On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers

Adaptive ***Behavior***

On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers

Sunil Nakrani
Computing Laboratory, University of Oxford

Craig Tovey
School of Industrial and Systems Engineering, Georgia Institute of Technology

Internet centers host services for e-banks, e-auctions and other clients. Hosting centers then must allocate servers among clients to maximize revenue. The limited number of servers, costs of reallocating servers, and unpredictability of requests make server allocation optimization difficult.

Based on the many similarities between server and honey bee colony forager allocation, we propose a new decentralized honey bee algorithm which dynamically allocates servers to satisfy request loads. We compare it against an omniscient optimality algorithm, a conventional greedy algorithm, and an algorithm that computes omnisciently the optimal static allocation. We evaluate performance on simulated request streams and commercial trace data.

Our algorithm performs better than static or greedy for highly variable request loads, but greedy can outperform it under low variability. Honey bee forager allocation, though suboptimal for static food sources, may possess a counterbalancing responsiveness to food source variability.

Keywords internet hosting · server allocation · biomimicry · heuristic · honey bee · foraging · self-organization · decentralized algorithm

1 Introduction

The internet landscape offers myriad services such as online stock trading, banking, ticket reservations, auctions and shopping. Internet computing infrastructure is being increasingly relied upon for day-to-day operations by a rapidly growing portion of human civilization. It is difficult to provision servers efficiently for such services because unconstrained customer behavior is unpredictable, highly variable, and can create sudden surges in request arrivals (Chase, Anderson, Thakar, & Vahdat, 2001).

An emerging market trend in internet computing is the proliferation of large hosting centers, based on common hardware platforms, that host third-party service application and content data on servers for a fee (IBM, 2003; Verio, 2003). The managed hosting center benefits from economies of scale and presents an opportunity for dynamic capacity provisioning while shielding content owners from capital overhead and unpredictable demand. Given that the hosting fee may be negotiated on the basis of requests served, the hosting center's objective is to dynamically allocate its finite servers to hosted services,

Correspondence to: Sunil Nakrani, Computing Laboratory, University of Oxford, Oxford OX1 3QD, England, UK.
E-Mail : Sunil.Nakrani@comlab.ox.ac.uk
Tel.: +44-1865-273-838, *Fax*: +44-1865-273-839

taking into account switching costs, such that revenue is maximized.

In this paper, we model the dynamic server allocation problem and propose a *biologically inspired* approach to this optimization problem in an internet hosting center. Specifically, our work has been inspired by the study of honey bee colonies and the behavior of forager bees, characterized by decentralized and elementary interactions, that effect a complex collective behavior to solve the problem of adequate food collection to ensure survival of the colony. The new *honey bee* algorithm models servers and request queues in an hosting center as foraging bees and flower patches respectively. We perform experimental work on a simulated hosting center using request trace data from a commercial service provider and from simulated request streams. Our results indicate that the *honey bee* algorithm adapts well to highly variable request streams.

The remainder of the paper is organized as follows. Section 2 describes the internet hosting center allocation problem and reviews the pertinent literature. Section 3 describes the honey bee colony forager allocation problem and its many parallels to the server allocation problem. Section 4 describes how honey bee colonies deploy forager bees to collect nectar among diverse flower patches, and Section 5 outlines our honey bee-mimicking server allocation algorithm. The greedy, omniscient and optimal-static algorithms, against which the honey bee algorithm is tested, are described in Section 6. In Section 7, we describe the simulation model and the three kinds of test data used in the experiments. Test results from the comparison of the performance of the honey bee algorithm against the other algorithms are given in Section 8. Finally, in Section 9, we discuss how our results provide some confirmatory evidence for a survival advantage conferred by the forager allocation mechanism in real honey bee colonies.

2 Internet Hosting Centers

2.1 Service Hosting

There is a trend toward the internet computation model where a single hosting provider manages content delivery of myriad internet services to requests from the global audience (IBM, 2003; Verio, 2003). This is already a multi-billion-dollar/year industry in the USA (Markoff, 2003). The prevailing architecture of choice for a hosting center is an ensemble of commodity servers which are partitioned into clusters (virtual servers) and configured to host internet services (Brewer, 2000). Incoming streams of requests for a given service are held on a service queue and spread by a load balancing switch to any server that belongs to the cluster hosting such a service. Any server in the cluster can respond to requests for service and a server from one cluster can be reallocated to another cluster by reconfiguration, thereby providing scalability and fault tolerance (Fox, Gribble, Chawathe, Brewer, & Gauthier, 1997; Chase et al., 2001). During a server's reallocation from one cluster to another, it becomes unavailable due to the time involved in scrubbing the existing service application/data and installing the new service application/data (Appleby et al., 2001). As far as customers of any service is concerned, the hosting center appears as a virtual server.

2.2 Service Level Agreements (SLAs)

Under this service hosting paradigm, the content owners are clients who purchase resources on a *pay-per-use* service level agreement (SLA). The hosting provider shares resources among its clients. The hosting center benefits from an economy of scale due to resource sharing and insulation from maintenance and over-provisioning costs for its clients. This type of outsourcing is attractive to clients for whom the internet is central to business strategy, but who do not wish to invest in the infrastructure outlay or do not have the in-house technical expertise to build and maintain such a service infrastructure.

Pay-per-request-served is a common way in which a *pay-per-use* SLA is negotiated between the hosting center and the content owner. Such an SLA implicitly acknowledges that limited servers, reallocation costs, and uncertainty may lead to overloading and lost requests (Jayram, Kimbrel, Krauthgamer, Schieber, & Sviridenko, 2001). Consequently, the prime incentive of the hosting center is to maximize the total value of requests served.

2.3 Internet Server Allocation Problem

Given multiple clients, the internet hosting center thus faces the following revenue collection problem: how to utilize its limited number of servers to collect as much revenue in SLA fees as possible over a lengthy time horizon from its clients. At each moment of the entire time horizon, it in effect has to decide which of its servers to allocate to each client. It may decide to keep the current allocation or it may decide to reallocate some servers from one client to another. If it reallocates a server from one client to another then that server suffers a downtime during which it earns no revenue. The stream of future request arrivals for each client is unknown and highly variable. The average time to serve a request and the fee paid per request differ from client to client. The request queueing behavior, in terms of how long on average a customer is willing to wait to be serviced, differs among clients as well. A revenue opportunity is lost if a customer balks, i.e., leaves the queue before being served.

2.4 Related work

There is some previous work on the internet hosting allocation problem, but most of it is proprietary (Ensim, 2004; HP, 2004; Sychron, 2004). A paper by researchers at IBM (Jayram et al., 2001) proves that no finite competitive ratio guarantee is possible for any allocation algorithm which has no knowledge of the future. The authors give a theoretical algorithm which, like our omniscient algorithm, requires knowledge of the future, and hence could not be used in practice. They also describe a related heuristic, which does not require such knowledge but apparently was not implemented or tested at the time. Their heuristic turns out to be of precisely the same type as our greedy algorithm, to the extent that can be determined from their paper. (The algorithm employs a forecasting module, the details of which are not provided in Jayram et al., 2001). Therefore, as we intended, our greedy algorithm provides a good point of comparison with conventional optimization methods.

The problem of dynamically allocating resources as unpredictable requests arrive has been studied in a general abstract context as the K-server problem (Manasse, McGeoch, & Sleator, 1988). In this formulation, we are given a metric space S and K mobile servers that reside at points in S. When a request arrives at point $x \in S$, it must be processed by moving one of the K servers to x, at a cost equal to the distance moved in S. The objective is to minimize cost in serving a sequence of requests. However, in the K-server literature, the online minimum competitive ratio is used as the criterion of algorithmic quality, which is not very meaningful in our context. This may indeed be fortunate since no deterministic algorithm can guarantee a competitive ratio better than K (Manasse et al., 1988).

Dynamic resource allocation has been well studied in the context of effectively exploiting available resources to realize certain performance goals. Several papers address the problem of sharing processor resources between competing applications to improve throughput and/or realtime response (Dusseau, Arpaci, & Culler, 1996; Banga, Druschel, & Mogul, 1999; Ford & Susarla, 1996; Jones, Rosu & Rosu, 1995, 1997; Waldspurger & Weihl, 1994, 1995). Numerous papers focus on sharing processors in distributed systems of networked machine for throughput (Baker, Buyya, & Laforenza, 2000; Baratloo, Dasgupta, & Kedem, 1995; Baratloo, Itzkovitz, Kedem, & Zhao, 1998; Buyya, Abramson, & Giddy, 2000; Czajkowski et al., 1998; Dasgupta, Kedem, & Rabin, 1995; Dusseau & Culler, 1997; Hill, Donaldson, & Lanfear, 1998; Litzkow, Tannenbaum, Basney, & Livny, 1997). However, these research efforts focus on sharing CPU cycles among competing applications as opposed to a whole server being shared in a serial manner, and moreover they do not consider reallocation downtime. A smaller number of papers have considered the problem in a context similar to the one described in this paper. Wolf and Yu (2001) propose a server allocation algorithm for the case where a single server can handle requests for multiple services and the performance objective is to minimize response time by load balancing. De Farias, King, Squillante, and Roy (2002) formulate the server allocation problem in a dynamic programming framework but propose an approximate linear programming solution given that the dynamic programming approach is computationally infeasible. It computes an allocation policy based on a particular arrival rate model. In contrast, we do not make any *a priori* assumptions regarding request arrivals to make allocation decisions. Palmer and Mitrani (2003) examine the server allocation problem and propose non-optimal heuristics that are easily implementable. However, only a simple case of hosting 2 services with total of 9 servers is considered. Chase et al. (2001) propose an economic approach to server allocation for cases where energy conservation is the primary goal. Finally, Appleby et al. (2001) propose an SLA based approach where monitoring agents provide load and performance feedback to a central resource director which makes allocation decisions. In our approach, feedback is provided by all servers in the hosting center but individual servers are responsible for allocation decisions.

3 Honey Bee (*Apis mellifera*) Colonies

The honey bee (*Apis mellifera*), indigenous to Europe, Africa, and the Middle-East, has been introduced to other parts of the world by beekeepers. Colonies of honey bees have been studied extensively for many years and a significant volume of literature has been devoted to understanding their social organization (Seeley, 1995 and references therein). The social organization is adapted to colony survival and propagation; particularly in colder regions (Northern Europe and North America), a colony must accrue adequate honey to sustain itself through the winter. It must, therefore, coordinate its foraging activities in order to efficiently collect food during summer months when the surrounding countryside is replete with flora while facing competition from other colonies as well as other foraging insects.

Typically, the honey bee colony is comprised of approximately 20–50 thousand bees with one queen, a few drones, and the rest workers. Depending on age, a worker bee specializes in performing different tasks that range among cleaning, maintenance, guarding, temperature regulation, and foraging for food (nectar, pollen, water). During infancy, worker bees serve the queen and nurse the young while adulthood is spent in foraging endeavors. Approximately 25% of worker bees are engaged in foraging to satisfy an annual requirement of up to 120 kg of nectar, 20 kg of pollen and 20 liters of water. Nectar collection is a principal foraging activity. The bee colony requires approximately 50 kg of nectar as a reserve for winter survival—without it the queen freezes and the colony dies. During the summer, the colony consumes approximately 70 kg of nectar. A significant portion of this nectar goes to meet the energy demands of the foragers which exploit at least 100 km^2 of the surrounding countryside.

3.1 Forager Allocation Problem

In colder climate zones such as much of North America, blossoming of flora that yields nectar is regulated by seasonal and daily variations. At the seasonal level, the flora is scarce in winter and abundant in summer. On a daily level, the availability and quality of nectar vary directly with micro-climatic conditions, the blooming/withering cycle, and exploitation. Consequently, within the space of a few days, conditions may give rise to a dearth or an abundance of nectar supply. Effectively, nectar is available during about 10 weeks of summer each year. During this period, not only is the availability of nectar unpredictable but also the scale of fluctuation from dearth to abundance is unknown. It is not uncommon for a colony to experience *abundance-to-dearth* nectar in-take rate fluctuation by a factor of more than 100:1 within a day or so (Seeley, 1995).

A honey bee colony has a limited number of nectar foragers which are distributed among the available flower patches in the surrounding countryside. A forager can be reallocated from one patch to another but requires time to learn the location of a new flower patch. Given the volatility of flower patches and uncertainty of nectar supply, the forager allocation problem is to dynamically allocate foragers among flower patches such that nectar intake is maximized.

Consider a honey bee colony with N nectar foragers, and forage sites indexed by i. Let $f_i(x_i)$ denote the value returned from the ith forage site if the number of foragers collecting nectar from there equals x_i. If the functions $f_i(\ )$ did not change over time, the resulting *static* forager allocation problem would be: max $\sum_i f_i(x_i)$ subject to $x_i \geq 0$ and $\sum_i x_i \leq N$. Applying the Karush–Kuhn–Tucker conditions immediately shows that the optimal static solution has the form

$$f_i'(x_i) = \lambda\ \forall x_i > 0;\ f_i'(0) \leq \lambda\ \forall x_i = 0.$$

That is, the marginal contributions are equal at each active patch. This is intuitively obvious because if the marginal contribution $f_i'(x_i)$ at patch i exceeds the marginal contribution at another active patch j, more nectar could be obtained by shifting a small amount of forager resource from j to i.

3.2 Server and Forager Allocation Problem Parallels

At the outset of our research, it was immediately obvious that the server and forager allocation problems were similar, at least on a superficial level. The allocation of servers to collect revenue in internet hosting centers parallels the allocation of foragers to collect nectar in honey bee colonies; the ensemble of servers is analo-

Table 1 Parallels between server and forager allocation problems.

Internet hosting center	Honey bee colonies
An ensemble of *N* servers	An ensemble of *N* nectar foragers
Unit of allocation: Single server	Unit of allocation: Single nectar forager
Host *M* services (web-application) which are accessed by requests	*M* distinct flower patches in the surrounding countryside.
A group of servers with the same service application serving requests from a specific service queue	A group of nectar foragers collecting nectar at a specific flower patch
Time to service a request will be dependent on the service application	Time to travel will be dependent on the location of the flower patch
Time taken by a server to find a request to serve at a service application	Time taken by a forager to collect nectar at a flower patch
Server switching to another service incurs downtime due to purging and installation of service application/data	Forager switching to another flower patch incurs switching cost due to time spent learning the location and successful discovery
A service application and its requests have a *value-per-request*	A flower patch has quality of nectar (sugar concentration)
Variable rates of requests arrival and balking behavior	Variable rates of flower patch quality, density and replenishment

gous to a colony of forager bees; the service queues relating to hosted internet services are analogous to sources of nectar to be exploited profitably. Our examination at a finer level of detail then revealed a remarkably close mapping between the two problems. This strong similarity motivated our biomimicry approach.

A hosting center with a certain number of servers hosting multiple internet clients is analogous to a honey bee colony with a certain number of bees foraging at multiple sites in the surrounding countryside. The amount of time needed by a server to earn revenue from a client depends partly on client characteristics, but also degrades as more servers are allocated to that client. Similarly, the amount of time needed by a forager to return with a load of nectar depends partly on the site's characteristics, but also degrades as more foragers are allocated to that site. This and other similarities are detailed below and in Table 1.

The service request queues which buffer arriving requests are analogous to available flower patches (forage sites). The request stream behavior and flower patch volatility are similar in their variability. Request streams are variable due to fluctuating arrivals and balking behavior; flower patches are variable due to fluctuations in micro-climate, quality, density and nectar replenishment. In a hosting center, a virtual server is composed of one or more servers collecting revenue from its service queue. The unit of augmentation or reduction is a single server. In a honey bee colony this is equivalent to the set of forager bees collecting nectar from a specific flower patch. The unit of augmentation or reduction is a single forager bee. A server in a hosting center collects some *value-per-request-served* while a forager bee in a honey bee colony collects some mol/L sugar concentration from the nectar.

The time to service a request in a hosting center depends on the service application of the internet service and the time to find a request to serve will be a function of the number of servers allocated to the service in question and the request arrival pattern. In a honey bee colony, the travel time to a specific forage site depends on its location in the countryside, and the time needed to collect nectar will be a function of the number of foragers allocated to the forage site in question and the rate of nectar replenishment of the flowers at that site. Different internet services have different values of *service-time-per-request* and *value-per-request-served,* just as different forage sites have different values *forager-round-trip-time* and *nectar quality.*

Finally, when a server is reallocated from one internet service to another, it becomes unavailable for a fixed time duration while it undergoes a process of reassignment. Similarly, when a forager bee is reallocated to a new forage site, it becomes unavailable due to time spent learning the location of the new forage site (Seeley, Camazine, & Sneyd, 1991).

4 Honey Bee Solution: Self-Organizing Forager Allocation

Colonies of social insects (bees, ants, wasps, termites) possess what has been classed as *Swarm Intelligence*. A broad definition of the term implies a sophisticated collective behavior borne out of primitive interactions among members of the group to solve problems beyond the capability of individual members. Such colonies are characterized by (i) self-organization: Decentralized and unsupervised coordination of activities, (ii) adaptiveness: Response to dynamically varying environments, and (iii) robustness: Accomplishing the group's objective even if some members of the group are unsuccessful. It has been suggested that these group level adaptive properties lend themselves well to distributed optimization problems in telecommunications, manufacturing, transportation and so on (Bonabeau, Dorigo, & Theraulaz, 1999; Bonabeau & Meyer, 2001; Cicirello & Smith, 2001).

A model of self-organization that takes place within a colony of honey bees has been presented by Seeley (1995). The model describes interactions between members of the colony and the environment that leads to dynamic distribution of foragers to efficiently collect nectar from an array of flower patches (food sources) that are capricious in terms of *profitability* to the colony. Foraging bees visiting flower patches return to the hive with nectar and with a profitability rating of respective flower patches. At the hive, forager bees interact with receiver bees to offload collected nectar. This interaction provides feedback on the current status of nectar flow into the hive. This feedback mechanism sets a response threshold for an enlisting signal. An amalgamation of response threshold and profitability rating (function of nectar quality, nectar bounty and distance from the hive) influences the length of the enlisting signal known as the *waggle dance*.

The waggle dance is performed on the dance floor where inactive foragers can observe and follow. Effectively, each active forager bee provides feedback on her local flower patch while observing bees have access to the set of attractive food sources being capitalized by the colony. However, individual foragers do not acquire the full set of global knowledge but, rather, randomly select a dance to observe from which they can learn the location of the flower patch and leave the hive to forage.

The resulting self-organized proportionate allocation pattern, derived from multiple and proportionate feedback on *goodness* of food sources, is described by Seeley et al. (1991) and validated by experimental study on a real honey bee colony. The model given by Bartholdi, Seeley, Tovey, and Vande Vate (1993) predicts a steady-state pattern of forager allocation where rate of value accumulation equalizes among forage sites being exploited, i.e., self-organizing allocation results in equal *average* nectar return $\frac{f_i(x_i)}{x_i} = \mu \forall i$ where $f_i(x_i)$ denotes the value returned from x_i foragers collecting nectar at the *i*th forage site.

Intriguingly, this allocation pattern is *not* optimal (unlike the famed hexagonal comb structure), although Bartholdi et al. (1993) prove that no allocation is more than twice as efficient. Why would a sub-optimal foraging allocation mechanism evolve in honey bee colonies? We suggest that the static sub-optimality is related to the dynamic nature of the foraging environment. Flower patch availability and quality may wax and wane rapidly even during a single day. We believe that a real honey bee colony is quite adaptive to changes in flower patch availability and quality, and that its allocation method is considerably less sub-optimal when evaluated in a realistic dynamic environment rather than a static one. This performance quality was a key motivation for our biomimicry algorithm.

One possible explanation for a performance difference is that, as noted previously, static optimization requires equalization of derivatives (marginal rates). That is, static optimization requires $f_i'(X_i) = \lambda$ for all active patches *i*. This would seem to require a *marginal rate bee* for each patch, a bee who acquires nectar value at rate $f_i(X_i) - f_i(X_i - 1)$, thus limiting migration rates to one bee at a time. If this explanation is correct, the honey bee colony trades off static optimality for adaptiveness. It has no marginal rate bee, but it has the abi-

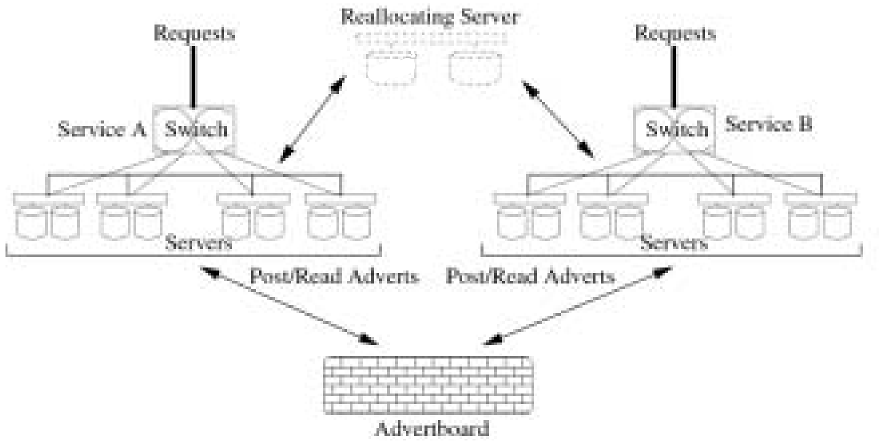

Figure 1 An Internet hosting center with two services hosted.

Table 2 Mimicking key features of forager allocation.

Forager allocation	Server allocation
Waggle dance	Advertisement
Dance floor	Advertboard
Waggle dance duration	Advertisement duration
Flower patch location	Web-site identifier
Following a waggle dance	Reading an advertisement
Waggle dancing	Posting an advertisement

lity to migrate many bees at once, increasing its responsiveness to changed circumstances.

5 Honey Bee Server Allocation Algorithm

When we began work on the server allocation problem, we selected the honey bee forager algorithm for inspiration because (besides the obvious superficial resemblances) the rapidly changing request streams, the significant downtime cost of reallocation, and the distributed nature of the processors all matched to performance strengths of honey bee foraging. The general idea is depicted in Figure 1: At any time each processor is devoted to servicing one class of requests (just as each bee is devoted to foraging at one flower patch); after a processor completes a transaction, it places an advertisement on an advertboard with some probability (just as returning foragers perform a waggle dance with some probability); processors from time to time randomly read from the advertboard and are thus recruited to service a different request class (just as returned foragers may be recruited to different patches).

Let any server in a hosting center be either a forager or a scout server. As outlined in Table 2, let the *dance floor* be represented by an advertboard, a *waggle dance* be represented by an advertisement and its *duration* by the length of time an advertisement posting appears on the advertboard. Furthermore, let a *flower patch location* be represented by a service identifier while *waggle dancing* and *following a waggle dance* are represented by advertisement posting and advertisement reading respectively. Consider an internet hosting center with N servers partitioned into M groups called virtual servers $V_0 \dots V_{M-1}$. There are M service request queues $Q_0 \dots Q_{M-1}$ which buffer the stream of customer requests to be served by respective virtual servers. Each virtual server V_i is allocated n_i servers, $n_i \geq 0$, $\sum_{i=0}^{M-1} n_i = N$, and the unit of allocation

```
[A] Initialization– $s_i \in V_j$ serving $Q_j$, Advert posting probability $p$,
     Advert reading probability $r_i$ , Revenue rate interval $T_{pr}$ ,
     Advert reading interval $T_r$
[B] forever
[C]    while $Q_j \neq$ Empty do
           serve request;
           if $T_{pr}$ expired then
               compute revenue rate;
               adjust $r_i$ from lookup table;
           if Flip(p)==TRUE then Post Advert;
           if $T_r$ expired && Read($r_i$)==TRUE then
               /* randomly select an advert or a virtual server */
               Select advert( if forager) or Virtual Server $V_k$ (if scout);
               Read advert id $V_j$ ( if forager);
               if $V \neq V_j$ then Switch($V_k$ );
       endwhile
     endforever
```

Figure 2 Server behavior in the honey bee allocation algorithm.

is a single server. Let the cost of server reallocation, i.e., length of time the server is unavailable as it undergoes repurposing, be C time units. Assume that a server $s_i \in V_j$ serving requests from service request queue Q_j is paid v_j, the *value-per-request-served*. Figure 1 illustrates the salient features of the honey bee algorithm in a hosting center with two services hosted.

The behavior of any server in the hosting center is controlled by the pseudo-code of Figure 2. A server $s_i \in V_j$, on completion of each request from Q_j, will attempt with probability p to post an advert on the advertboard with duration $D = c_j A$, where A denotes the advert scaling factor. Also, it will attempt with probability r_i to read a randomly selected advert from the advertboard if it is a forager or randomly select a V_j, $j : 0 \ldots (M - 1)$ if it is a scout. The probability r_i is dynamic and changes as a function of the forager/scout server's own revenue rate and hosting center's overall revenue rate. The revenue rate, P_i, for a server s_i is given by $P_i = \frac{c_j R_i}{T_i}$ where R_i is the total number of requests served by a given server in the time interval T_i. The hosting center's overall revenue rate, P_{hosting}, is given by $P_{\text{hosting}} = \frac{1}{T_{\text{hosting}}} \sum_{j=0}^{(M-1)} c_j R_j$ where R_j denotes the total number of requests served by V_j in the time interval T_{hosting}. A server $s_i \in V_j$ serving queue Q_j determines its *profitability* by comparing the revenue rate P_i with P_{hosting}. It updates r_i according to the rules shown in Table 3. These rules quantitatively model forager behavior in real honey bee colonies given that *waggle dancing* foragers never get recruited to another patch and foragers use global information (hive's nectar intake rate cued by time to find a receiver bee) to decide whether to dance or not. Thus, bees at a profitable patch have a decreased chance of following another *waggle dance*.

Table 3 Rules for adjusting probability of reading the advertboard.

Revenue rate	P[read] r_i
$P_i \leq 0.5P_{\text{hosting}}$	0.60
$0.5P_{\text{hosting}} < P_i \leq 0.85P_{\text{hosting}}$	0.20
$0.85P_{\text{hosting}} < P_i \leq 1.15P_{\text{hosting}}$	0.02
$1.15P_{\text{hosting}} < P_i$	0.00

6 Implementation and Testing: Server Allocation Algorithms

We implemented the honey bee algorithm in a simulated environment and tested it against three other algorithms on several different arrival patterns. This section describes the three other algorithms; the next section describes the simulation environment and the request arrival test data.

We now detail three alternative algorithms to allocate servers to competing hosted services. The basic challenge is to determine server demand of each service based on its current request arrivals. We assume there are M groups comprised from n servers, called virtual servers $V_0 \dots V_{M-1}$, and service queues $Q_0 \dots Q_{M-1}$. A server $s_i \in V_j$ serving queue Q_j is paid c_j cents for each request served.

6.1 Omniscient

The omniscient algorithm provides an upper bound on possible profitability. (It would be both informationally impossible and computationally prohibitive in practice.) It computes, by dynamic programming, the optimal server allocation given complete knowledge of future request arrivals. Let the time horizon be divided into T time steps indexed by $t = 1, \dots, T$. Also, let S_t denote the state of the system at the start of time step t, including the allocation of servers among hosted services, and residual requests from time step $t - 1$. Let A_t denote the arrivals during time step t. We let π denote an allocation decision for a time step. Let $P(\pi, S, A)$ denote the revenue earned by π during a time step with initial state S and arrival A. Similarly, let $f(\pi, S, A)$ denote the state of the system which would eventuate at the start of the next time step, if the current time step starts in state S, allocation π is used in the current step, and the arrivals is A. Define $v^{T+1}(S_{T+1}) = 0$, i.e., there is no state-dependent salvage value at the end of the time horizon. Therefore, the value function of the omniscient policy for time step t will be

$$v^t(S_t) = \max_\pi \{P(\pi, S_t, A_t) + v^{t+1}(f(\pi, S_t, A_t))\}$$

and the corresponding optimal policy for time step t will be given by

$$\pi_*^t(S_t) = \arg\max_\pi \{P(\pi, S_t, A_t) + v^{t+1}(f(\pi, S_t, A_t))\}.$$

The recursive nature of the value function calls for dynamic programming to solve for the optimal allocation policy decisions over the complete time horizon. It would be remiss of us not to point out that this algorithm is greatly time and space intensive as a function of a problem size. For example, to allocate 50 servers across 3 virtual servers, using 11 interpolation buckets, requires at least 1.3 GB for the interim results table and exploration of 167 million states per time step.

6.2 Greedy

The greedy allocation algorithm represents a conventional heuristic approach to the problem. At time step t, if we were omniscient, we would choose an allocation π^t that maximized the sum of revenue during the time step, $P(\pi^t, S_t, A_t)$, plus the future revenue that could be earned if we began time step $t + 1$ in state $f(\pi^t, S_t, A_t)$. The greedy algorithm ignores the second term of this sum. Part of the rationale for being myopic is that we do not know the future, and hence cannot evaluate the second term. Also, as we have seen, such evaluations are both time and memory intensive.

To maximize $P(\pi^t, S_t, A_t)$ exactly also would require knowledge of the future. So the greedy algorithm requires a forecast $\tilde{A}_t$ of A_t, and it chooses

$$\pi_G^t(S_t) = \arg\max_\pi \{P(\pi, S_t, \tilde{A}_t)\}.$$

The state is then updated according to

$$S_{t+1} = f(\pi_G^t, S_t, A_t).$$

As stated in the introduction, the heuristic proposed independently in Jayram et al. (2001) is identical to greedy. This confirms that greedy represents a standard heuristic approach to the problem.

Jayram et al. (2001) do not report an implementation of greedy, nor do they even specify a forecasting method. (Such details may have been available but considered proprietary.) In our implementation of greedy, we used the natural forecast $\tilde{A}_t = A_{t-1}$. The idea is that the immediate past is the best available guide to the uncertain future. Thus, the algorithm chooses $\pi_G^t = \arg\max_\pi \{P(\pi, S_t, A_{t-1})\}$.

6.3 Optimal-Static

The optimal-static algorithm omnisciently chooses the best from among all *static* (fixed) allocations. This reflects an upper bound on the current level of revenue of many hosting centers, which do not change their allocations more often than once a month. Their revenue will be lower than optimal-static revenue because it is impossible for them to know next month's request arrivals in advance. The best static allocation policy for a time horizon split into T time steps is defined as follows:

```
[A] Initialization– Advertboard, Advert expiry time $T_{exp}$,
     Current time $T_{cur}$
[B] forever
[C]    while Advertboard $\neq$ Empty do
          find advert;
          if advert.$T_{exp} \leq T_{cur}$ then
               expunge advert;
       endwhile
    endforever
```

Figure 3 Advertboard manager for the honey bee allocation algorithm.

$$\pi_{\text{static}} = \arg\max_{\pi} \sum_{t=1}^{T} P(\pi, S_t, A_t)$$

subject to

$$S_{t+1} = f(\pi, S_t, A_t).$$

Thus, the same allocation policy π_{static} is used in every time step and the total revenue will be given by

$$\sum_{t=1}^{T} P(\pi_{\text{static}}, S_t, A_t).$$

7 Implementation and Testing: Simulation Model and Arrival Data

In this section we describe the simulation model of the dynamic server allocation problem. We have developed discrete event simulation models for the four server allocation algorithms—honey bee, omniscient, greedy, and optimal-static. All algorithm simulation models are implemented in C++SIM (Little, 1994) on an IBM XSeries with Linux operating system. The following assumptions are common to all models. All servers are homogeneous in terms of processing capacity and employ a *first-come-first-served* scheduling policy. The time to serve a request is exponentially distributed with a mean service time depending on request type. Each server is paid a fixed revenue per request served, the amount depending again on request type. A reallocated server becomes unavailable for a migration downtime (Appleby et al., 2001). This migration cost is incurred because a server must be purged of its current application and data, then reloaded with the new application and data of the internet service to which it is being reallocated. A stream of requests arriving for a particular virtual server is held in a service queue. Each request has a waiting threshold to receive service and on crossing this threshold, a request randomly chooses to keep waiting or balk. Each virtual server has an independent request stream.

In the honey bee model, servers can be reallocated at any time. One server per virtual server is designated as a *scout*. The rest are designated as *foragers*, reflecting the low proportion of scouts in real honey bee colonies (Seeley, 1995). A *scout* server randomly reallocates itself to any virtual server in the hosting center at any time while *forager* servers randomly reallocate themselves in response to an advert read for a particular virtual server. Each server (*scout/forager*) may advertise its own virtual server by placing an advert on the advertboard with given time duration. As depicted in the pseudo-code of Figure 3, the advertboard is kept up to date by purging adverts with expired time duration.

In the omniscient and greedy models, servers are reallocated at the beginning of each allocation interval. For the optimal-static model, servers allocated at the beginning remain unchanged for the complete duration of the simulation and, therefore, do not suffer any reallocation downtime costs. All simulation models except honey bee and greedy require an allocation policy which is computed offline using the request arrival data.

The parameters and values used for the simulation models are depicted in Table 4. In general, we chose environment parameter values based on data from studies or observation. However, we had to limit the total number of servers so that time/space constraints could be satisfied when computing allocation policies for the omnisicient algorithm, and to manage the memory footprint when running simulations. The server reallocation downtime results from scrubbing existing and

Table 4 Simulation parameters.

Parameter	Value
Common:	
Total number of Servers	50
Server reallocation downtime	300 seconds
Revenue per request served	0.5 cent
Exponentially dist. mean service time	15 milliseconds
Request waiting threshold	10 seconds
Balk rate	1.01 seconds
Balk probability	0.04
Honey Bee:	
Advert posting probability	0.10
Advert reading probability (initial)	0.10
Scouts per hosted service	1
Revenue rate computation interval	5 seconds
Omniscient/Greedy:	
Server allocation interval	1800 seconds

installing new application/data. Appleby et al. (2001) give 270–330 seconds downtime for servers and network used in their hosting center setup. The request waiting threshold is determined by the behavior of customers when experiencing delay between a *mouse-click* and receiving a response. Nielsen (2000) suggests that delays greater than 10 seconds lead to customers becoming distracted and likely to *click-away* (balk). The parameter *balk rate* represents how often customers, having waited more than 10 seconds for service, think about balking, and the parameter *balk probability* represents the likelihood of balking on each occasion. Given that a balking rate of 1.01 seconds is chosen for each waiting customer, a low balking chance of 4% is chosen based on appropriate scaling. The revenue per request served and the mean service times are arbitrarily chosen. The exponential distribution for the service time is conventional.

For the honey bee algorithm, we depended again on data from studies, together with common-sense scaling reasoning, to determine the parameter values. The advert posting probability was chosen to be 0.1 since approximately 10% of returning foragers in the real bee colony perform waggle dances (Seeley, 1995). In effect, servers that are busy will post adverts more frequently. Each server decides to read the advertboard upon expiration of the advert reading interval and has a chance of succeeding equal to *reading* probability. In order to limit unnecessary server reallocation downtime, the following is observed:

$$\frac{\text{Advert reading interval}}{\text{Advert reading probability}} \gg \text{Server reallocation downtime.}$$

Therefore, a low reading probability of 0.1 was chosen. The revenue rate computation interval determines the frequency with which servers receive new feedback on revenue prospects. This interval is chosen to be 5 seconds so that feedback on short bursts of request arrivals is minimized bearing in mind the 15 milliseconds mean service time. We did not alter the honey bee algorithm's parameter values after starting computational experiments (as is often done to improve a heu-

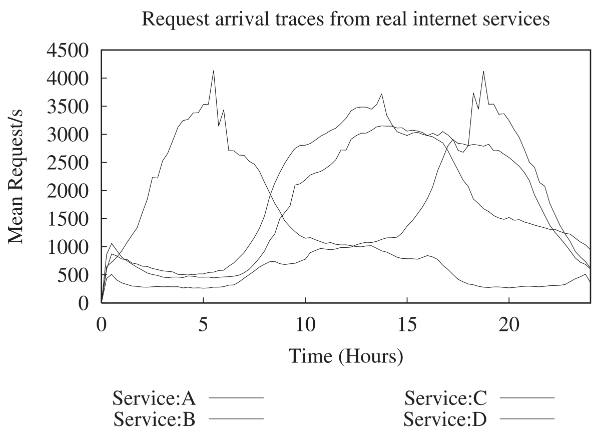

Figure 4 Real internet service request streams.

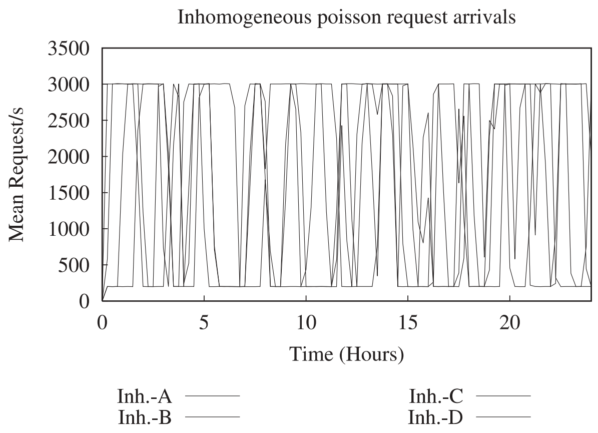

Figure 5 Synthetically generated inhomogeneous Poisson request streams.

ristic's performance). All of the results we report here are for the original untuned parameter values.

We use trace data from a commercial service provider and synthetically generated request streams to evaluate algorithm performance. The trace data, depicted in Figure 4, are from an international commercial service provider (a confidentiality agreement restricts us from disclosing its name). The simulated data are derived in two ways—drawn from inhomogeneous Poisson processes, and drawn from a heavy tail distribution as depicted in Figures 5 and 6. The inhomogeneous Poisson model was developed by Brown et al. (2002) as a good fit to real data; it is controlled by a variability parameter which is the ratio between the maximum and minimum intensities of the arrival process. Realistic values of the variability parameter range from approximately 10 to 100 or more (Chase et al., 2001; Arlitt & Jin, 1999; Crovella, 1998). The heavy tail distribution characteristics have been observed through statistical analysis of heavily accessed internet services such as the 1998 Winter Olympics held in Nagano Japan (Gelenbe, 2000). We use the Cauchy distribution, which has undefined mean and standard deviation, to generate heavy tail request streams.

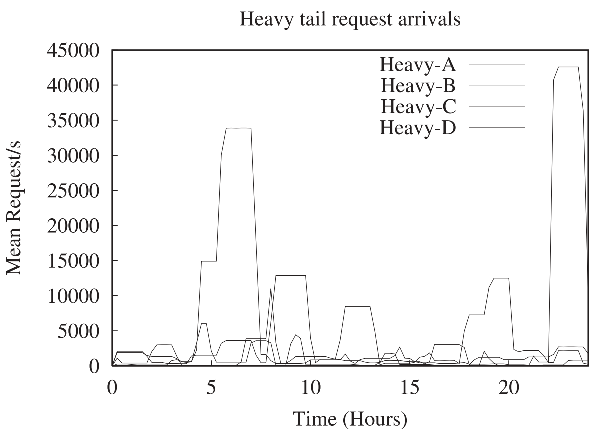

Figure 6 Synthetically generated heavytail request streams.

8 Experimental Results

In this section, experimental results are presented comparing the performance of honey bee algorithm with omniscient, greedy, and optimal-static algorithms using synthetic request arrival data as well as trace data from a commercial service provider. These experiments consider the cases of a hosting center with 2, 3, and 4 virtual servers (internet services) composed from a total of 50 servers. We use the performance metric of total revenue earned.

8.1 Experiments

The request arrival trace data from a commercial internet service provider as shown in Figure 4 were used to run simulations of a hosting center configured to host 2, 3, and 4 internet services. The request arrival traces labeled *service-B/C* were used for hosting 2 services, *service-A/B/C* were used for 3 services hosted and all traces were used for 4 services hosted. The revenue earned by the hosting center over a 24 hour period using the honey bee, omniscient, greedy and optimal static allocation algorithms for each service hosting configuration is depicted in Table 5 and normalized performance is depicted in Figure 7. As expected,

Table 5 Revenue from real internet service traces.

	Revenue($)		
Algorithm	2 services	3 services	4 services
Omniscient	1,071,741.83	1,352,872.12	1,415,403.63
Honey bee	1,050,110.00	1,238,470.00	1,292,460.00
Greedy	1,043,400.00	818,040.00	1,170,620.00
Optimal-static	844,822.00	1,108,360.00	1,242,390.00

Table 6 Revenue from synthetically generated request streams.

		Mean revenue($)			
#	Req.	Honey	Greedy	Optimal-static	Omniscient
2	4:1	728,141.60	781,654.20	785,380.20	785,382.63
4	4:1	786,873.00	832,072.25	860,978.75	860,985.69
2	10:1	945,781.35	852,730.00	841,992.15	1,179,710.86
4	10:1	992,512.95	920,372.50	900,523.25	1,311,975.69
2	Heavy	972,323.30	951,800.05	921,498.70	1,048,839.73
4	Heavy	1,004,615.00	926,127.50	965,757.25	1,240,434.94

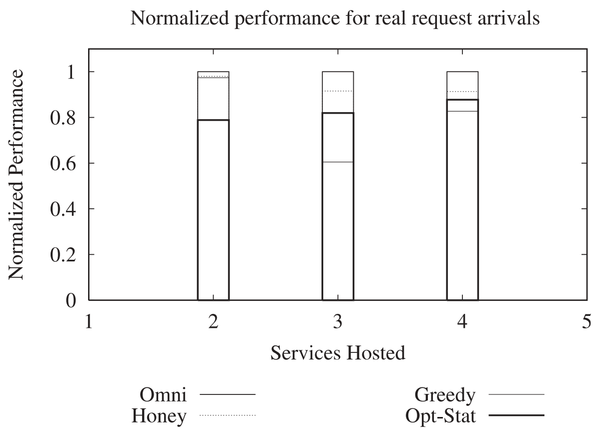

Figure 7 Revenue normalized with respect to omniscient algorithm for real internet service traces.

the omniscient algorithm outperforms all three algorithms. The honey bee algorithm performs the best of the other three algorithms, earning revenue within 2% (2 services), 9.2% (3 services) and 9.5% (4 services) of the omniscient algorithm's. The honey bee algorithm outperforms the greedy algorithm by 0.6% (2 services), 33.9% (3 services), and 9.5% (4 services); it outperforms optimal-static by 19.5% (2 services), 10.5% (3 services), and 3.9% (4 services). It is evident from the experimental results that the honey bee algorithm adapts well to the highly varying request arrival patterns of real customers of internet services.

The synthetically generated request streams (inhomogeneous Poisson and heavy tail) were used to run simulations with the hosting center configured to host 2 and 4 internet services. Examples of these request streams are depicted in Figures 5 and 6. The mean revenue earned by the hosting center over a 24 hour period using all four algorithms is depicted in Table 6 and normalized performance is depicted in Figures 8–10. The revenues reported are the mean revenue based on 20 simulation runs for each combination of configuration and request stream type. For each simulation run, a different initial seed value was used in the distribution function utilized for generating respective request streams. In the case of low variability inhomogeneous Poisson request streams with a *peak-to-trough* ratio of 4:1, the honey bee algorithm is outperformed by the greedy, optimal static and, as expected, omniscient algorithms

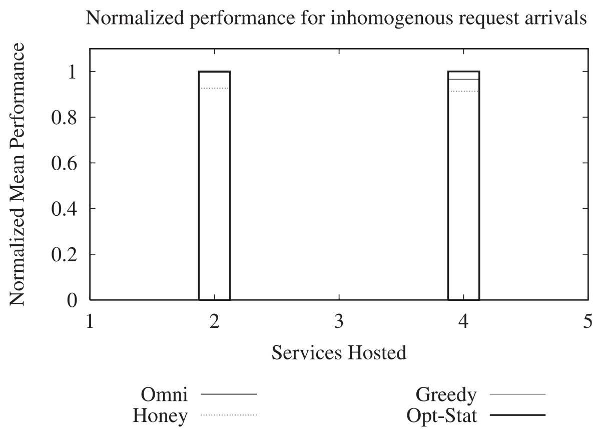

Figure 8 Revenue normalized with respect to omniscient algorithm for 4:1 inhomogeneous Poisson request stream.

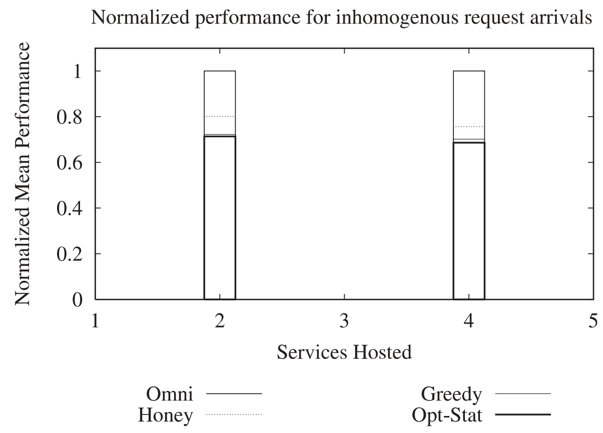

Figure 9 Revenue normalized with respect to omniscient algorithm for 10:1 inhomogeneous Poisson request stream.

for both 2 and 4 services hosted. The greedy algorithm outperforms honey bee algorithm by 7.3% (2 services) and 5.7% (4 services). The optimal-static and omniscient algorithms outperform honey bee algorithm by approximately 7.9% (2 services) and 9.4% (4 services). In the case of higher variability inhomogeneous Poisson request streams with a 10:1 *peak-to-trough* ratio, the honey bee algorithm is, of course, outperformed by omniscient algorithm but it performs better than greedy or optimal-static algorithm. It performs within 19.8% (2 services) and 24.3% (4 services) of the omniscient algorithm. It outperforms the greedy algorithm by 10.9% (2 services) and 7.8% (4 services); it outperforms optimal-static by 12.3% (2 services) and 10.2% (4 services). In the case of heavy tail request streams, the performance of the honey bee algorithm is similarly good. It performs within 7.3% (2 services) and 19.0% (4 services) of the omniscient algorithm. It outperforms the greedy algorithm by 2.2% (2 services) and 8.5% (4 services) while, against optimal-static, it outperforms by 5.5% (2 services) and 4.0% (4 services).

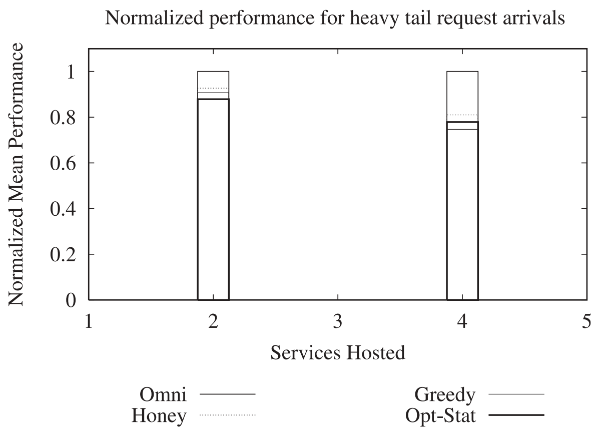

Figure 10 Revenue normalized with respect to omniscient algorithm for heavy tail request stream.

The main qualitative observation we draw from the experimental results is that the honey bee algorithm performs very well, except in situations of low variability. We emphasize that all these results are for an *untuned* honey bee algorithm. That is, we chose the parameter values for the honey bee algorithm based on biological data or common-sense scaling reasoning, as described in previous sections, and froze those values before we ran any test cases. Despite the stringency imposed by not tuning, our algorithm adapts well to dynamic changes in the arrival patterns. Tuning these parameters using two-level fractional factorial experiments resulted in a performance difference of only 5.5% between the best and worst parameter tunings, indicating that the honey bee algorithm is robust across a range of parameter settings (Nakrani & Tovey, 2004). We also tested all four algorithms with inhomogeneous Poisson request streams for other degrees of inhomogeneity, up to a 30:1 ratio, for the hosting center configured to host 2, 3, and 4 internet services and observed the same qualitative behavior. This is illustrated in Figures 11–13 where we use a numerical scale to denote degree of inhomogeneity (0 = 1:1, 30 = 30:1). We also performed a test at low variability and a low customer demand rate (half the usual number of customers per server). In this scenario, the honey bee algorithm performs as well as the other methods, because virtual server capability never comes close to being saturated. Hence there is enough excess capacity to absorb

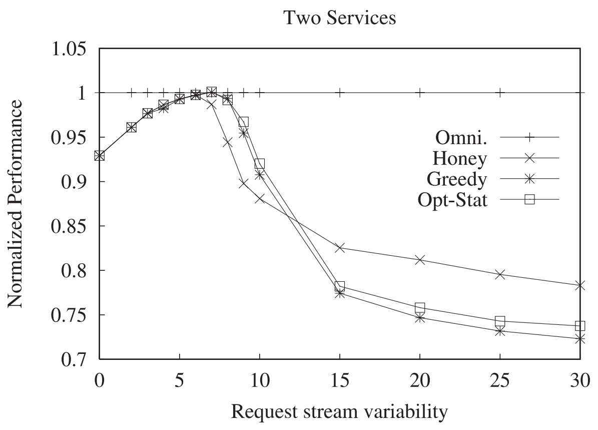

Figure 11 Adapting to variability in inhomogeneous Poisson request arrivals with hosting center configured to host 2 internet services.

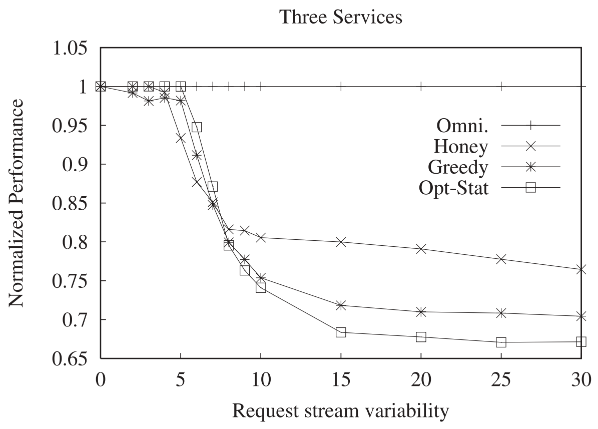

Figure 12 Adapting to variability in inhomogeneous Poisson request arrivals with hosting center configured to host 3 internet services.

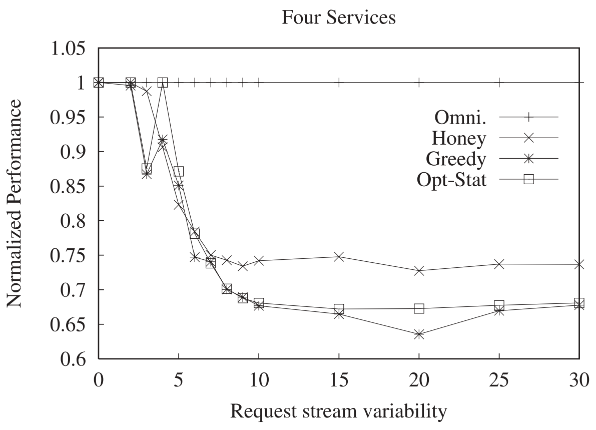

Figure 13 Adapting to variability in inhomogeneous Poisson request arrivals with hosting center configured to host 4 internet services.

the overhead of the honey bee algorithm, i.e., the downtime resulting from steady-state random switching.

To test for statistical significance of the results, we use the standard method for paired comparisons. This method is appropriate because within each single experiment, the same random seed was used for each algorithm. The sample means and standard errors of the differences are given in Table 7. Every comparison supports the qualitative conclusions above with confidence better than 99%.

9 Conclusion

In this paper, we proposed a new *honey bee* allocation algorithm based on self-organized behavior of foragers in honey bee colonies, and many similarities between the nectar collection problem faced by a honey bee colony and the revenue collection problem faced by an internet server colony. Results to date support the effectiveness of the algorithm, particularly in the highly dynamic and unpredictable (Arlitt & Jin, 1999) internet environment. The sub-optimality of the pattern of forager allocation in honey bee colonies, with respect to *unchanging* flower patches, was mimicked by the sub-optimality of the honey bee algorithm compared with the static algorithm, for test cases with low variability.

Earlier in this paper, we have suggested that honey bee colonies may have evolved to be responsive to changes in flower patch availability and quality, and that their allocation patterns may be considerably less sub-optimal when evaluated in a realistic dynamic environment rather than a static one. We believe it would be impossible to design a convincing experimental test of this idea, even if we had impossibly perfect knowledge of present honey bee behavior, because there is no satisfactory way to generate alternative evolutionary paths. We could hypothesize, for example, a honey bee colony with a linear dominance hierarchy among foragers, so that the lowest ranking bee at a patch retrieved nectar at the marginal rate, but it would be absurd to try to justify any particular instantiation of such a colony.

But the computational results presented here do provide some confirmatory evidence of adaptiveness. The honey bee heuristic compares better with other algorithms as the environment becomes more dynamic. This would be consistent with the honey bee colony's adaptation to dynamic environments.

Table 7 Revenue difference from synthetically generated request stream.

#	Req.	Revenue ($)					
		Hon.–greedy		Hon.–opt.		Hon.–omni.	
		$\bar{d}$	$\frac{\bar{\sigma}}{\sqrt{n}}$	$\bar{d}$	$\frac{\bar{\sigma}}{\sqrt{n}}$	$\bar{d}$	$\frac{\bar{\sigma}}{\sqrt{n}}$
2	4:1	–53,512.60	1825.23	–57,238.60	1906.43	–57,241.03	1906.82
4	4:1	–45,199.25	1903.43	–74,105.75	1553.78	–74,112.69	1554.70
2	10:1	93,051.35	9055.47	103,789.20	8843.29	–233,929.51	6718.32
4	10:1	72,140.25	10,346.94	91,989.50	8600.71	–319,462.94	7154.13
2	Heavy	20,523.25	7,080.90	50,824.60	9912.45	–76,516.43	45,652.41
4	Heavy	78,487.50	60,630.87	38,857.75	17,382.23	–235,819.94	13,012.29

Acknowledgements

We would like to express our gratitude to Professor Tom Seeley for helpful discussions and insights on the inner workings of honey bee colonies. We would also like to thank Professor Richard Brent for making it possible to run simulations at the Oxford Supercomputing Center. Finally, we thank two anonymous referees and guest editor Carl Anderson, for many helpful suggestions on improving the presentation of the research.

References

Appleby, K., Fakhouri, S., Fong, L., Goldszmidt, G., Kalantar, M., Krishnakumar, S., Pazel, D. P., Pershing, J., & Rochwerger, B. (2001). Oceano—SLA based management of a computing utility. *Seventh IFIP/IEEE International Symposium on Integrated Network Management*, pp. 855–868.

Arlitt, M., & Jin, T. (1999). Workload characterization of the 1998 World Cup web site. *Technical Report* HPL-1999-35R1. HP Laboratories.

Baker, M., Buyya, R., & Laforenza, D. (2000). The grid: International efforts in global computing. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*.

Banga, G., Druschel, P., & Mogul, J. C. (1999). Resource containers: A new facility for resource management in server systems. *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation* (OSDI).

Bartholdi III, J. J., Seeley, T. D., Tovey, C. A., & Vande Vate, J. H. (1993). The pattern and effectiveness of forager allocation among flower patches by honey bee colonies. *Journal of Theoretical Biology, 160*, 23–40.

Baratloo, A., Dasgupta, P., & Kedem, Z. M. (1996). CALYPSO: A novel software system for fault-tolerant parallel processing on distributed platforms. *Proceedings of the Fourth IEEE International Symposium on High Performance Distributed Computing HPDC-4*.

Baratloo, A., Itzkovitz, A., Kedem,Z. M., & Zhao, Y. (1998). Just-in-time transparent resource management in distributed systems. *Technical Report* TR1998-762. Computer Science Department, New York University.

Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm intelligence: Fom natural to artificial systems*. Oxford: Oxford University Press.

Bonabeau, E., & Meyer, C. (2001). Swarm intelligence: A whole new way to think about business. *Harvard Business Review, 19(5)* 107–114.

Brewer, E. A. (2000). Lessons from giant-scale services. http://db.cs.berkeley.edu/~jmh/cs262

Brown, L., Gans, N., Mandelbaum, A., Sakov, A., Shen, H., Zeltyn, S., & Zhao, L. (2002). Statistical analysis of a telephone call center: a queueing-science perspective. http://stat.wharton.upenn.edu/~haipeng/

Buyya, R., Abramson, D., & Giddy, J. (2000). Economy driven resource management architecture for computational power grids. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2000).*

Chase, J. S., Anderson, D. C., Thakar, P. N., & Vahdat, A. M. (2001). Managing energy and servers resources in hosting centers. In *18th ACM Symposium on Operating Systems Principles (SOSP)* (pp. 103–116). New York, NY: ACM Press.

Cicirello, V. A., & Smith, S. F. (2001). Insect societies and manufacturing. In IJCAI-01 *Workshop on Artificial Intelligence and Manufacturing: New AI Paradigm for Manufacturing*.

Crovella, M. E. (1998). Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance '98/ACM SIGMETRICS'98*. http:// www. cs.bu.edu/faculty/crovella/links.html

Czajkowski, K., Foster, I. T., Karonis, N. T., Kesselman, C., Martin, S., Smith, W., & Tuecke, S. (1998). A resource management architecture for metacomputing systems. In *Proceedings of the IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing* (pp. 62–82).

Dasgupta, P., Kedem, Z. M., & Rabin, M. O. (1995). Parallel processing on networks of workstations: A fault-tolerant, high performance approach. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS-15).*

Dusseau, A. C., Arpaci, R. H., & Culler, D. E. (1996). Effective distributed scheduling of parallel workloads. In *ACM SIGMETRICS'96 Conference on the Measurement and Modeling of Computer Systems* (pp. 25–36). New York, NY: ACM Press..

Dusseau, A. C., & Culler, D. E. (1997). Extending proportional-share scheduling to a network of workstations. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97).* New York, NY: ACM Press.

Ensim (2004). http://www.ensim.com/products/index.html

Ford, B. & Susarla, S. (1996). CPU inheritance scheduling. In *Usenix Association Second Symposium on Operating Systems Design and Implementation (OSDI)* (pp. 91–105).

Fox, A., Gribble, S. D., Chawathe, Y., Brewer, E. A., & Gauthier, P. (1997). Cluster-based scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16).*

De Farias, D. P., King, A. J., Squillante, M. S., & Roy, B. V. (2002). Dynamic control of web server farms. *INFORMS Conference–Revenue Management Section*, Columbia University.

Gelenbe, E. (Ed.) (2000). Internet traffic: periodicity, tail behavior and performance implication. *System Performance Evaluation: Methodologies and Application* (Chapt. 2). Boca Raton, FL: CRC Press.

Hill, J. M. D., Donaldson, S. R. & Lanfear, T. (1998). Process migration and fault tolerance of BSPlib programs running on networks of workstations. D. J. Pritchard & J. Reeve, *European Conference on Parallel Processing* (Europar-98) (pp. 151–164).

HP (2004). http://www.hp.com/hps/infrastructure/in_datacenter.html

IBM (2003). http://www-3.ibm.com/services/e-business/hosting/managed_hosting.html

Jayram, T. S., Kimbrel, T., Krauthgamer, R., Schieber, B., & Sviridenko, M. (2001). Online server allocation in a server farm via benefit task systems. In *33rd ACM Symposium on Theory of Computing*. New York, NY: ACM Press.

Jones, M. B., Rosu, D., & Rosu, M. (1995). Modular real-time resource management in the Rialto operating system. *Technical Report* MSR-TR-95-16, Microsoft Research.

Jones, M. B., Rosu, D., & Rosu, M. (1997). CPU reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Symposium on Operating Systems Principles* (pp. 198–211). New York, NY: ACM Press.

Little (1994). C++SIM. University of Newcastle Upon Tyne. http://cxxsim.ncl.ac.uk/

Litzkow, M., Tannenbaum, T., Basney, J., & Livny, M. (1997). Checkpoint and migration of UNIX processes in the Condor distributed processing system. *Technical Report* 1346, Computer Science Department, University of Wisconsin at Maddison.

Manasse, M. S., McGeoch, L. A., & Sleator, D. D. (1988). Competitive algorithm for online problems. In *20th Annual ACM Symposium on Theory of Computing* (pp. 323–333).

Markoff, J. (2003). IBM says it benefits two ways from ondemand computing. *New York Times* 13th November. http://www.iht.com/articles/117552.html.

Nakrani, S., & Tovey C. (2004). Honey bee waggle dance protocol and autonomic server orchestration in internet hosting centers. In *Workshop on Nature Inspired Approaches to Network and Telecommunication in 8th International Conference on Parallel Problem Solving from Nature,*

Nielsen, J. (2000). *Designing web usability.* Berkeley, CA: Pearson Education/New Rider Publishing.

Palmer, J., & Mitrani, I. (2003). Dynamic server allocation in heterogeneous clusters. *Technical Report* CS-TR:799, School of Computing Science, University of Newcastle, UK.

Seeley T. D. (1995). *The wisdom of the hive*. Cambridge, MA, Harvard University Press.

Seeley, T. D., Camazine, S., & Sneyd, J. (1991). Collective decision making in honey bees: How colonies choose among nectar sources. *Behavioral Ecology and Sociology,* 28, 277–290.

Sychron (2004). http://www.sychron.com/

Verio (2003). http://hosting.verio.com/

Waldspurger, C. A., & Weihl, W. E. (1994). Lottery scheduling: Flexible proportional-share resource management. In *Proceedengs of the First Symposium on Operating Systems Design and Implementation* (pp. 1–11) New York, NY: ACM Press.

Waldspurger, C. A. & Weihl, W. E. (1995). Stride scheduling: Deterministic proportional-share resource management. *Technical Memorandum* MIT/LCS/TM-528 MIT Laboratory for Computer Science.

Wolf, J. L., & Yu, P. S. (2001). On balancing the load in a clustered web farm. *ACM Transactions on Internet Technology*, 1,(2), 231–261.

About the Authors

Sunil Nakrani is a D.Phil. student in the Computing Laboratory at the University of Oxford. He received his MSc degree in communication engineering from Imperial College, University of London, 1990. Before returning to pursue a D.Phil. degree, he worked for IBM Corp. focusing on networking and transaction processing. He has held positions as software engineer at the IBM Hursley Laboratory in Winchester, UK and at the IBM Networking Laboratory in Research Triangle Park, North Carolina, USA. It was during this professional engagement that he encountered the server allocation problem.

Craig Tovey is a professor in the School of Industrial and Systems Engineering and in the College of Computing at Georgia Tech. He received an A.B. in applied mathematics from Harvard College in 1977 and an M.S. in computer science and Ph.D. in operations research from Stanford University in 1981. His principal research and teaching activities are in optimization, probabilistic analysis, and natural systems. He received a Presidential Young Investigator Award in 1985, the 1989 Jacob Wolfowitz Prize, and a Senior Research Associateship from the National Research Council in 1990. He was named an Institute Fellow at Georgia Tech in 1994. *Address*: School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. E-mail: ctovey@isye.gatech.edu