
[15] M. Bennani and D. Menasce. Assessing the robustness of self-managing computer systems under highly variable workloads. In *IEEE International Conference on Autonomic Computing*, Washington DC, 2004.
[16] M. Bennani and D. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *IEEE International Conference on Autonomic Computing*, Seattle, 2005.
[17] N. Bhatti and R. Friedrich. Web Server Support for Tiered Services. *IEEE Network*, 13(5), 1999.
[18] S. Bouchenak, S. Mittal, and W. Zwaenepoel. Using code transformation for consistent and transparent caching of dynamic web content. Technical Report 200383, EPFL, Lausanne, 2003.
[19] J. Bruno, E. Gabber, B. Ozden, and A. Silberschatz. The eclipse operating system: Providing quality of service via reservation domains. In *USENIX ATC*, New Orleans, Louisiana, 1998.
[20] V. Cardellini, M. Colajanni, and P. S. Yu. Geographic load balancing for scalable distributed web systems. In *MASCOTS*, San Francisco, CA, 2000.
[21] R.L. Carter and M. Crovella. Server selection using dynamic path characterization in wide-area networks. In *IEEE INFOCOM*, Kobe, Japan, 1997.
[22] J. Chase et al. Managing energy and server resources in hosting centers. In *ACM SOSP*, 2001.
[23] Y. Chen, R.H. Katz, and J.D. Kubiatowicz. Dynamic replica placement for scalable content delivery. In *IPTPS*, Cambridge, MA, 2002.
[24] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, 2004.
[25] Z. Fei, S. Bhattacharjee, E.W. Zegura, and M.H. Ammar. A novel server selection technique for improving the response time of a replicated service. In *IEEE INFOCOM*, San Francisco, CA, 1998.
[26] Hewlett-Packard. HP utility data center architecture. http://www.hp.com/solutions1/infrastructure/solutions/utilitydata/archi%tecture/index.html.
[27] S. Jamin, C. Jin, A.R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the internet. In *IEEE INFOCOM*, Anchorage, AK, 2001.
[28] R. Jump. Yacsim reference manual. ECE Dept, Rice University.
[29] J. Kangasharju, K.W. Ross, and J.W. Roberts. Performance evaluation of redirection schemes in content distribution networks. *Computer Communications*, 24(2), 2001.
[30] V. Kanodia and E. Knightly. Multi-class latency bounded web services. In *IWQoS*, Pittsburg, PA, 2000.
[31] D. Karger, A. Sherman, A. Berkhemier, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. In *World Wide Web Conference*, 1999.
[32] L. Kleinrock. *"Queueing Systems, Volume II: Computer Applications"*. Wiley, 1976.
[33] J. G. Koomey. Estimating total power consumption by servers in the us and the world. Technical Report TR-02-390, Lawrence Berkeley National Laboratory, 2007.
[34] K. Li and S. Jamin. A measurement-based admission controlled web server. In *IEEE INFOCOM*, Tel Aviv, Israel, 2000.
[35] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. *IEEE TPDS*, 17(9), 2006.
[36] S. Ranjan. *High Performance DDoS-Resilient Web Cluster Architecture*. PhD thesis, Rice Univ., Houston, 2005.
[37] S. Ranjan, R. Karrer, and E. Knightly. Wide area redirection of dynamic content in internet data centers. In *IEEE INFOCOM*, HongKong, 2004.
[38] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *IWQoS*, Miami, FL, 2002.
[39] J. Rolia, S. Singhal, and R. Friedrich. Adaptive Internet Data Centers. In *SSGRR'00*, L'Aquila, Italy, 2000.
[40] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based internet services. In *OSDI*, Boston, MA, 2002.
[41] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *SIGMETRICS*, Banff, 2005.
[42] D. Villela and D. Rubenstein. Performance analysis of server sharing collectives for content distribution. In *IWQoS*, 2003.
[43] Vmware. http://www.vmware.com.
[44] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on cdn robustness. In *OSDI*, Boston, MA, 2002.
[45] X. Zhou, J. Wei, and C. Xu. Resource Allocation for Session-Based Two-Dimensional Service Differentiation on e-Commerce Servers. *IEEE TPDS*, 17(8), 2006.
[46] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation for cluster-based network servers. In *IEEE INFOCOM*, Anchorage, AK, 2001.

Supranamaya Ranjan (S'00, M'06) is a Senior Member of Technical Staff at Narus Inc. He received the M.S. and Ph.D. degrees from Rice University in 2002 and 2005 respectively. He served on the technical program committee for IEEE INFOCOM 2006. His research interests are in the areas of network security, anomaly detection and high-performance distributed systems.

Edward Knightly (S'91, M'96, SM'04) is a professor of Electrical and Computer Engineering at Rice University. He received the M.S. and Ph.D. degrees from the University of California at Berkeley in 1992 and 1996 respectively. He is an associate editor of IEEE/ACM ToN, served as technical co-chair of IEEE INFOCOM 2005 and received the NSF CAREER Award in 1997 and the Sloan Fellowship in 2001. His research interests are in the areas of mobile and wireless networks and high-performance and denial-of-service resilient protocol design.