Future Generation Computer Systems 28 (2012) 24–35

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Overlay network resource allocation using a decentralized market-based approach☆

Jay Smith [a,b,*], Edwin K.P. Chong [b,d], Anthony A. Maciejewski [b], Howard Jay Siegel [b,c]

[a] DigitalGlobe, Longmont, CO 80503, USA

[b] Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523–1373, USA

[c] Department of Computer Science, Colorado State University, Fort Collins, CO 80523–1373, USA

[d] Department of Mathematics, Colorado State University, Fort Collins, CO 80523–1373, USA

ARTICLE INFO

Article history:
Received 18 March 2011
Received in revised form
10 June 2011
Accepted 16 July 2011
Available online 27 July 2011

Keywords:
Robustness
Dynamic resource management
Market-based resource allocation
Heterogeneous computing
Overlay network

ABSTRACT

We present a decentralized market-based approach to resource allocation in a heterogeneous overlay network. This resource allocation strategy dynamically assigns resources in an overlay network to requests for service based on current system utilization, thus enabling the system to accommodate fluctuating demand for its resources. Our approach is based on a mathematical model of this resource allocation environment that treats the allocation of system resources as a constrained optimization problem. From the solution to the dual of this optimization problem, we derive a simple decentralized algorithm that is extremely efficient. Our results show the near optimality of the proposed approach through extensive simulation of this overlay network environment. The simulation study utilizes components taken from a real-world middleware application environment and clearly demonstrates the practicality of the approach in a realistic setting.

1. Introduction

Recently, information technology systems have begun to rely heavily on the concept of Services Oriented Architecture (*SOA*). SOA is a means of leveraging existing applications as services within a distributed computing environment to develop new applications. A framework commonly used to integrate existing applications is known as the enterprise services bus (*ESB*) [1]. According to the Business Integration Journal [2], "The [ESB] supports the unifying integration infrastructure required for SOA and heterogeneous environments". By relying on an ESB to communicate with service providers, service requesters need not depend on the details of service provider implementations, e.g., the physical location of the service provider. Instead, service requesters depend on the abstract definition of the service that they are using and trust the ESB to forward their requests to an appropriate service provider. In our environment, we model the ESB as an overlay network that interconnects service providers and service requesters. Request forwarding is provided by a *broker* service that is deployed within the overlay network.

Because the service requester relies on the broker service and is not explicitly dependent on any single instance of a service provider, multiple service providers could be deployed within the ESB to provide additional capacity for a service that is in high demand. In this context, *capacity* is defined in terms of the number of service requests that can be processed per unit time. To take advantage of the added capacity provided by these additional service providers, the broker service deployed within the ESB serves as a proxy for communication between the service requesters and the service providers. That is, the broker service allocates incoming service requests to the collection of service providers for processing.

Although the ESB is commonly deployed as a single centralized platform, the scalability and reliability of the ESB can be greatly improved through replication of the platform. A common approach to increasing the reliability of a system is to duplicate that system many times across many hardware deployments, a technique often referred to as *replication* [3,4]. An important aspect of an ESB implementation is that it can be decentralized both to increase its reliability and to ensure its scalability [2,4].

Successfully replicating brokers within an ESB requires maintaining network transparency [3]; i.e., the user of the system

☆ This research was supported by the NSF under Grants CNS-0615170, CNS-0905399, and ECCS-0700559, and by the Colorado State University George T. Abell Endowment.

* Corresponding author at: DigitalGlobe, Longmont, CO 80503, USA. Tel.: +1 720 839 5378.

E-mail addresses: jtsmith@digitalglobe.com (J. Smith), echong@colostate.edu (E.K.P. Chong), aam@colostate.edu (A.A. Maciejewski), hj@colostate.edu (H.J. Siegel).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2011.07.002

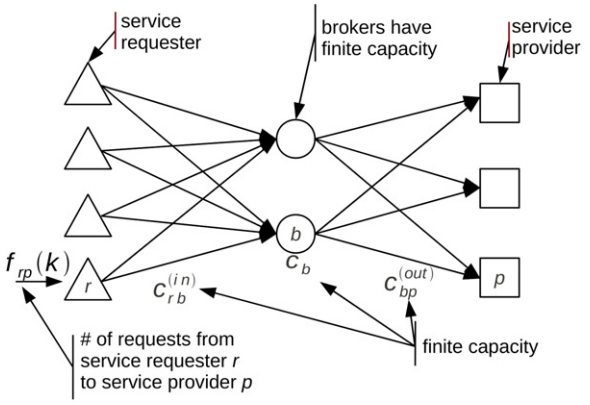


Fig. 1. An example system where four service requesters are utilizing two brokers to communicate with three service providers. Service requesters are shown as triangles, brokers as circles, and service providers as squares. Each input link to a broker from a service requester has a finite capacity, denoted $c_{rb}^{(\text{in})}$. Likewise, each output link from a broker to a service provider has a finite capacity denoted $c_{bp}^{(\text{out})}$. Finally, each broker has a finite capacity for servicing requests, denoted c_b.

should be shielded from the existence of any redundant components used to provide the ESB or its attached services. To the user, the broker service should appear as a single highly available system that always has sufficient capacity to route service requests. Achieving this kind of transparency in service delivery requires a mechanism for allowing the broker service to adapt dynamically to the changes in system load. That is, transparently utilizing replica service providers deployed within the ESB requires that the broker provide a mechanism for routing requests to their physical destination that can be decentralized. In this work, we focus on the allocation of resources to the effective transmission of service requests to service providers.

In this work, we investigate the application of a decentralized market-based approach to resource management within a heterogeneous deployment of an ESB platform. Service requesters send tasks to service providers using an overlay network provided by the ESB. The decision of how to allocate broker capacity to service requests is made in a decentralized manner based on a quantification of current resource demand relative to current system capacity. Individual service requesters select transmission rates through Brokers that maximize their individual benefit, while the brokers adjust ''prices'' for network links and broker computing capacity to reflect current demand for shared resources. Thus, price setting enables service requester decision making by enabling shared resources to communicate a simple quantification of current system congestion to requesters.

Fig. 1 presents a graphical model of a simple overlay network provided by an ESB. Service requesters are depicted in the figure as triangles, service providers as squares, and brokers as circles. Each service requester is connected to a collection of brokers that ''service'' requests by dispatching them to a service provider capable of completing the request. The number of requests produced by each service requester may vary with time according to some unknown process. The capacity of the brokers to service incoming requests may differ from one component to another, i.e., the collection of brokers are assumed heterogeneous in their performance [5–7]. Finally, each broker is connected to a collection of service providers by a finite capacity link.

Our mechanism for resource management can be thought of as a market-based approach where market demand for shared resources helps the system to set prices for those resources. Some market-based approaches rely on an auction to create a market where prices are set by the highest bidder [8–12]. In this environment, the service requesters, that would be bidding for service in an auction based mechanism, are part of the resource management mechanism and are cooperating to maximize the utility of the overall system. Because the service requesters are not adversarial participants in the system the complexity of an auction based mechanism is not required.

Our approach utilizes price setting based on duality theory [13]. In duality theory, selected constraints are directly accounted for in the optimization criterion as a penalty. This approach is analogous to that used in congestion control on the Internet [14], in ad-hoc sensor networks [15–17], and server provisioning within large scale distributed clusters [18]. In our system model, price variables are introduced to model market demand for shared resources, where prices provide a simple quantification of demand relative to supply. For example, as the number of requests through a broker increases, the broker raises its ''price''. Conversely, if the number of requests through a broker decreases, its quoted price also decreases. In addition to measuring the demand for the component itself, the broker also is responsible for stating the demand for the links from that broker to all of the service providers with which the broker can communicate. The procedure used to calculate optimal prices for resources is presented in Section 3.

In this study, we are considering an environment where all brokers are controlled by a single organization. As such, the brokers are assumed to be cooperating to provide an efficient system. We assume that all brokers use the common pricing mechanism presented in Section 3 and as such do not bias their pricing by over or under stating demand for shared resources. Individual service requesters directly utilize current pricing information provided by the brokers to make local resource allocation decisions about how best to assign their volume of requests within the overlay network, given current network utilization. Obviously, this requires that each service requester is logically connected to every broker within the system. Using all of these components in concert, we will show that our decentralized mechanism results in provably optimal resource allocations.

In the replicated broker environment, we assume that each service provider offers a unique service to the overall system and that each service provider always has sufficient capacity to service all incoming requests. In a real system, a service provider may be implemented using a collection of finite-capacity replica providers, where the replicas combine to provide a more reliable scalable service implementation. Within each of these collections, we treat the allocation of finite-capacity service provider resources as a separate resource allocation problem. In Fig. 2, we have re-drawn the distributed environment to show how the collection can be used to provide such a service to the system. When treated in isolation, the allocation of service provider capacity within each collection is simpler than our original distributed broker problem because there is only one class of shared resource to manage.

To demonstrate resource allocation in this simpler environment, we apply our approach to a related distributed web hosting environment [19]. In a distributed web hosting environment, the hosted web site is replicated to multiple web servers to increase the apparent reliability and performance of the web site. Incoming user-driven HTTP requests for data are routed to the web servers for processing by a collection of independent service requesters. By indirecting web server access through service requesters, we can allocate incoming user requests using a decentralized approach that is similar to that of the ESB environment. The overlay network topology of this environment includes only service requesters and service providers, where a service provider is defined to be a web server. This two-layer overlay network can also be used to model the allocation of requests by brokers to replicated service providers as in our previous example. That is, in this simpler model, the brokers act as service requesters to a collection of replicated service providers.

The contributions of this work include a new mathematical model of resource management in an overlay network that treats the allocation of shared resources to service requests as

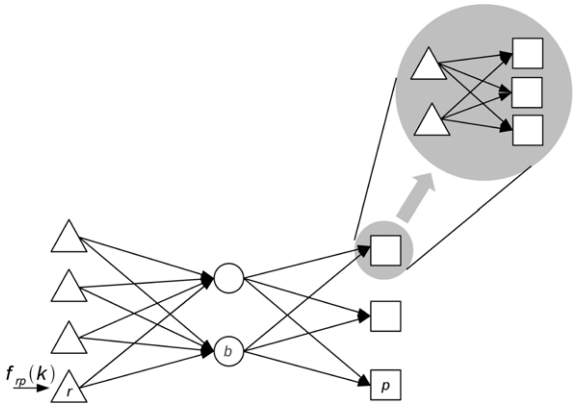


Fig. 2. An example use of finite-capacity service provider resources to implement a scalable reliable service within the context of the distributed ESB system. We have expanded the depiction of a service provider to reflect the added complexity of providing a scalable, highly available implementation through replication. In this example, the ESB environment operates as before and the added complexity within a service provider can be treated as a separate allocation problem, called out in the figure in gray. In this sub-problem, the internal service requesters directly allocate incoming requests to the internal service providers.

an optimization problem. We also present a new decentralized market-based approach to resource management within an ESB environment and a distributed web hosting environment. Through comparison with a known optimal technique, we demonstrate that our decentralized approach to resource management is capable of providing near optimal resource management in an ESB setting. Although an analogous methodology has been studied previously in the context of Internet congestion control [14], it has never been applied within the context of an ESB or to route service requests in a distributed web hosting environment.

In the next section, we present an example derivation of a decentralized routing mechanism for the distributed web hosting environment. Using the concepts developed in this example environment, Section 3 provides a more detailed view of the ESB system model and develops an analogous decentralized approach. In Section 4, we analyze the robustness of this approach to resource allocation in the ESB environment. Sections 5 and 5.2 present our simulation setup and results for the evaluation of this approach in a real-world setting.

2. Web hosting example

2.1. System model

The environment in this example is that of a heterogeneous, distributed computing system designed to service a high-volume web site of world-wide interest. The system being modeled was used to implement a portion of the 1998 World Cup web site [19] that processed more than 1.3 billion HTTP requests during the summer of 1998. The web site was provided to a world-wide audience by four heterogeneous, geographically dispersed systems, each with their own processing capacity and workload distribution techniques. This class of system is very challenging to implement but occurs surprisingly frequently. The World Cup football tournament is just one example of an event of world-wide appeal that necessitates web-based coverage. Sites of this type are typically constructed for specific events, and the volume of traffic is on the order of billions of requests processed in a period of only a few months or less. Other such events that are reasonably expected to draw the attention of a world-wide audience in the billions might include the Olympics or the Tour de France.

Mapping requests to service providers is challenging because of the large volume of requests that must be processed. To help cope with this large volume of requests, the providers of the World Cup web site [19] chose a hierarchical deployment, where multiple instances of the distributed web hosting environment are deployed throughout the world. This work focuses on the application of our decentralized resource allocation mechanism to the design of a single instance of a distributed web hosting environment.

Fig. 3 presents a graphical depiction of the distributed web hosting environment employed to deliver the 1998 World Cup web site. In this example system, users send requests for data to a web site over the Internet. These incoming requests are first routed to an instance of the web hosting environment by a network-layer load balancer (e.g., [20]). A *request* in this environment is defined to be a menu-driven HTTP request for data that originates with a user. When incoming requests arrive to a web-hosting environment, they are placed into the input queue of a service requester within that environment. Acting on the user's behalf, service requesters remove incoming requests from the input queues and route them to a service provider (i.e., web server) for processing. The service provider processes the request and returns the results to the user.

In our model, each distributed web hosting environment utilizes a collection of *service requesters* to route incoming requests to web servers for processing. Each web server is defined to be a *service provider* capable of servicing any incoming request.

Let $\mathcal{R}$ denote the set of all service requesters and let $\mathcal{P}$ denote the set of all service providers. Each service requester r ($r \in \mathcal{R}$) routes each incoming request to a service provider p ($p \in \mathcal{P}$). Service requester r produces a variable number of requests at each time-step k, denoted $f_r(k)$, based on the arrival rate of user driven requests. Each link connecting service requester r and service provider p is assumed to have a finite capacity for moving requests, denoted $c_{rp}^{(\text{in})}$. Finally, each service provider p can only process a limited number of requests in each time-step, denoted c_p.

For each service requester r, the fraction of incoming requests that are sent to service provider p is denoted η_{rp}. In this example, each service requester r uniquely quantifies the value of using each service provider p with a simple scalar measure of value, denoted ζ_{rp}, e.g., speed. We expect that the ζ_{rp} values can be set by the system implementer based on the details of their deployed system. For a given service requester r and service provider p, we can define an aggregate *service quality* delivered by the system as:

$$\sum_p \eta_{rp}\zeta_{rp}. \tag{1}$$

Intuitively, the service quality delivered by the system for traffic sent by service requester r can be thought of as the weighted average route quality realized for this traffic, where the η_{rp} values provide the weights. Finally, the system realizes some utility from delivering service requests to their destination. We account for differing service quality by quantifying utility, denoted $U(x)$, as the worth of receiving service quality x. In our model, we have chosen a utility function that depends on a scalar quantification of service quality. However, our technique is applicable to a more general utility function that accepts the full vector of η_{rp} and ζ_{rp} values to compute a scalar utility value.

2.2. Centralized optimization

We initially model resource allocation in this system as a constrained optimization problem that will serve as the basis for the derivation of our decentralized approach to resource allocation. We combine the elements of our system model presented in the previous subsection to form the following optimization problem:

$$\text{maximize} \quad \sum_r f_r(k) U\left(\sum_p \eta_{rp}\zeta_{rp}\right) \tag{2}$$

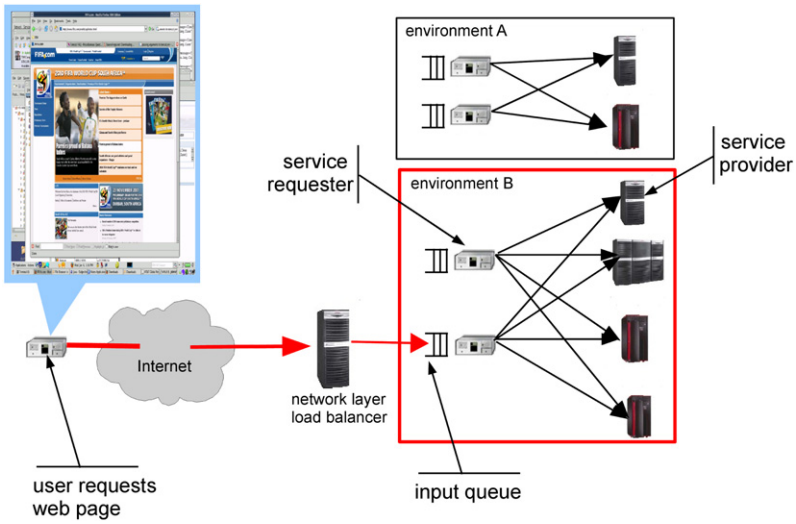


Fig. 3. An example deployment of a distributed web-hosting environment. The system is designed to provide a web-site of world-wide appeal. Users send requests for data to a web site over the Internet. These incoming requests are first routed to an instance of the web hosting environment by a network layer load balancer (e.g., [20]). Incoming requests are queued for a service requester within the environment. The service requester applies our decentralized allocation mechanism to route each incoming request to one of the service providers. The service provider processes the incoming request and returns the results to the user.

$$\text{subject to:} \quad \forall r, \sum_p \eta_{rp} = 1 \tag{3}$$

$$\forall r, p, \quad \eta_{rp} \geq 0 \tag{4}$$

$$\forall r, p, \quad \eta_{rp} f_r(k) \leq c_{rp}^{(\text{in})} \tag{5}$$

$$\forall p, \quad \sum_r \eta_{rp} f_r(k) \leq c_p. \tag{6}$$

Intuitively, for each service requester/service provider pair, Eq. (2) sums the realized utility of the service quality scaled by the production rate for that service requester. By taking this sum over all service requester/service provider pairs, we are evaluating the total realized utility for the system at time-step k.

The four constraints that must be satisfied for this optimization enforce the capacity limitations of the system. First, Eq. (3) requires that, for each service requester r, all of the requests received in a given time-step are sent to a service provider for processing, and Eq. (4) enforces the fact that negative decision variables are meaningless in this context. Eq. (5) enforces the constraint on processing capacity for each service provider. Eq. (6) enforces the constraint on capacity for each service provider.

The solution to this constrained optimization problem provides an optimal allocation of system resources at any given time-step k. The solution can be applied by a centralized resource manager to allocate requests to service providers. However, because the production rates of requests change from one time-step to the next, the optimal allocation of requests also changes from one time-step to the next. In a problem of reasonable size, the time required to solve the centralized constrained optimization problem may be longer than a single time-step. That is, because the optimal allocation changes with time, a centralized approach may continually lag behind the optimal allocation.

2.3. Decentralized approach

To transform the original optimization problem described in the previous subsection into a decentralized algorithm for resource allocation, we apply the Lagrangian multiplier method [21,22,13]. In this way, we convert the centralized optimization problem into an equivalent problem that can be separated into $|\mathcal{R}|$ independent sub-problems where each of the sub-problems is much simpler to solve. The constraint of Eq. (6) in the centralized problem reflects a constraint on *shared* resources. The Lagrangian method provides a mechanism for introducing Lagrange multipliers (interpreted as "prices") for these shared resources that can be directly accounted for during optimization as a penalty. An important feature of this approach is that solving for the η_{rp} values for each service requester no longer requires detailed knowledge of the η_{rp} values of the other service requesters. These detailed values are instead replaced by a collection of prices produced by the service providers, where each price reflects the current demand for the capacity of that service provider. Let $\phi_p(k)$ denote the price of using service provider p ($\forall p \in \mathcal{P}$) at time-step k. Each service requester r must choose η_{rp} values to solve the following problem:

$$\text{maximize} \quad f_r(k) \sum_p \left[U(\eta_{rp} \zeta_{rp}) - \phi_p(k) \eta_{rp} \right] \tag{7}$$

$$\text{subject to:} \quad \sum_p \eta_{rp} = 1 \tag{8}$$

$$\forall p, \quad \eta_{rp} \geq 0 \tag{9}$$

$$\forall p, \quad \eta_{rp} f_r(k) \leq c_{rp}^{(\text{in})}. \tag{10}$$

The above optimization problem only depends on the prices obtained from each service provider and information that is locally available to the service requester. The maximization of this simpler problem is modified from the centralized approach to account for the cost of using each service provider. In this way, the constraint on service provider capacity is accounted for as a penalty as opposed to a direct constraint [13]. However, the remaining constraints on local resources must still be enforced as before.

By reformulating the centralized optimization problem in this manner, we are able to obtain a collection of sub-problems whose combined solution is mathematically equivalent to the solution of the original centralized optimization problem. The final required ingredient in this model is an algorithm for computing the appropriate price for the consumption of capacity at each service provider.

Each service provider p is required to update the price in the model for its capacity such that the price reflects expected near-term future demand for the resource. The price of each shared resource reflects the amount of excess capacity that the resource has for processing requests. If the demand for a shared resource is greater than the supply, then the price of the resource should

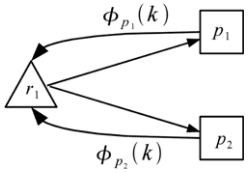


Fig. 4. An example of the price update procedure. In the example, service requester r_1 receives updates for the price of using service provider p_1 and service provider p_2.

increase. Conversely, if the demand is less than the supply, then the price should decrease. The price of each shared resource is updated according to the difference between the capacity of the shared resource and current demand for the resource scaled by a constant factor. In our model, we introduce a constant step-size parameter for controlling price updates for service provider capacity, denoted ν, whose value must be greater than or equal to 0. Each service provider p can simply update prices $\phi_p(k)$ using the following update procedure where $[x]_+ = \max\{x, 0\}$:

$$\phi_p(k+1) = \left[\phi_p(k) - \nu\left(c_p - \sum_r \eta_{rp} f_r(k)\right)\right]_+ .$$

The update procedure is intentionally expressed in terms of abstract time-steps. Each pair of successive time-steps can be mapped to a specific interval of real time. There are a variety of possible interpretations of the real time that passes during the interval between any pair of successive time-steps k and $k+1$. One natural interpretation treats all of the time intervals as having the same constant length, i.e., each interval corresponds to the same amount of real time.

The step-size ν determines how the system will react to fluctuations in demand for shared resources. Notice that the step-size determines the magnitude of price updates. That is, if ν is too small, then the prices will be slow to react to changes in demand. For example, if price updates are too small and the demand for a resource is much greater than its capacity, then the price for the shared resource may not increase enough to deter future requesters from using it. Consequently, the number of queued requests in the input queue of the shared resource may increase because it is consistently receiving more requests than it can process. The value ν needs to be large enough to enable the system to react to substantial changes in demand, i.e., increase the price enough to deter recurrent excessive demand. Care must also be taken to prevent the opposite scenario, where prices are increased so much that future demand for the shared resource is unnecessarily reduced to 0. That is, if ν is set too high, then the system may "thrash", i.e., demand may oscillate between shared resources potentially overwhelming some resource in any given time-step.

An example to illustrate communicating prices is shown in Fig. 4. In this example, service requester r_1 is sending requests to service providers p_1 and p_2. To facilitate the allocation decisions made by r_1, each of the service providers provide prices for using each of the shared resources that it is responsible for in the system, e.g., the capacity of service provider p_1.

3. ESB environment

3.1. System model

In this section, we introduce a more complex system model involving multiple shared resources. This example system is based on modeling an ESB as an overlay network (shown in Fig. 1). In this computing system, there are three types of entities: service requesters, brokers, and service providers. As in our previous example, service requesters send requests to service providers; however, in this example, requests are routed to the service providers by a collection of independent brokers acting as proxies for the service providers. In our previous example, all the service providers performed the same function; e.g., they all served up pages from the same web-site. In this example, service requesters request a service by name and it is the responsibility of the brokers to route service requests to a service provider capable of processing the request. It is helpful to consider a service requester in this context as an agent that is making requests to an ESB on behalf of an application that is external to the ESB system. That is, an external application passes requests to the service requester which makes routing decisions to brokers on behalf of the application. In this way, the service requester can be treated as a component of the overall ESB infrastructure instead of as an outside entity.

Let $\mathcal{B}$ denote the set of all brokers, let $\mathcal{R}$ continue to denote the set of all service requesters, and let $\mathcal{P}$ continue to denote of the set of all service providers. In this system, rates of requests from individual service requesters are not required to be observable; instead, the system only requires that each broker be able to measure the number of requests that have arrived in each time-step—a value readily available by inspection of the input queue of each broker. In this system, each service provider is assumed to provide a unique service to the system. Thus, each service requester r ($r \in \mathcal{R}$) produces requests for a service provider p ($p \in \mathcal{P}$) at a rate of $f_{rp}(k)$ requests per time-step k. Service requesters are modeled as having an output queue, and all requests produced by the requester are written to this output queue prior to transmission. Requests are transmitted from the output queue of the service requester using the appropriate overlay network link to the input queue of the selected broker. Each broker processes requests from its input queue, forwarding each request to its chosen service provider.

In the example of Fig. 1, each service requester is connected to each broker, i.e., there is a finite-capacity link connecting each service requester to each broker, where the capacity of each link is modeled as a constraint on the transmission rate through that link. Thus, a link between a service requester $r \in \mathcal{R}$ and a broker $b \in \mathcal{B}$ is subject to a capacity constraint $c_{rb}^{(\text{in})}$ such that the rate of requests sent from service requester r to broker b cannot exceed $c_{rb}^{(\text{in})}$ in any given time-step.

Requests received by a broker are assumed to be queued for processing in a finite-capacity input queue for that broker. Each broker also maintains a finite-capacity output queue for storing requests that have been processed and are ready to be transmitted to an appropriate service provider. Each broker $b \in \mathcal{B}$ is subject to a capacity constraint on its computing capabilities such that the rate at which each broker b can process requests is limited to c_b. That is, a broker b with processing capacity c_b can move at most c_b requests from its input queue to its output queue in a single time unit.

The outgoing links from each broker to its attached service providers are subject to capacity constraints. The link connecting a broker b to a service provider $p \in \mathcal{P}$ has a finite capacity $c_{bp}^{(\text{out})}$ to move data from the output queue of broker b to the input queue of service provider p. Finally, in this example, there are no modeled constraints on the capacity of the service providers. That is, for this model we chose to focus on applying the market-based resource allocation mechanism to the broker layer. However, it should be clear, based on our earlier presentation of the web hosting example, that the presented approach could easily be extended to include allocation decisions regarding multiple service providers for the same class of service. That is, by applying the model of Section 2 to the selection of service provider replicas, the capacity of each service can be extended to accommodate excessive demand.

There are two types of shared resources for this system: the brokers and the links connecting the brokers to the service

providers. For these shared resources, there is a difference between the advertised capacity (c_b and $c_{bp}^{(\text{out})}$) of a shared resource and the physical capacity of the underlying resource. The physical capacity of any given shared resource can be viewed as the "true" capacity afforded by the physical limitations of the device, i.e., the physical capacity is an upper bound on best possible performance of the device. If a system was sized such that it were required to operate at its physical limit for the duration of its execution, then if that system was ever asked to process more than that limit, it would be incapable. It is incumbent upon each broker b to construct prices from its advertised capacity and not from its true capacity. Specifically, the advertised capacity is given as some reasonable fraction of the true system capacity; e.g., we can define the advertised capacity as ρ times the true capacity where $\rho \in (0, 1)$ is often called the "load factor". By communicating availability in terms of advertised capacity, the broker is insulated from instantaneous fluctuations that occur during the normal course of system operation that might otherwise overwhelm the component. Thus the capacity for each shared resource used in our model (c_b and $c_{bp}^{(\text{out})}$) is assumed to be the advertised capacity.

The Information Technology industry is attempting to recover the cost of maintaining Wide Area Network (*WAN*) links by billing for network traffic that is transported across these links. That is, some companies are beginning to monitor WAN traffic volume in an attempt to bill customers for the amount of data transferred across these expensive network links. In a globally distributed system such as our modeled ESB environment, network link costs need to be accounted for during resource allocation. By introducing a pricing scheme for WAN links, the network provider has created a situation where some links are more valuable than others. To help for illuminating the impact of such a decision, consider the following simple example. Using the model of Fig. 1, assume that one broker is physically located in Fort Collins, CO, in the USA, and another is located in Bangalore, India. If a service requester located in the USA intends to communicate with a service provider also in the USA, then the lowest cost route for this traffic may be through the broker located in Fort Collins. In this example, by sending traffic through the USA based broker, a network charge can be avoided. Although somewhat exaggerated, this example illustrates the heterogeneity of the available routes. The system model accounts for this heterogeneity by incorporating a quantification of route quality into the optimization problem. The quality of each route from service requester r through broker b to a service provider destination p is given by a single numeric value, denoted s_{rbp}, quantifying the quality (e.g., speed) of the route from the perspective of each service requester. Just as the ζ_{rp} values are provided by a system implementer in our previous example, we expect that the s_{rbp} values can be determined by the system implementer based on the details of their deployed environment.

The goal of this model is to help ascertain an optimal allocation of brokers and associated links to service providers for transmitting service requests. Recall, each service requester r produces $f_{rp}(k)$ service requests per time-step k for a particular service provider p. This traffic is sent through some broker in $\mathcal{B}$ and all of the traffic must be eventually transmitted to its destination, e.g., to some service provider p. Thus, we can identify the percentage of requests from a service requester r sent to a service provider p that are transmitted through broker b, denoted g_{rbp}. The goal of a resource management heuristic in this environment is to choose a combination of the g_{rbp} values such that a system-wide goal is maximized.

In this example, we assume that a service requester receives some utility from the successful transfer of a request to a service provider. For example, the service provider may provide a printer repair service—the data being transferred by the service requester might be a notification of a printer outage. Delivering this service request to an appropriate service provider enables the provider to dispatch a technician to repair the broken printer. That is, each service requester can realize some quantifiable utility from each request that is successfully delivered through the system. Additionally, some service requesters may be considered more important than others. For example, the system provider may wish to provide a higher level of service to some special customers than what is normally offered. To model this behavior, we assume that the system implementer is capable of prioritizing the traffic sent by each service requester to each service provider. Let θ_{rp} denote the priority of requests sent from service requester r ($r \in \mathcal{R}$) to service provider p ($p \in \mathcal{P}$).

For a given service requester r and service provider p, we can define an aggregate *service quality* delivered by the system as:

$$\sum_b g_{rbp} s_{\text{rep}}. \tag{11}$$

Intuitively, the service quality delivered by the system for traffic sent by service requester r to service provider p can be thought of as the weighted average route quality for this traffic, where the g_{rbp} values provide the weights. As described earlier, the system realizes some utility from delivering service requests to their destination. As in our previous example, we account for varying service quality in the model by quantifying utility, denoted $U(x)$, as the worth of receiving service quality x. As in our previous example, we have again chosen a utility function that depends on a scalar quantification of service quality. However, our technique is applicable to a more general utility function that accepts the full vector of g_{rbp} and s_{rbp} values to compute a scalar utility value.

3.2. Centralized optimization

In a manner similar to our previous example, we first pose the resource management problem as an optimization problem. The resulting optimization can be directly solved using a centralized approach by a system-wide resource manager. Using the definitions and system constraints from the previous section, the following constrained optimization problem can be defined, where each g_{rbp} value is chosen such that the following objective is maximized:

$$\text{maximize} \sum_{r,p} \theta_{rp} f_{rp}(k) U\left(\sum_b g_{rbp} s_{rbp}\right) \tag{12}$$

$$\text{subject to:} \quad \forall r, p, \sum_b g_{rbp} = 1 \tag{13}$$

$$\forall r, b, p, \quad g_{rbp} \geq 0 \tag{14}$$

$$\forall r, b, \quad \sum_p g_{rbp} f_{rp}(k) \leq c_{rb}^{(\text{in})} \tag{15}$$

$$\forall p, b, \quad \sum_r g_{rbp} f_{rp}(k) \leq c_{bp}^{(\text{out})} \tag{16}$$

$$\forall b, \quad \sum_{r,p} g_{rbp} f_{rp}(k) \leq c_e. \tag{17}$$

In the proposed centralized problem, the realized utility of the overall achieved service quality is scaled by the priority and volume of the traffic. Eq. (12) expresses the overall goal of the optimization to maximize the realized utility, given a collection of service requesters each with their own priority. The constraint of Eq. (13) ensures that all of the requests for a given service requester/service provider pair are routed through the overlay network. Eq. (14) ensures that the g_{rbp} values are positive and Eqs. (15)–(17) enforce the capacity constraints for each of the constrained system resources.

This centralized approach could be a reasonable solution for the special case of a static rate of requests from each service

requester. That is, if the rate of requests produced by all requesters in the system remains constant, then it may be reasonable to calculate a solution to the centralized problem off-line. However, if the rate of requests to be processed is changing with time, then the centralized solution becomes infeasible to maintain for all systems of reasonable size because the optimal solution to the problem changes faster than the centralized solution can be calculated. Similarly, if the centralized problem requires too many decision variables, i.e., there are too many service requesters, service providers, or brokers, then it may be unreasonable to calculate the solution in advance off-line. That is, the centralized approach to resource allocation does not scale well.

In this environment, as the rate of requests sent from each service requester to each service provider, i.e., $f_{rp}(k)$, is a function of time, this centralized form of the problem must be solved whenever any of the request production rates change. However, for any problem of reasonable size, the computation time required to solve this problem makes it difficult to complete the solution prior to the request production rates changing again. In the next section, we identify an equivalent solution that is much faster to calculate and does not require central control.

3.3. Decentralized approach

To transform the centralized optimization problem into a decentralized algorithm, we apply the Lagrangian multiplier method to the centralized optimization problem to define an equivalent problem that is separable into $|\mathcal{R}|$ independent subproblems.[1] The constraints of Eqs. (15) and (16) in the centralized problem reflect constraints on *shared* resources. An important feature of this approach is that solving for the g_{rbp} values for each service requester no longer requires detailed knowledge of the g_{rbp} values of the other service requesters. These detailed values are instead replaced by a collection of price vectors that are produced by the brokers, where each price vector reflects the current demand for shared resources attached to that broker. Let $\pi_b(k)$ denote the price of broker b ($\forall b \in \mathcal{B}$) at time k and let $q_{bp}(k)$ be the price for a link from component b to service provider p ($\forall b \in \mathcal{B}, \forall p \in \mathcal{P}$) at time-step k. Each service requester r must choose g_{rbp} values to solve the following problem:

$$\text{maximize} \sum_p \theta_{rp} f_{rp}(k) U\left(\sum_b g_{rbp} s_{rbp}\right) - \sum_{b,p} (\pi_b(k) + q_{bp}(k)) g_{rbp} f_{rp}(k). \tag{18}$$

Service requester r also must enforce the following constraints corresponding to Eqs. (13)–(15) from the centralized problem,

$$\forall p, \quad \sum_b g_{rbp} = 1 \tag{19}$$

$$\forall b, p, \quad g_{rbp} \geq 0 \tag{20}$$

$$\forall b, \quad \sum_p g_{rbp} f_{rp}(k) \leq c_{rb}^{(\text{in})}. \tag{21}$$

The prices $\pi_b(k)$ and $q_{bp}(k)$ ($\forall b \in \mathcal{B}, \forall p \in \mathcal{P}$) are obtained from each broker b as input to the above model in the form of an update $\pi_b(k) + q_{bp}(k)$ ($\forall p \in \mathcal{P}$) received at every time-step k.

To complete the decentralized algorithm, we require a mechanism for computing the prices for each of the brokers and the links connecting them to the service providers. Each broker b is required to update prices in the model for its shared resources such that the prices reflect expected near-term future demand. The price of each shared resource reflects the amount of excess capacity that the resource has for processing requests. In this model, we differentiate between updates to broker prices and link prices by introducing two different constant step-size values, one for the broker prices, denoted α, and one for the link prices, denoted γ—both step-sizes must be greater than 0. The broker b can simply update prices π_b and $q_{bp}(\forall p)$ using the following update procedure:

$$\pi_b(k+1) = \left[\pi_b(k) - \alpha\left(c_b - \sum_{rp} g_{rbp} f_{rp}(k)\right)\right]_+$$

$$q_{bp}(k+1) = \left[q_{bp}(k) - \gamma\left(c_{bp}^{(out)} - \sum_r g_{rbp} f_{rp}(k)\right)\right]_+.$$

As in the previous example, care must be taken in selecting an appropriate step-size to ensure that prices react expeditiously to market fluctuations.

3.4. Resource management using this approach

Each service requester r must have a procedure for utilizing the results obtained by solving the local optimization problem for the g_{rbp} values. Because the service requester cannot send fractions of a record, we need to approximate the direct use of the g_{rbp} decision variables. The simplest mechanism for converting the g_{rbp} decision variables into a resource allocation is to interpret their values as probabilities. Recall that the g_{rbp} values are constrained to the interval [0,1]. Thus, each g_{rbp} value can be interpreted as the probability that any given record will utilize the route from service requester r through broker b to service provider p. We can apply the g_{rbp} values to the route selection process, by constructing a cumulative distribution function (cdf) where each g_{rbp} value is interpreted as the probability of selecting its associated route. The service requester then indexes into the constructed cdf using a generated uniform random number in the range [0,1]. Using the g_{rbp} values in this way will, over many sent records, approximate the direct use of the g_{rbp} values.

4. Robustness of the decentralized approach

In a real-world computing environment, network traffic is often triggered by world events outside the control of the computing system. For example, a financial market sell off could reasonably be expected to result in an increase in network traffic communicating market sell orders to the financial market. In a similar manner, changes in the volume of service requests that an ESB must process is a source of system uncertainty. That is, a resource manager responsible for allocating brokers to service requests cannot accurately predict the upcoming volume of requests that will need to be serviced.

A system can be considered robust to perturbations in system parameters, if the change in system performance due to this uncertainty is limited [23]. In this system, we utilize an overall performance measure that accounts for requester priorities, the number of requests being transferred, and the quality of the network routes being used. Requests are produced by service requesters at some rate and transmitted to brokers where they are queued before being forwarded on to their final destination. Because the system is decentralized and the production rate of service requests is changing with time, it is possible for the system to experience contention for shared system resources. When contention for shared resources occurs, potentially due to the changes in the production rate of requests, it is possible for low-priority requests to inhibit the transfer of high-priority requests causing the performance measure to degrade.

Intuitively, the robustness [23–26] of the decentralized allocation approach can be established by answering the following three

[1] A convergence proof for the Lagrange multiplier method is provided in [13].

questions. What behavior of the system makes it robust? What uncertainties is the system robust against? Quantitatively, exactly how robust is the system? Uncertainty in service request production rates can directly impact the system performance measure. In this system, we might consider a resource allocation strategy robust as long as it maintains a performance measure that is within $X\%$ of the optimal value, where X is a user defined constraint on the acceptable performance of the system. We can quantify robustness in this environment as the proximity of the system performance measure to its optimal value. Based on this intuitive understanding of robustness in this context, we can utilize the FePIA procedure [23] to derive a more formal definition of robustness.

In step 1 of the FePIA procedure, we derive a robustness requirement that clearly defines robust operation of the system. Let the *realized utility* of the system be defined as:

$$\Psi(k) = \sum_{r,p} \theta_{rp} f_{rp}(k) U\left(\sum_{b} g_{rbp} s_{rbp}\right). \tag{22}$$

From our intuitive definition of robustness, we can state that the system is robust if at each time-step k the realized utility $\Psi(k)$ remains within an acceptable range. Let $\beta_{\min}(k)$ be the smallest acceptable realized utility value where $\beta_{\min}(k)$ is expressed in terms of the optimal realized utility value as given by the centralized solution found at time-step k. That is, given an optimal utility value measured at time-step k, denoted $\omega(k)$, let $\beta_{\min}(k) = X\omega(k)$ where X is in the range (0,1). The robustness requirement for our system can be expressed as $\beta_{\min}(k) \leq \Psi(k)$. That is, if the realized utility at each time-step k remains larger than $\beta_{\min}(k)$, then the system can be considered robust.

The principal source of uncertainty in this system that may cause fluctuation in $\Psi(k)$ is the variability in the production rate of requests. Recall that the production rate of requests $f_{rp}(k)$ is not known in advance and may differ from one time-step to the next. This variability in the production rate may directly impact the realized utility and consequently $\Psi(k)$. Finally, the robustness of a resource allocation can be quantified as the smallest $\Psi(k)$ value that occurs over all time-steps k. That is, the robustness of a resource allocation, denoted Ψ, can be measured through time-step k and expressed as:

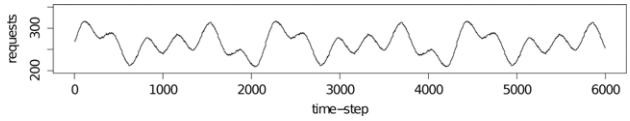
$$\Psi = \min_{k} \Psi(k). \tag{23}$$

5. Simulation study

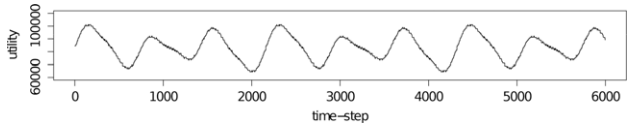
5.1. Setup

Several simulations were conducted to evaluate both the accuracy of the implemented system as well as the efficacy of the overall approach. We evaluate the approach in a realistic application of the ESB system where the production rate for each service requester varies as a function of time. The variable production rate of records in the simulation is modeled using a simple scaled sinusoid that provides a periodic change in record production unique to each service requester service provider pair. The superposition of the record production functions over all service requester service provider pairs is presented in Fig. 5(a). The initial values for each service requester service provider pair were chosen such that the initial system would be slightly under-subscribed. Each point in the plot corresponds to the number of records produced in the simulation at that time-step and presents a view of the load placed on the overall system in that time-step. The combination of sinusoids was such that the pattern of summed traffic repeats approximately every 2200 time-steps, given four service requesters and three service providers, as in our initial simulations.

In user driven systems, actual request arrival rates may depend on the application being executed, the time of day, the popularity of the web-site, etc. Traces from any one instantiation of these



(a) Production rate.



(b) Priority scaled production rate.

Fig. 5. (a) The summed production rates over all service requesters plotted versus simulation time-step; (b) the summed production rates scaled by service requester priority plotted versus simulation time-step.

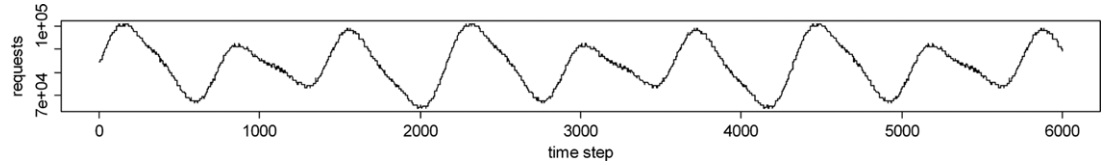
parameters may not capture the challenges posed by other possible combinations. Therefore, we use a synthetic sinusoidal-based arrival rate because its periodic changes in load pose a challenging resource allocation problem. Furthermore, the use of a synthetic workload facilitates a comparison with an optimal centralized approach.

The plot of Fig. 5(a) can be scaled according to the priority of each service requester service provider pair (θ_{rp}) producing a plot (Fig. 5(b)) of the optimal utility, given homogeneous network routes, i.e., assuming all routes are of equal value ($s_{rbp} = 1\ \forall r, b, p$). The θ_{rp} values in our initial simulations were set to increase by an order of magnitude based on the requester where service requester $r = 0$ was given priority 1. Ideally, the solution found by the described decentralized routing mechanism will track this optimal time-varying utility. The initial simulations conducted consisted of four service requesters, two brokers, and three service providers, where the collective production rate of the service requesters is varied according to Fig. 5(a).

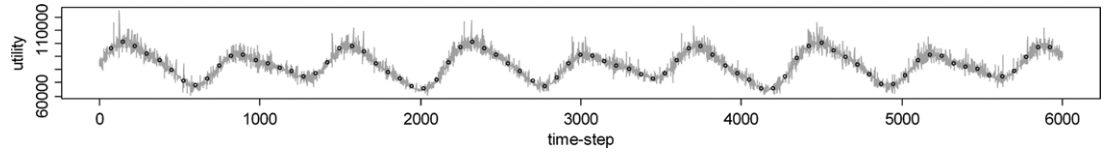
5.2. Results

To begin evaluating the decentralized approach, we first created a simulation where the quality of all routes in the system are homogeneous (i.e., $s_{rbp} = 1\ \forall r, b, p$). In this case, it is possible to compare the results of the decentralized solution to the sum of the scaled request production rate over all service requesters. Under these conditions where $\forall r, b, ps_{rbp} = 1$ and a linear utility function, Eq. (13) reduces to $\sum_{r,p} \theta_{rp} f_{rp}$, i.e., the scaled production rate plotted in Fig. 5(b). In other words, we sum the decentralized realized utility over each time-step for the entire simulation and do the same for the request production rate scaled by the priority of the requests. Ideally, these two values should be identical, given homogeneous route quality. In this evaluation, our decentralized technique realizes 99.5% of the possible scaled production rate result.

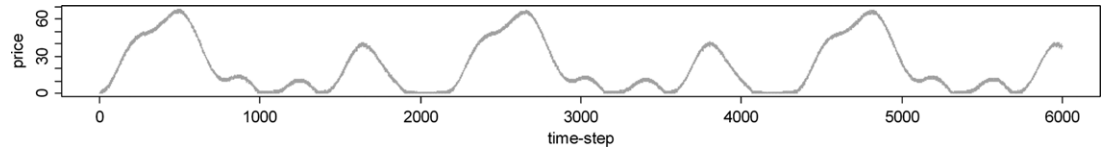
In a real-world environment, the rate of requests submitted to the system will vary with time in an unpredictable manner. To assess the viability of our approach, we modeled the production rate of requests from service requesters as a function of time. Thus, at each instant in time the optimal allocation is given by a unique optimization problem with its own unique solution. Consequently, a centralized solution in this environment is impractical because it would require recalculating the entire solution at every time-step. Alternatively, in the decentralized approach, each service requester can solve their own optimization problem locally using the prices provided by the brokers along with information that is locally available to each requester. Recall that the centralized solution provides an optimal solution as described in Section 3. The combined results of all of the service requesters should be equivalent to that found using the centralized solution, i.e., the decentralized implementation should be capable of tracking the optimal solution as it varies over time.



(a) Request production rate.



(b) Realized utility.



(c) Average shared resource price.

Fig. 6. Sample results for a homogeneous network, i.e., $s_{rbp} = 1\ \forall r, b, p$, given service request production rates that vary as functions of time. (a) The collective request production rate for the system as a function of time. (b) The realized utility for our decentralized approach (plotted as a line). Periodically, throughout the simulation, an equivalent centralized optimization problem was extracted from the state of the simulation at a given time-step and solved. These results are plotted as circles in plot (b) overlaid on the top of the decentralized solution. (c) The average shared resource price in the network as a function of time.

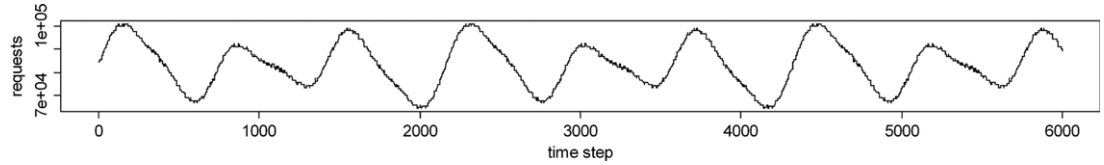
We compared the results of our decentralized approach to the solution generated by applying the centralized approach. To calculate the centralized solution at a given time-step, we extracted the information required to produce the centralized optimization problem from the details of the dynamic simulation. We solved these instances of the centralized problem offline and compared the result to the result produced by our decentralized approach. Fig. 6 presents a more detailed view of the results for the homogeneous simulation discussed earlier (where $s_{rbp} = 1\ \forall r, b, p$). The three plots of Fig. 6 present (a) the total number of requests at each time-step during the simulation, (b) the realized utility over time, and (c) the average price for shared resources in the system over time.

Embedded within the plot of realized utility (Fig. 6(b)), we have plotted the exact centralized solution overlaid as circles on the top of the decentralized solution. From the plots of the figure, we can clearly see that the decentralized solution tracks the periodic results of the centralized solution. However, there is a slight (less than 1%) fluctuation in performance due to the queuing of requests within the system. That is, each service requester/service provider pair has a different priority and in any given instant there may be minor contention for a shared resource (i.e., a broker or broker to service provider link). When this contention occurs, some high-priority requests may be delayed due to the system processing lower priority requests. In this case, the realized utility will temporarily be reduced as a result of the delay but will increase in some subsequent time-step as the delayed records arrive at their destination. Thus, the realized utility from the decentralized solution in this later time-step will appear to be greater than optimal. However, because of contention and request delays in a previous time-step, the decentralized and centralized approaches are considering different sets of requests when this apparent better than optimal behavior occurs.

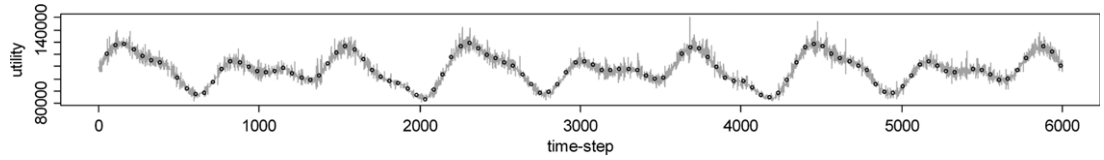
Our second simulation includes routes of differing quality, where the s_{rbp} values were different for each route. For this simulation, individual s_{rbp} values were selected at random in the range [1,10]. Recall that the s_{rbp} values appear in the utility function as a multiplier for the g_{rbp} values. In this way, the g_{rbp} values are scaled according to the s_{rbp} values prior to calculating utility. Consequently, a centralized controller might attempt to maximize the likelihood that high priority traffic will be assigned to its best route, thus maximizing the realized utility. In Fig. 7, we can see that, given heterogeneous routes, the results of the decentralized solution still effectively track the results of the centralized solution.

In both Figs. 6(c) and 7(c), we plotted the average shared resource price over the entire simulation. Notice that the prices are periodic, with a period identical to that of the production rates plotted in Figs. 6(a) and 7(a), respectively. For these simulations the system appears stable. That is, because the request production rate is periodic, a stable system would imply that the average shared resource price would return to its starting point at the end of any given period. Recall that the periodicity of the request production rate is such that the pattern repeats approximately every 2200 time-steps; similarly the average shared resource price is periodic repeating the exact same pattern every 2200 time steps. In our simulations, the initial configuration of the system represents a slightly over-provisioned system. Thus, the initial prices of the shared resources are all 0. At the end of the period, we should expect the average shared resource price to return to 0, and indeed, it does.

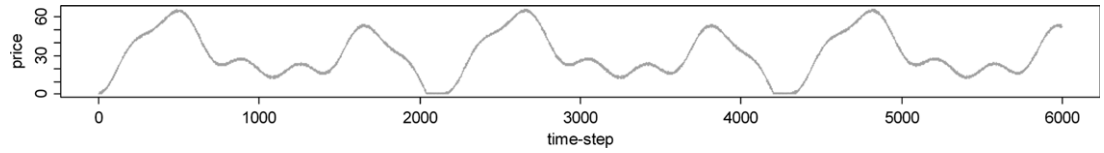
The previous two simulations were structured to facilitate a direct comparison of our decentralized approach with an equivalent centralized approach that provides a known optimal solution. To complete our simulation study, we demonstrate the performance of our decentralized approach in a more complex environment. In this simulation, four brokers are used to route requests to eight service providers where requests originate with four service requesters. The increased complexity of this simulation prohibits the use of the centralized approach. That is, using the centralized solver from our previous simulations, an attempt to solve the centralized problems in this more complex environment required several hours to calculate a solution for a single time-step in this simulation. Thus, calculating the centralized solution over the entire simulation in this more complex environment is impractical because it would require potentially thousands of hours of calculation.



(a) Request production rate.

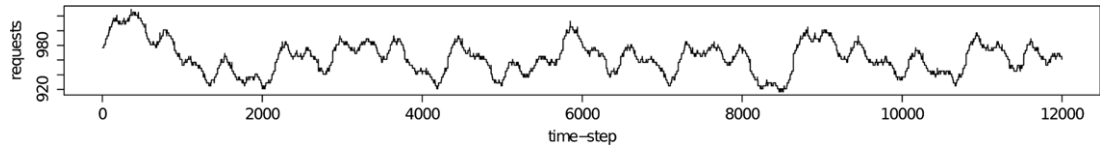


(b) Realized utility.

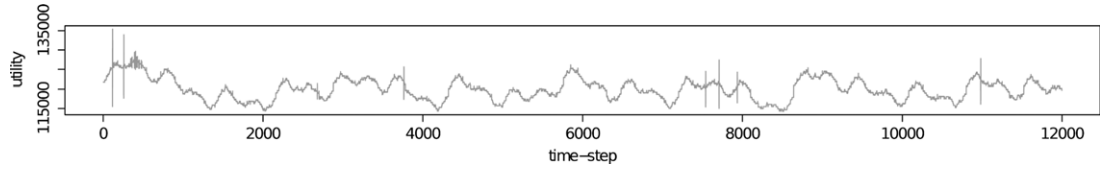


(c) Average shared resource price.

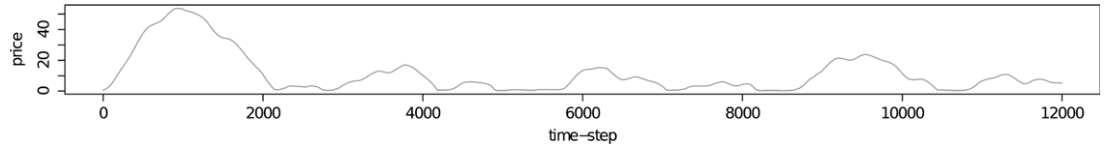
Fig. 7. Sample results for a heterogeneous network, i.e., each route through the network connecting a service provider to a service requester has a unique service quality associated with it. Plots a, b, and c correspond to the same plots in Fig. 6 for the homogeneous case.



(a) Request production rate.



(b) Realized utility.



(c) Average shared resource price.

Fig. 8. Results taken from a simulation of an environment where four service requesters produce on average 950 requests per time unit that are routed to one of the eight service providers.

Fig. 8 presents the results for this more complex simulation. In Fig. 8, plot (a) in the figure presents the request production rate per unit time over the entire simulation. For this simulation, we employed the same periodic process for generating service requests, but because we increased the number of service providers in the simulation the absolute number of requests produced by each requester is increased. In evaluating the results of this simulation, recall that each service requester is assigned a unique priority. Thus, in a successful system, the pattern of realized utility in each time unit should mimic the pattern of the request production rate. The realized utility for traffic processed by service providers during the simulation is plotted in Fig. 8(b). Notice that the realized utility throughout the simulation directly mimics the request production rate of Fig. 8(a).

Fig. 8(c) plots the average price for a resource over the life of the simulation. In these results, there is an initial period where the average price for resources is inflated due to contention for the shared resources of the system. After this initial period, the average price for a shared resource increases when the request production rate is over 950 requests per time-step and decreases when the request production rate is below this threshold. This regular pricing pattern suggests that the system represented by this simulation is inherently stable. That is, as long as the request production rate averages 950 requests per time-step then the system will be capable of servicing all requests.

Our simulation results clearly demonstrate the viability of our decentralized market-based approach to resource management in a heterogeneous distributed computing system such as the

distributed ESB environment. The plots of realized-utility in both Figs. 6(b) and 7(b) demonstrate the ability of the decentralized approach to successfully track the optimal performance of the system as defined by an impractical but known optimal approach. Finally, the results of Fig. 8 demonstrate the viability of this approach in more complex environments.

6. Related work

A related field to the study of an Enterprise Services Bus is that of a Content Delivery Network (CDN) commonly used to improve the apparent performance and reliability of web sites by distributing their content throughout the world wide web [27]. In a CDN, web-site content is cached at replica servers that are capable of replying to the web requests on behalf of the owning web site. In [28], the authors introduce the concept of a collaborative CDN (CCDN). A CCDN is described as being an overlay network that utilizes end-user machines in a peer-to-peer fashion to provide a CDN across a wide-area network. In the Globule system, user requests for data available on the CCDN are delivered to replica servers using a redirection service capable of HTTP redirection. In [29], the authors present the AS-path length heuristic used in Globule to provide a redirection policy for user requests. The AS-path length heuristic greedily redirects user requests to the closest replica server available in the CDN, where proximity is defined in terms of the number of network hops between the requester and the replica. This simple greedy approach does not account for contention among the shared resources of the CCDN, i.e., the replica servers. Because the Globule system is an instance of an overlay network it can be modeled as a trans-shipment network flow problem. By modeling an overlay network in this manner, our market-based resource allocation technique can be applied to the routing of web-site requests to replica servers based on current network load, where the proximity of the requester to the caches and the network bandwidth of the caches can be used to construct a quality value for each route in the CDN, i.e., s_{rep}. In this way, our approach can account for both the proximity of replicas to users as well as any contention for the shared resources of the CCDN, where contention is not considered in [29].

Another approach to market-based resource allocation involves the use of an auction as opposed to price setting [9,10,30,31]. The Tycoon resource allocation system presented in [11] provides such an auction based market for resource allocation in a distributed system. In the Tycoon system, users bid for the right to use compute resources within the Tycoon network. Bids are accepted by a collection of auctioneers that manage access to Tycoon compute resources. In an auction system, the auctioneer must accept bids for a resource for some period of time before closing the auction. The waiting period for an auction to close is acceptable as long as the time required to complete the auction is less than that of the task to be executed. In our environment, tasks are extremely short lived; e.g., the time required to produce a static web page or deliver a message in an overlay network. Consequently, the time delays incurred by an auction for a resource are infeasible within our context. However, prices for shared resources are set within our network based on current demand. Thus, our price setting approach is more analogous to a bid-ask auction system where the resource seller sets an asking price and the buyer accepts that price by purchasing the right to that resource. In this way, as demand for a shared resource fluctuates, so do the prices for that resource.

In [9], the authors provide a survey of existing market-based models applicable to resource management systems. In their taxonomy, our system can be described as a commodity market model where resource pricing is driven by supply-and-demand. This type of price setting model is applicable to our problem domain because of the non-adversarial nature of our distributed system. Recall that our evaluation of system performance is based on the collective realized utility that the system achieves. Recently, some authors have begun to investigate the use of auctions in resource allocation problems where the objective is to maximize the collective performance of the overall system [10]. However, in our system, the service requesters that use shared resources are not in competition with each other. For example, given two service requesters a and b the system benefits from the total realized utility gained by processing the requests from a and b. That is, the system benefits from a and b cooperating in the use of shared resources. Auction approaches to market-based resource allocation are more commonly used in environments where the service requesters have an adversarial relationship. The added complexity of an auction-based mechanism is not required in this system because of the cooperative relationship among service requesters.

In [32], a system called WebSeAl is introduced that provides resource allocation in a CDN. One of the many claims of the WebSeAl system is its ability to balance the request load for a web site across multiple geographically dispersed replica web servers. Their approach to resource management of the server pool is to introduce prices for the use of servers in their network that force the clients to route their requests based on this price information. Clients in the WebSeAl environment make routing decisions based on a combination of performance data about the response time of each replica server and a weighting factor for each replica. The authors assume that clients in the WebSeAl environment will be "sensitive" to the weighting factor and account for current system weights while making resource allocation decisions. As the authors of [32] state, the WebSeAl environment is therefore best suited to serving web sites where there is no incentive to circumvent the balancing aspects of the system, e.g., web sites delivered on a corporate intranet. By ignoring the weighting factor a client may instead request solely based on selfish performance data, i.e., always selecting the replica that provides the best possible performance to the client.

Like the WebSeAl environment our system utilizes a price setting scheme to enable clients in the system to make routing decisions. However, unlike WebSeAl, prices in our environment are set based on direct feedback from the system regarding current demand for shared resources. Furthermore, the clients (service requesters) in our system solve an optimization problem locally that leverages current prices for shared resources to account for network congestion. By solving the local optimization problem to maximize their individual utility, the system as a whole is able to maximize its realized utility. In our system, because the prices are set by current demand and utilization, there is no benefit to the client to try and "cheat" the pricing scheme, i.e., the prices in the system directly reflect the impacts of congestion on the client's selfish interests.

7. Conclusion

In this paper, we have demonstrated a technique for resource allocation in overlay network based environments that are derived from Lagrangian optimization techniques similar to those used in Internet congestion control. Our approach has some clear advantages over some obvious solutions for routing data within an overlay network. Principally, our decentralized approach is capable of producing a near-optimal assignment, and maintains the more attractive attributes of a decentralized solution, e.g., scalability and reliability. In addition, we use the model of this overlay network to derive a metric suitable for measuring the robustness of this approach.

Throughout this paper, we have assumed that a feasible solution to the centralized allocation problem exists. Future work in this area may explore problems where fluctuations in the production rate of requests results in an infeasible system. Additional research also may include combining the two example environments into a single system.

Acknowledgments

A preliminary version of part of this material was presented at the 22nd International Parallel and Distributed Processing Symposium [33].

References

[1] M.T. Schmidt, B. Hutchison, P. Lambros, R. Phippen, The enterprise service bus: making service-oriented architecture real, IBM Systems Journal 44 (4) (2005) 781–797.
[2] K. Emo, The 'enterprise-class' service bus: IT's direct route to OA, Business Integration Journal (2005).
[3] G. Coullouris, J. Dollimore, T. Kindberg, Distributed Systems Concepts and Design, 4th ed., Addison Wesley, Harlow, England, 2005.
[4] K.P. Birman, Reliable Distributed Systems, Technologies, Web Services, and Applications, 1st ed., Springer Science + Business Media, New York, New York, 2005.
[5] M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996.
[6] R.F. Freund, H.J. Siegel, Heterogeneous processing, IEEE Computer 26 (6) (1993) 13–17.
[7] A. Khokhar, V.K. Prasanna, M.E. Shaaban, C. Wang, Heterogeneous computing: challenges and opportunities, IEEE Computer 26 (6) (1993) 18–27.
[8] X. Bai, D.C. Marinescu, L. Bölöni, H.J. Siegel, R.A. Daley, I.-J. Wang, A macroeconomic model for resource allocation in large-scale distributed systems, Journal of Parallel and Distributed Computing 68 (2) (2008) 182–199.
[9] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic models for resource management and scheduling in grid computing, Concurrency and Computation: Practice and Experience 14 (13–15) (2002) 1507–1542.
[10] H. Chen, H.C. Lau, Decentralized resource allocation and scheduling via walrasian auctions with negotiable agents, in: Proceedings of 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. 2, 2010, pp. 356–360.
[11] K. Lai, L. Rasmusson, E. Adar, L. Zhang, B.A. Huberman, Tycoon: an implementation of a distributed, market-based resource allocation system, Multiagent and Grid Systems 1 (3) (2005) 169–182.
[12] M. Feldman, K. Lai, L. Zhang, A price-anticipating resource allocation mechanism for distributed shared clusters, in: EC'05: Proceedings of the 6th ACM Conference on Electronic Commerce, ACM Press, New York, NY, USA, 2005, pp. 127–136.
[13] E.K.P. Chong, S.H. Zak, An Introduction to Optimization, 3rd ed., John Wiley, New York, NY, 2008.
[14] R. Srikant, The Mathematics of Internet Congestion Control, Birkhäuser, Boston, MA, 2003.
[15] E.K.P. Chong, B.E. Brewington, Decentralized rate control for tracking and surveillance networks, Ad Hoc Networks 5 (6) (2007) 910–928. (special issue on) Recent Advances in Wireless Sensor Networks.
[16] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, A. Vahdat, Mirage: a microeconomic resource allocation system for sensornet testbeds, in: Proceedings of The Second IEEE Workshop on Embedded Networked Sensors, 2005, EmNetS-II, 30–31 May 2005, pp. 19–28.
[17] G. Mainland, D.C. Parkes, M. Welsh, Decentralized, adaptive resource allocation for sensor networks, in: NSDI'05: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation, USENIX Association, Berkeley, CA, USA, 2005, pp. 315–328.
[18] M. Stokely, J. Winget, E. Keyes, C. Grimes, B. Yolken, Using a market economy to provision compute resources across planet-wide clusters, in: Proceedings for the 23rd International Parallel and Distributed Processing Symposium, IPDPS 2009, March 2009.
[19] M. Arlitt, T. Jin, A workload characterization study of the 1998 World Cup Web Site, IEEE Network 14 (2000) 30–37.
[20] f5 network layer load balancer [Online] 2008. Available: http://www.f5.com (accessed on: 27.02.2008).
[21] D.P. Bertsekas, Nonlinear Programming, 2nd ed., Athena Scientific, Belmont, MA, 2003.
[22] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, 1st ed., Prentice Hall, Englewood Cliffs, NJ, USA, 1989.
[23] S. Ali, A.A. Maciejewski, H.J. Siegel, J.-K. Kim, Measuring the robustness of a resource allocation, IEEE Transactions on Parallel and Distributed Systems 15 (7) (2004) 630–641.
[24] L. Bölöni, D. Marinescu, Robust scheduling of metaprograms, Journal of Scheduling 5 (5) (2002) 395–412.
[25] V. Shestak, J. Smith, A.A. Maciejewski, H.J. Siegel, Stochastic robustness metric and its use for static resource allocations, Journal of Parallel and Distributed Computing 68 (8) (2008) 1157–1173.
[26] J. Smith, L.D. Briceño, A.A. Maciejewski, H.J. Siegel, Measuring the robustness of resource allocations in a stochastic dynamic environment, in: Proceedings of the 21st International Parallel and Distributed Processing Symposium, IPDPS 2007, March 2007.
[27] R. Buyya, M. Pathan, A. Vakali (Eds.), Content Delivery Networks, 1st ed., Springer-Verlag, Berlin, Germany, 2008.
[28] G. Pierre, M. van Steen, Globule: a collaborative content delivery network, IEEE Communications Magazine 44 (8) (2006) 127–133.
[29] S. Sivisubramanian, B. van Halderen, G. Pierre, Globule: a user-centric content delivery network, in: Proceedings of the 4th International Systems Administration and Network Engineering Conference, SANE 2004, 2004.
[30] R. Ranjan, A. Harwood, R. Buyya, A case for cooperative and incentive-based federation of distributed clusters, Future Generation Computer Systems 24 (4) (2008) 280–295.
[31] C.S. Yeo, R. Buyya, A taxonomy of market-based resource management systems for utility driven cluster computing, Software—Practice and Experience 36 (13) (2006) 1381–1419.
[32] M. Karaul, Y.A. Korilis, A. Orda, A market-based architecture for management of geographically dispersed, replicated web servers, Decision Support Systems 28 (1–2) (2000) 191–204.
[33] J. Smith, E.K.P. Chong, A.A. Maciejewski, H.J. Siegel, Decentralized market-based resource allocation in a heterogeneous computing system, in: Proceedings of the 22nd International Parallel and Distributed Processing Symposium, IPDPS 2008, March 2008.

Jay Smith received his Ph.D. in Electrical and Computer Engineering from Colorado State University in 2008. Jay is currently a Senior Researcher at DigitalGlobe. In addition to his position at DigitalGlobe, Jay is a research faculty member in the Electrical and Computer Engineering Department at the Colorado State University. His research interests include high performance computing and resource management. He is a member of the IEEE and the ACM.

Edwin K. P. Chong received the B.E.(Hons.) degree with First Class Honors from the University of Adelaide, South Australia, in 1987, and the M.A. and Ph.D. degrees in 1989 and 1991, respectively, both from Princeton University, where he held an IBM Fellowship. He joined the School of Electrical and Computer Engineering at Purdue University in 1991, where he was named a University Faculty Scholar in 1999, and was promoted to Professor in 2001. Since August 2001, he has been a Professor of Electrical and Computer Engineering and a Professor of Mathematics at Colorado State University. His current interests are in networks and optimization methods. He coauthored the recent best-selling book, an Introduction to Optimization, 3rd Edition, Wiley-Interscience, 2008. He was on the editorial board of the IEEE Transactions on Automatic Control, and is currently an editor for Computer Networks and the Journal of Control Science and Engineering. He is a Fellow of the IEEE, and served as an IEEE Control Systems Society Distinguished Lecturer. He received the NSF CAREER Award in 1995 and the ASEE Frederick Emmons Terman Award in 1998. He was a co-recipient of the 2004 Best Paper Award for a paper in the journal Computer Networks. He has served as Principal Investigator for numerous funded projects from DARPA and other funding agencies.

Anthony A. Maciejewski received the BSEE, MS, and Ph.D. degrees from Ohio State University in 1982, 1984, and 1987, respectively. From 1988 to 2001, he was a professor of Electrical and Computer Engineering at Purdue University, West Lafayette. He is currently a Professor and Department Head of Electrical and Computer Engineering at Colorado State University. He is a Fellow of the IEEE. His research interests include robotics and high performance computing. A complete vita is available at: http://www.engr.colostate.edu/~aam.

Howard Jay Siegel was appointed the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University in 2001, where he is also a Professor of Computer Science and Director of the university-wide Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a professor at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received two B.S. degrees (1972) from the Massachusetts Institute of Technology (MIT), and his M.A. (1974), M.S.E. (1974), and Ph.D. (1977) degrees from Princeton University. He has co-authored over 350 technical papers. His research interests include heterogeneous parallel and distributed computing, parallel algorithms, and parallel machine interconnection networks. Home page www.engr.colostate.edu/~hj.