

# On Honey Bees and Dynamic Allocation in an Internet Server Colony <sup>†</sup>

Sunil Nakrani<sup>1</sup> and Craig Tovey<sup>2</sup>

1. Computing Laboratory, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK. *Corresponding author*: Sunil.Nakrani@comlab.ox.ac.uk
2. School of Industrial and Systems Engineering, Georgia Tech, Atlanta, GA 30332, USA.

## Abstract

Internet server colonies co-host web services on servers that are leased by its customers on a *pay-per-use* Service Level Agreement(SLA). Given multiple co-hosted web services, the problem encountered by a server colony is to allocate available servers to maximise revenue. The limited number of servers, costs of migrating from one service to another, and unpredictability of future HTTP request loads pose a significant challenge for optimising server allocation.

Based on the many similarities between server allocation and nectar collection in honey bee colonies, we propose a new *honey bee* algorithm founded on self-organisation of honey bee colonies to allocate foragers among food sources. This decentralised algorithm dynamically allocates servers to satisfy the unpredictable HTTP request loads. We compare it against an *omniscient* algorithm that computes an optimal allocation policy, a *greedy* algorithm that uses past history as a guide to the future to compute allocation policy, and an *optimal-static* algorithm that computes omnisciently the best among all possible static allocation policies. We use HTTP trace data from a commercial service provider and synthetically generated HTTP requests for evaluating performance of our algorithm.

The experimental results show that our algorithm performs better than static or greedy, indeed very close to the optimal omniscient level for highly variable request loads. In contrast, it is outperformed by *greedy* for some low variability access patterns. This suggests that real honey bee colony forager allocation, which is suboptimal for static food sources, possesses a counterbalancing responsiveness to food source variability.

*Keywords*: web-servers, honey bee, foraging, self-organisation, Internet infrastructure, SLA, server farm, service performance management.

## 1 Introduction

The web landscape offers myriad Internet services such as online stock trading, banking, ticket reservations, auctions and so on. Consequently, Internet computing infrastructure is being increasingly relied upon for *day-to-day* operations by a rapidly growing fraction of human civilisation. Difficulty in provisioning server resources for such services stem from the fact that user behaviour is difficult to predict and unconstrained demand can create sudden surge in load (Chase et. al., 2001).

An emerging market trend in Internet computing is the proliferation of large data centres, based on common hardware platforms, that co-host third-party web content on servers for a fee (IBM,2003; Verio,2003). This managed hosting model benefits from economies of scale and presents an opportunity for dynamic capacity provisioning whilst shielding web-content owners from capital overhead and unpredictable demand. Given that the hosting fee may be negotiated on the basis of requests served, the hosting centre's objective is to dynamically allocate finite servers to co-hosted services, taking into account switching costs, such that revenue is maximised.

In this paper, we model the dynamic server allocation problem and propose a *biologically-inspired* approach to this optimisation problem in a managed *Internet server colony*. Specifically, our work

---

<sup>†</sup>To appear in Proceedings of 2<sup>nd</sup> International Workshop on The Mathematics and Algorithms of Social Insects, Atlanta, Georgia, USA. December 15-17, 2003

has been inspired by the study of honey bee colonies and the behaviour of forager bees, characterised by decentralised and elementary interactions, that effect a complex collective behaviour to solve the problem of adequate food collection to ensure survival of the colony. The new *Honey Bee* algorithm models servers and HTTP request queues in an Internet server colony as, respectively, foraging bees and flower patches. We perform experimental work on a simulated web content hosting centre using trace data from a commercial service provider and trace data derived from a simulated web-server. Our results indicate that the *honey bee* algorithm adapts well to highly variable request loads.

The remainder of the paper is organised as follows: In section 2, we describe how honey bee colonies deploy forager bees to collect nectar amongst diverse flower patches. The server colony model and the mapping of server allocation to honey bees forager deployment is given in section 3. The server allocation algorithm based on honey bee foraging behaviour is described along with the greedy, omniscient and optimal-static algorithms, to which it is compared, in section 4. In section 5, we describe the simulation model used in the experiments and compare the performance of the honey bee algorithm against other algorithms. Finally, in section 6, we speculate on how our results may point to a survival advantage conferred by the forager allocation mechanism in real honey bee colonies.

## 2 Swarm Intelligence– Honey Bee Colony

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 behaviour borne out of primitive interactions amongst members of the group to solve problems beyond capability of individual members. Such colonies are characterised by (i) Self-Organisation: Decentralised and unsupervised coordination of activities, (ii) Adaptiveness: Response to dynamically varying environment and (iii) Robustness: Accomplishing groups' objective even if some members of the group are unsuccessful. These group level adaptive properties, it has been suggested, lend themselves well to distributed optimisation problems in telecommunication, manufacturing, transportation and so on (Bonabeau et. al., 1999; Bonabeau and Meyer, 2001; Cicirello and Smith, 2001).

A model of self-organisation that takes place within a colony of honey bees has been presented in (Seeley, 1995). Specifically, it describes interactions between members of the colony and the environment that leads to dynamic distribution of foragers to efficiently collect nectar from array of flower patches (food sources) that are capricious in terms of *profitability* to the colony. In brief, foraging bees visiting flower patches not only return to the hive with nectar but also with a profitability rating of respective patches. At the hive, forager bees interact with receiver bees to offload collected nectar which also 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) establishes the length of the enlisting signal known as *waggle dance*.

The waggle dance is performed on the dance floor where inactive foragers can observe and follow. Effectively, each active forager bee provides a feedback on her local flower patch whilst observing bees have access to the set of attractive food sources being capitalised 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-organised proportionate allocation pattern, derived from multiple and proportionate feedback on *goodness* of food sources, is described in (Seeley et. al., 1991) and validated by experimental study on real honey bee colonies. The model given by (Bartholdi et. al., 1993) predicts a steady state pattern of forager allocation where rate of value accumulation equalises among the patches being exploited. Intriguingly, this pattern is *not* optimal (unlike the famed hexagonal comb structure). Note that this sub-optimality is with respect to a static problem.

### 3 An Internet Server Colony

There is a trend toward a web computational model where a single hosting provider manages content delivery of myriad Internet services to HTTP requests from global audience (IBM,2003; Verio,2003). Under this paradigm, the content owners purchase resources on a *pay-per-use* Service Level Agreement(SLA) and hosting provider shares resources among its clients. The model benefits from economy of scale due to resource sharing and insulation from maintenance and over-provisioning costs for its clients. *Pay-per-request-served* is a common way in which *pay-per-use* SLA is negotiated between hosting provider and content owner. Such an SLA implicitly acknowledges that limited resources, migration costs, and uncertainty may lead to overloading and lost requests (Jayram et. al., 2001). Consequently, the hosting provider is incented to maximise the total value of served requests.

The prevailing architecture of choice for a server colony is an ensemble of commodity servers which are partitioned into clusters (virtual servers) and configured to host web services (Brewer, 2000). Incoming HTTP 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 et. al., 1997; Chase et. al., 2001). During server migration from one cluster to another, it becomes unavailable due to time involved in *scrubbing* of existing and installation of new web application (Appleby et. al., 2001). As far as user audience of any service is concerned, the server colony appears as a virtual server.

In our model, we map the Internet Server allocation problem to the honey bee forager allocation as follows: We consider respective service queues as available forage sites i.e. flower patches. The dynamic and unpredictable nature of incoming HTTP requests arrival patterns, which is a function of user behaviour, is equivalent to flower patch volatility, which is a function of daily climatic and other changes. A unit of allocation in the Internet server colony i.e. a single server, equates to a single forager bee. Therefore, a given virtual server engaged in serving a specific HTTP requests queue can be thought of as a group of forager bees engaged in collecting nectar from a specific foraging site. Finally, we think of the ensemble of servers as a colony of forager bees, and the service queues relating to co-hosted web services as sources of nectar to be exploited profitably. Different services have different time cost and *value-per-request-served*, just as different flower patches have different *forager-round-trip times* and nectar quality. We believe that the principal characteristics of the Internet server problem match very closely to forager allocation in the colony of honey bees. For example, in addition to the correspondences mentioned, the migration cost mimics the changeover cost when a bee switches from one patch to another (Seeley et. al., 1991). Moreover, the self-organising property is well suited to managing resources in a large server colony where traditional administrative methods can be cumbersome and prohibitive.

### 4 Server Allocation Algorithms

This section details algorithms to allocate servers to competing co-hosted services. The basic challenge is to determine server demand of each service based on its current HTTP request load. We assume there are  $M$  groups comprised from  $n$  servers, called virtual servers  $VS_0 \dots VS_{M-1}$ , and service queues  $Q_0 \dots Q_{M-1}$ . A server  $s_i \in VS_j$  serving queue  $Q_j$  is paid  $c_j$  cents per request served.

#### 4.1 Honey Bee

We present an allocation algorithm for an Internet server colony inspired by forager allocation in the honey bee colony. Let any server in the server colony be either a forager or a scout server. Also, let the *dance floor* be represented by an advertboard and *waggle dance* be represented by an advert.

A server  $s_i \in VS_j$ , 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 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  $VS_j, j : 0 \dots (M-1)$  if it is a scout. The probability  $r_i$  is dynamic and changes as a function of forager/scout servers' own profit rate and server colony's overall profit rate. The profit rate,  $P_i$ , for a server  $s_i$  is given by  $P_i = \frac{c_i R_i}{T_i}$  where  $R_i$  is the total number of request served by a given server in the time interval  $T_i$ . The server colony's overall profit rate,  $P_{colony}$ , is given by  $P_{colony} = \frac{1}{T_{colony}} \sum_{j=0}^{(M-1)} c_j R_j$  where  $R_j$  denotes the total number of request served by  $VS_j$  in the time interval  $T_{colony}$ . A server  $s_i \in VS_j$  serving queue  $Q_j$  determines its *profitability* by comparing profit rate  $P_i$  with  $P_{colony}$ . It will adjust  $r_i$  according to the lookup table 1. In essence, a server is more likely to read an advert if it is serving a queue that is low in profitability. We point out that this is effectively the same as in real honey bee colonies given that dancing foragers never get recruited to another patch and foragers use global information (hive's nectar intake rate, time to find a receiver bee) to decide whether to dance or not. Thus, bees at profitable patch decrease the chance of following another dance(advert).

Table 1: Lookup table for adjusting probability of reading the advertboard

Profit Rate	P[Reading] $r_i$
$P_i \leq 0.5P_{colony}$	0.60
$0.5P_{colony} < P_i \leq 0.85P_{colony}$	0.20
$0.85P_{colony} < P_i \leq 1.15P_{colony}$	0.02
$1.15P_{colony} < P_i$	0.00

## 4.2 Omniscient

The omniscient algorithm provides an upper bound on possible profitability. (It would be both informationally impossible and computationally prohibitive in practise.) It computes, by dynamic programming, the optimal server allocation given complete knowledge of future HTTP request loads. For an Internet server colony, the recursive profit function is given by  $f_t(\pi_t^*, A_t) = \{P(\pi_t^*, A_t) + f_{t+1}(\pi_{t+1}^*, A_{t+1})\}$  for  $0 \leq t \leq N-1$  and  $f_t(\pi_t^*, A_t) = 0$  for  $t \geq N$ . where  $t$  denotes discrete time steps,  $\pi_t^*$  denotes optimal server allocation policy,  $A_t$  denotes total HTTP request arrivals in the current time step  $t$  plus residual requests from time step  $t-1$ ,  $P(\pi_t^*, A_t)$  denotes maximum profit made with optimal policy  $\pi_t^*$  and arrivals  $A_t$ . For time horizon split into  $N$  time steps,  $f_0(\pi_0^*, A_0)$  represents the maximum profit that is possible from time steps  $0 \dots (N-1)$  with allocation policies  $\pi_0^* \dots \pi_{N-1}^*$ . It would be remiss of us not to point out that this algorithm is, both, time and space intensive as a function of problem size. For example, 50 server to be allocated across 3 virtual servers, 11 interpolation buckets, requires atleast 1.3 GB for interim result tables and exploration of 167 million states per time step.

## 4.3 Greedy

The greedy allocation algorithm represents a standard conventional heuristic approach to the problem. It allocates servers based on what would have been optimal during the preceding time period. The idea is that the immediate past is the best available guide to the uncertain future. In particular, when a specified time interval  $T_A$  expires, the algorithm reallocates servers across the server colony based on the optimal profit that could have been made given the HTTP request arrivals data from expired time interval and the current allocation (thus taking migration costs into account). The optimal profit is given by  $P_{opt-Greedy} = \max_{\{0 \leq i \leq (k-1)\}} \{P(\pi_i, A_{T_A})\}$ , over  $k$  possible allocation policies with  $A_{T_A}$  denoting the total number of request that arrived during the time interval  $T_A$ . Thus, algorithm chooses, for time interval  $T_A + 1$ , an allocation policy  $\pi_i$  that maximises  $P_{opt-Greedy}$  for the time interval  $T_A$ .

## 4.4 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 profitability of many SLA providers, which do not change their allocations more often than once a month. Their profitability will be lower than optimal-static value because they cannot know the coming month’s HTTP request load in advance. It attempts to determine the best static allocation policy for a time horizon split into  $N$  discrete time steps:  $P_{opt-stat} = \max_{\{0 \leq i \leq (k-1)\}} \left\{ \sum_{t=0}^{(N-1)} P(\pi_i, A_t) \right\}$ , over  $k$  possible server allocation policies,  $P(\pi_i, A_t)$  represents the profit made with server allocation policy  $\pi_i$  and HTTP request load at time step  $t$ . The best static-optimal allocation policy is the one that maximises  $P_{opt-stat}$  over  $0 \leq t \leq N - 1$  for time horizon split into  $N$  time steps.

## 5 Experimental Results

In this section we describe the simulation model of the dynamic server allocation problem and present experimental results comparing the performance of honey bee algorithm with Omniscient, Greedy, and Optimal-Static algorithms using synthetic HTTP request data as well as trace data from a commercial service provider. These experiments consider the case of an Internet server colony with 2 and 3 virtual server(websites) composed from a total of 50 servers. We use the performance metric of total revenue earned by the server colony from serving HTTP requests.

### 5.1 Simulation Model

We have developed discrete event simulation models for dynamic server allocation algorithms—Honey Bee, Omniscient, Greedy, and Optimal-Static. All models and algorithms are implemented in C++SIM(Little, 1994) on Sparc Sun-Blade-100 running SunOS 5.9. The following assumptions are common to all models: All servers are homogeneous in terms of processing capacity and employ *First-Come-First-Serve* scheduling policy. The time to serve a HTTP 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 fixed migration time (Appleby, 2001). This time is accounted by the fact that a server must be purged of current application and data, reloaded with a new application and data of the website to which it is allocated. A stream of HTTP requests arriving for a particular virtual server(website) is held in a 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 HTTP request stream. The trace data are from an international commercial service provider (confidentiality agreement restricts us from disclosing their name), and from simulated data from inhomogeneous Poisson processes (Brown et al. 2002).

In the Honey Bee model, servers can be reallocated at any time. There is one server per virtual server that is designated as a *scout* and rest 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 colony at any time whilst *forager* servers randomly reallocates themselves in response to an advert read for a particular virtual server. Each server(*scout/forager*) advertises its own virtual server by placing an advert on an advertboard with given time duration. The advertboards is kept up-to-date by purging adverts with expired time duration. In Omniscient and Greedy, we make the assumption that servers are reallocated at the beginning of each allocation interval and do not change during the remainder of this interval. For the Optimal-Static model, we make the assumption that servers allocated at the beginning remain unchanged for the complete duration of simulation and, therefore, do not suffer any migration costs. All simulation models except Honey Bee and Greedy require an allocation policy which is computed offline using the request arrival data. For example, Omniscient policy is computed using dynamic programming. The common parameters for all models are as follows— Total servers: 50; Revenue-per-request: 0.5 cent; Mean Execution Time: 15 mS; Request Waiting Threshold: 10 seconds; Balk rate: 1.01 seconds and Balk Probability: 0.04.

Specific parameters from each model as follows– Omniscient: Server migration Time: 300 seconds, Allocation Interval: 900 seconds; Honey Bee: Advert Posting probability: 0.1, Scouts Per Virtual Server: 1, Server Migration Time: 300 seconds; Greedy: Server Migration Time: 300 seconds, Server Allocation Interval: 1800 seconds.

## 5.2 Experiments

Table 2 shows the revenue earned by the server colony over a 24 hour period for respective algorithms configured as 2 and 3 virtual servers with synthetic HTTP request pattern as shown in fig. 1. Under this environment, our algorithm adapts well to dynamic changes in the arrival pattern. It outperforms Greedy and Optimal-Static whilst being within 11.7% and 11.6% of the Omniscient for 2 and 3 virtual server configuration respectively. We point out that for a low variability request pattern (with (peak:trough) 10:1/5:1), Greedy algorithm outperforms honey bee by 1.27%/1.21% on the 2 and 3 virtual server configurations, respectively.

Table 2: Revenue for synthetically generated HTTP requests

Algorithm	Revenue(\$)			
	2-VS		3-VS	
	Actual	Projected	Actual	Projected
Omniscient	970,636.00	991,298.00	1,119,400.00	1,159,039.55
Honey Bee	868,949.00	—	1,003,050.00	—
Greedy	836,077.00	—	968,087.00	—
Optimal-Static	810,510.00	827,455.00	860,363.00	875,210.90

Table 3 shows the revenue earned by the server colony over a 24 hour trace-driven simulation from HTTP request pattern that is derived from real Internet services as shown in fig. 1. For 2 virtual server configuration, HTTP traces are used from Services B and C whilst A, B and C traces are used for 3-VS configuration. Under these request arrival pattern, Honey Bee algorithm outperforms Greedy and Optimal-Static for 2 virtual configuration and performs within 1.55% of the Omniscient. For 3 virtual server configuration, Honey Bee algorithm outperforms Greedy, and Optimal-Static whilst performing within 7.95% of the Omniscient.

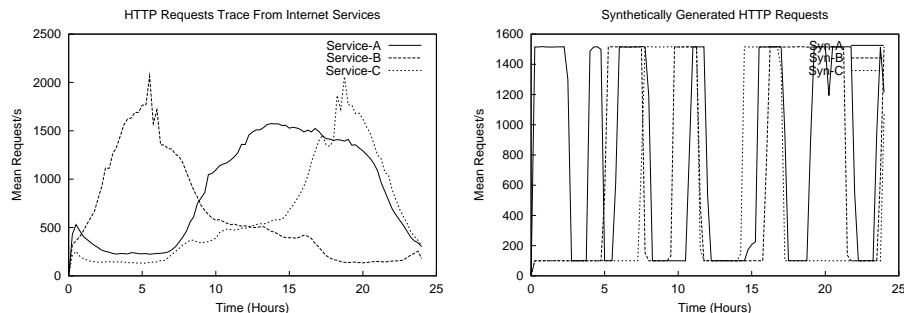


Figure 1: Average HTTP request access patterns.

To test the adaptability of our algorithm, we used HTTP request patterns that are synthetically generated and range across varying degrees of inhomogeneous Poisson arrivals. We use a numerical scale to denote degree of inhomogeneity(peak:trough) as follows: 1 : 1(0), 10 : 1(1), 15 : 1(2) and 25 : 1(3). The server colony is configured for 3 Virtual Servers. We normalise the performance of all algorithm to Omniscient and depict results in fig. 2. Interestingly, for zero variability, honey bee performs well as all other algorithms despite the constant migration overhead. This is attributed to

Table 3: Revenue from real Internet service traces.

Revenue(\$)				
Algorithm	2-VS		3-VS	
	Actual	Projected	Actual	Projected
Omniscient	1,066,440.00	1,071,741.83	1,336,960.00	1,352,872.18
Honey Bee	1,050,110.00	—	1,238,470.00	—
Greedy	1,043,400.00	—	818,040.00	—
Optimal-Static	844,822.00	845,076.00	1,108,360.00	1,108,554.58

the fact that servers at all three queues are earning revenue at a rate that is equal to or better than the combined rate of the hosting centre which results in decrease in the probability of reading an advert and hence reduction in migration. As the variability is increased, honey bee performance degrades and converges to within 10-15% of the Omniscient which suggests its adaptability to dynamic load. We emphasise 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 common-sense scaling reasoning, and froze those values before we ran any test cases.

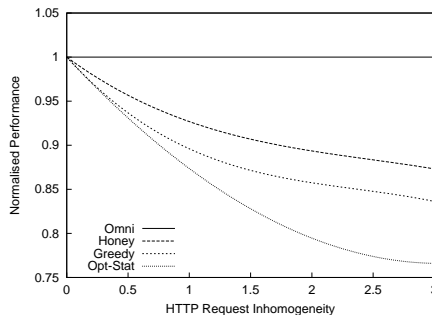


Figure 2: Adaptability to inhomogeneous requests pattern.

## 6 Conclusion

In this paper, we proposed a new *honey bee* allocation algorithm based on self-organised behaviour of foragers in honey bee colonies, and the 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 supported the effectiveness of the algorithm, particularly in the highly dynamic and unpredictable (Arlitt and 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. By the same token, we speculate that a real honey bee colony is quite adaptive to changes in flower patch availability and quality, and that its allocation pattern may be considerably less suboptimal when evaluated in a realistic dynamic environment rather than a static one. One possible explanation for a performance difference is that static optimisation requires equalisation of derivatives (marginal rates), implicitly requiring a *marginal rate bee* for each patch, thus limiting migration rates to one bee at a time. The honey bee algorithm trades off static optimality for adaptiveness. It has no marginal rate bee, but it has the ability to migrate several bees at once, increasing its responsiveness to changed circumstances.

## Acknowledgements

We would like to express our gratitude to Tom Seeley for helpful discussions and insights on the inner workings of honey bee colonies.

## References

- Appleby K., Fakhouri S., Fong L., Goldszmidt G., Kalantar M., Krishnakumar S., Pazel D. P., Pershing J., and Rochwerger B., 2001. Oceano - SLA Based Management of a Computing Utility. *7th IFIP/IEEE International Symposium on Integrated Network Management*.
- Arlitt M., and Jin T., 1999. Workload Characterisation of the 1998 World Cup Web Site. *Technical Report HPL-1999-35R1*. HP Laboratories.
- Bartholdi III J. J., Seeley T. D., Tovey C. A., and 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.
- Bonabeau E., Dorigo M., and Theraulaz G., 1999. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.
- Bonabeau E., and Meyer C., 2001. Swarm Intelligence: A whole New Way to Think About Business. *Harvard Business Review*. 107-114.
- Brewer E. A., 2000. Lessons from Giant-Scale Services. <http://db.cs.berkeley.edu/~jm/cs262>.
- Brown L., Gans N., Mandelbaum A., Sakov A., Shen H., Zeltyn S., and Zhao L., 2002. Statistical Analysis of a Telephone Call Centre: A Queueing-Science Perspective. <http://stat.wharton.upenn.edu/~haipeng/>.
- Chase J. S., Anderson D. C., Thakar P. N., and Vahdat A. M., 2001. Managing Energy and Servers Resources in Hosting Centres. *18th ACM Symposium on Operating Systems Principles (SOSP)*.
- Cicirello, V. A., and Smith, S. F. 2001. Insect Societies and Manufacturing. *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. *Proceedings of Performance '98/ACM SIGMETRICS '98*. <http://www.cs.bu.edu/faculty/crovella/links.html>
- Fox A., Gribble S. D., Chawathe Y., Brewer E. A., and Gauthier P., 1997. Cluster-Based Scalable Network Services. *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*. St. Malo, France.
- IBM, 2003. [http://www-3.ibm.com/services/e-business/hosting/managed\\_hosting.html](http://www-3.ibm.com/services/e-business/hosting/managed_hosting.html).
- Jayram T. S., Kimbrel T., Krauthgamer R., Schieber B., and Sviridenko M., 2001. Online Server Allocation in a Server Farm via Benefit Task Systems. *33rd ACM Symposium on Theory of Computing*. July 6-8, Hersonissos, Crete, Greece
- Little, 1994. C++SIM. University of Newcastle Upon Tyne. <http://cxxsim.ncl.ac.uk/>.
- 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.
- Seeley T. D., 1995. *The Wisdom of the Hive*. Pub. Harvard University Press.
- Verio, 2003 <http://hosting.verio.com/>.