2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops

A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing

Martin Randles, David Lamb, A. Taleb-Bendiab
School of Computing and Mathematical Sciences
Liverpool John Moores University
Liverpool, UK
{M.J.Randles, D.J.Lamb, A.Taleb-Bendiab}@ljmu.ac.uk

Abstract— **The anticipated uptake of Cloud computing, built on well-established research in Web Services, networks, utility computing, distributed computing and virtualisation, will bring many advantages in cost, flexibility and availability for service users. These benefits are expected to further drive the demand for Cloud services, increasing both the Cloud's customer base and the scale of Cloud installations. This has implications for many technical issues in Service Oriented Architectures and Internet of Services (IoS)-type applications; including fault tolerance, high availability and scalability. Central to these issues is the establishment of effective load balancing techniques. It is clear the scale and complexity of these systems makes centralized assignment of jobs to specific servers infeasible; requiring an effective distributed solution. This paper investigates three possible distributed solutions proposed for load balancing; approaches inspired by Honeybee Foraging Behaviour, Biased Random Sampling and Active Clustering.**

Keywords- Cloud Computing; Service-Oriented Architecture; Load Balancing

I. INTRODUCTION

Cloud computing is starting to provide an environment whereby Web Services can realise their initially promised potential. Up to the present time, Web Services within Service Oriented Architectures (SOA) have been used in a limited way within business boundaries for integration of applications [1]. The predicted widespread availability and uptake of web-delivered services has not occurred to any great scale [2]. Commonly cited reasons include; high complexity and technical expertise required, large expense of implementation and maintenance, and the inflexibility and lack of widely accepted standards for defining service cooperation, identification and orchestration [3]. These concerns arise as a consequence of associated service architecture management and maintenance difficulties. The scale and complexity of these systems makes centralized governance of specific servers infeasible; requiring effective distributed solutions. Distributed governance, achieved through local knowledge, is a vital prerequisite in order to enable the vision inherent in the Internet of Services/Things (IoS/T) model of service/hardware provision [4]. Systems may form digital ecosystems (systems of systems) providing the structure, communication and coordination necessary for a given federation (co-operative collection) of services to deliver the required system goals or intentions [5]. For this model to work efficiently, self-organisation and management capabilities must pervade every layer or plane of these digital ecosystems. Global outcome can thus be controlled through local interaction via self-*, autonomic properties and decentralised feedback control mechanisms [6]. In [7] the authors presented initial work towards this goal; enabling scalable virtualisation of these systems. Load balancing was identified as a major concern to allow Cloud computing to scale up to increasing demands. A distributed solution is required, as it is not practical or cost efficient in many cases to maintain idle service/hardware provision merely to keep up with all identified demands. Equally, when dealing with such complexity, it is impossible to fully detail all system future states. Therefore, it is necessary to allow local reasoning through distributed algorithms on the current system state. Our early work suggested that efficient load balancing cannot be achieved by individually assigning jobs to appropriate servers as the Cloud computing systems scale up and become more complex; rather jobs must be assigned with some uncertainty attached. As such, load balancing ought to be achieved using an inferred system state; based on locally gathered data - or the system must be optimised in structure to allow load balancing to be more easily provisioned.

This paper considers three potentially viable methods for load balancing in large scale Cloud systems. Firstly, a nature-inspired algorithm may be used for self-organisation, achieving global load balancing via local server actions. Secondly, self-organisation can be engineered based on random sampling of the system domain, giving a balanced load across all system nodes. Thirdly the system can be restructured to optimise job assignment at the servers. This paper aims to provide an evaluation and comparative study of these approaches, demonstrating distributed algorithms for load balancing.

Recently numerous nature-inspired networking and computing models have received a lot of research attention in seeking distributed methods to address increasing scale and complexity in such systems. The honey-bee foraging solution, used as the case study in [7], is investigated as a direct implementation of a natural phenomenon. Then, a

978-0-7695-4019-1/10 $26.00 © 2010 IEEE
DOI 10.1109/WAINA.2010.85

distributed, biased random sampling method that maintains individual node loading near a global mean measure is examined. Finally, an algorithm for connecting simile services by local rewiring is assessed as a means of improving load balancing by active system restructuring. The paper proceeds with a primer of Cloud Computing in Section II, detailing the need for distributed solutions. Section III discusses the three example solutions introduced above. Sections IV and V present results, analysis and comparison of these solutions, whilst Section VI concludes the paper and identifies future work.

II. Cloud Computing

Cloud computing, as a current commercial offering, started to become apparent in late 2007 [8]. It was intended to enable computing across widespread and diverse resources, rather than on local machines or at remote server farms. Although there is no standard definition of Cloud Computing, most authors seem to agree that it consists of clusters of distributed computers (Clouds) providing on-demand resources or services over a network with the scale and reliability of a data centre [9]; notions familiar from resource virtualisation and Grid computing. Where these clusters supply instances of on-demand Cloud computing; provision may be comprised of software (e.g. Software as a Service, SaaS) or of the physical resources (e.g. Platform as a Service, PaaS). The Amazon Elastic Compute Cloud (Amazon EC2) [10] is an example of such an approach, where a computing platform is provided. In common with many commercial approaches *provision* is the primary objective; management and governance handled via redundancy or replication, scaling capacity up or down as required. In contrast the authors proposed a Cloud Coordination framework in 2005 with the notion of a Cloud being a system of loose boundaries, which interacts and merges with other systems [11]. This definition of a Cloud is refined to a federation of interacting services and resources, which share and pool resources for greater efficiency. Thus governance, in general, and scalability are handled as part of the separated coordination framework. This separation permits sophisticated implementations of management techniques, such as load balancing.

Until recently the major works on load balancing assumed homogeneous nodes. This is obviously unrealistic for most instances of Cloud computing, as defined herein, where dynamic and heterogeneous systems are necessary to provide on-demand resources or services. In the Amazon EC2, dynamic load balancing is handled by replicating instances of the specific middleware platform for Web services. This is achieved through a traffic analyser, which tracks the time taken to process a client request. New instances of the platform are started when the load increases beyond pre-defined thresholds [12]. Therefore, combinations of rules prescribe the circumstances and solution for load balancing. As the systems increase in size and complexity, these rule sets become unwieldy and it may not be possible to maintain a viable monitoring and response cycle to manage the computational workload. In short, the size of these systems may exceed the capabilities of attached meta-systems to maintain a sufficiently agile and efficiently organized load balancing (or general management) rule-set. When so many management rules are defined within a system, there are likely to be conflicts amongst the rules; interactions and impact are in general very difficult to analyse. For instance, the execution of one rule may cause an event, triggering another rule or set of rules, dependent on current state. These rules may in turn trigger further rules and there is a potential for an infinite cascade of policy execution to occur. Additionally these rules are static in nature; there is usually no provision for rule refinement or analysis. A system rule requiring alteration or adjustment necessitates the system or component being taken offline, reprogrammed and deployed back into the system.

Thus, as an example management task; a load balancing system is required that self-regulates the load within the Cloud's entities without necessarily having to have full knowledge of the system. Such self-organised regulation may be delivered through distributed algorithms; directly implemented from naturally observed behaviour, specifically engineered to maintain a globally-balanced load, or directly altering the topology of the system to enhance the natural pattern of load distribution.

III. Distributed Load Balancing for the Cloud

There is a need, as discussed in the previous section, for load balancing in large-scale complex systems (such as massively-scaling Cloud computing domains) that is less reliant on exhaustive monitoring of the domain and the associated deliberation. Instead methods are sought that promote load balancing on the global (Cloud) scale via actions and interactions at the component (individual server) level. Three specific instances of distributed solutions have been identified involving nature inspired low-level engineered or topologically adjusted outcomes. The remainder of this paper considers an example of each of these solutions, with an extended Honeybee Foraging algorithm, a Biased Random Sampling on a random walk procedure and Active Clustering, followed by a comparative study of the three approaches.

A. Inspiration from the Honeybee

In [13] and [7], load balancing was presented as an illustrative case-study, in line with the envisaged IoS for generic service architecture. A simulation of a self-organizing, beehive-based load-balancing algorithm [14] was used at the application layer for optimum resource allocation. The work examined the allocation of servers in a large-scale SOA; allocating servers to Web applications to maximize effective capacity and usage. The authors consider such a scenario highly applicable to Cloud Computing and its adoption of the pay-per-use (utility computing) paradigm.

The algorithm used in this paper follows that from [14] for co-ordination of servers hosting Web Services. It is one of a number of applications inspired by the believed behaviour of a colony of honeybees foraging and harvesting food. Forager bees are sent to search for suitable sources of food; when one is found, they return to the hive to advertise this using a display to the hive known as a "waggle dance". The suitability of the food source may be derived from the quantity or quality of nectar the bee harvested, or its distance from the hive. This is communicated through the *waggle dance* display. Honey bees then follow the forager back to the discovered food source and begin to harvest it. Upon the bees' return to the hive, the *remaining* quantity of food available is reflected in their *waggle dances*, allowing more bees to be sent to a plentiful source, or exploited sources to be abandoned.

This biologically-inspired technique is now used as a search algorithm in a variety of computing applications; seeming particularly scalable on a fluctuating underlying system. As such, when applied to load balancing; as the demand for Web Services fluctuates, it is desirable to dynamically allocate servers to regulate the system against demand. The considered *honeybee-based* load balancing technique uses a collection of servers arranged into *virtual servers*, each serving a *virtual service queue* of requests. A *profit* is calculated by each server serving a request from a queue - representative of the bees' measure of *quality*. This *profit* measure is calculated based on the server's "cost" in serving the virtual server (e.g. used CPU time), taking into account "reward" allocated by the virtual server for its time (a service-type *revenue* model). Equally, the "dance floor" of the hive is reflected in a distributed shared-space *advert board*. This *advert board* is also used to communicate the notion of a global *colony profit* (per virtual server); a measure of how effectively resources are being used. Depending on domain-specific requirements, this may be a simple aggregation of individual server's CPU idle (wasted) time, or may encompass other metrics, such as average service queue length. A high-level overview of the interactions between servers (clustered into two "virtual" servers), the service queues they serve, and the shared advert board is shown below in Fig. 1 [14].

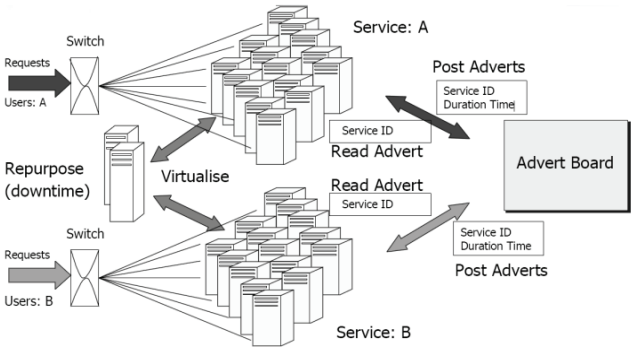

Figure 1. Virtual Server and Advert Boards

In load-balancing operation, each server takes a particular bee role with probabilities p_x or p_r. These values are used to mimic the honeybee colony whereby a certain number of bees are retained as foragers – to explore (p_x); rather than as harvesters – to exploit existing sources. A server successfully fulfilling a request will post on the advert board with probability p_r. A server may randomly choose a virtual server's queue with probability p_x (exploring), otherwise checking for an advert (watching a *waggle dance*). In summary, idle servers (waiting bees) follow one of two behaviour patterns: a server that reads the advert board will follow the chosen advert, then serve the request; thus mimicking harvest behaviour. A server not reading the advert board reverts to forage behaviour; servicing a random virtual server's queue request.

An executing server will complete the request and calculate the profitability of the just-serviced virtual server. The completed server (i.e. returning bee) influences system behaviour by comparing its calculated profit with the colony profit on the advert board, and then adjusts p_x (controlling the explore/exploit ratio) and colony profit accordingly. If the calculated profit was high, then the server returns to the current virtual server; posting an advert for it (*waggle-dancing*) according to probability p_r. If profit was low, then the server returns to the idle/waiting behaviour described above. Initially, every server starts with explore/forage behaviour, and as requests are serviced, the advert/*waggle-dance*-guided behaviour begins to emerge. Given a robust profit calculation method, this behaviour pattern provides a distributed and global communication mechanism; ensuring "profitable" virtual servers appear attractive to and are allocated to available servers.

B. Biased Random Sampling

In this second load balancing approach, the load on a server is represented by its connectivity in a virtual graph. A full analysis of this mechanism is found in [15], with this section providing a brief overview. The initial network is constructed with virtual nodes to represent each server node, with each in-degree mapped to the server's free resources or some measure of desirability. As such, a number (consistent with its available resources) of inward edges are created, connected from randomly-selected nodes. This approach creates a network system that provides a measure of initial availability status, which as it evolves, gives job allocation and usage dynamics.

Edge dynamics are then used to direct the load allocation procedures required for the balancing scheme. When a node executes a new job, it removes an incoming edge; decreasing its in-degree and indicating available resources are reduced. Conversely, when the node completes a job, it follows a process to create a new inward edge; indicating available resources are increased again. In a steady state, the rate at which jobs arrive equals the rate at which jobs are finished; the network would have a static average number of edges.

In ideally-balanced conditions, the degree distribution is maintained close to an Erdős-Rényi (ER) random graph degree distribution [16]. The increment and decrement process is performed via *Random Sampling*. The sampling walk starts at a specific node; at each step moving to a neighbour node chosen randomly. The last node in the sampling walk is selected for the load allocation. The effectiveness of load distribution is considered to increase with walk length, referred to herein as *w*. However, experimentation demonstrated an effective *w* threshold is around *log(n)* steps, where *n* is the network size [15] .

Therefore, *w* is used to control the behaviour of a node upon receiving a job. When a node receives a job it will execute it if the job's current walk length is greater than or equal to the walk length threshold. This node is then referred to as the job's *executing node*. Alternatively, if the walk length is less than the threshold, the job's *w* value is incremented and it is sent to a random neighbour, thus continuing the *Random Sampling* approach. When the job reaches the *executing* node, this allocation is reflected in the graph with the deletion of one of the *executing* node's *in-edges*. Once the job has completed, the result of the allocation is reflected by the creation of a new edge from the *initiating* node to the *executing* node.

To summarise, the balancing graph is altered in the following manner by job execution and completion:

- the *executing* node's in-degree (*available resources*) decreases *during job execution*, followed by:
- the *allocating* node's out-degree (i.e. *allocated jobs*) and the executing node's in-degree (available resources) increases *after job execution*, thus directing future load allocation.

The result is a directed graph, where the direction of the edges leads the propagation for random sampling. The load-balancing scheme is both decentralised and easily implemented using standard networking protocols. As the scheme is decentralised, this makes it suitable for many large network systems such as those required for Cloud computing platforms. As noted in [15], the performance of this load-balancing technique can be further improved by biasing the random sampling toward specific nodes. Hence, selection will be based on a predefined criterion (e.g. computing power or communication latency) rather than selecting the last node in the walk. For instance, the random sampling walk may be directed towards unvisited nodes or nodes with certain properties. Accordingly, the load balancing technique is improved by assigning the new job to the least loaded (highest *in-degree*) node in the walk, instead of the last node in the walk. Therefore, the scalability is identical to standard random sampling, yet the balancing performance is much improved.

C. Active Clustering

In [13] it was discovered that implementation of the honeybee-foraging distributed load balancing solution at the application layer caused a particular topology to emerge at the resource layer. This manifested itself as a small number of services attracting a disproportionate amount of connectivity from cooperating services whilst most services had only a small number of links. As such, well used services may be grouped to deal with load balancing through the topology of the Cloud's resources.

Active Clustering is considered in [17] as a self-aggregation algorithm to rewire the network. This procedure is intended to group like (i.e. similar service-type) instances together. Many load balancing algorithms only work well where the nodes are aware of "like" nodes and can delegate workload to them [18]. Active Clustering consists of iterative executions by each node in the network:

1. At a random time point a node becomes an *initiator* and selects a *matchmaker* node randomly from its current neighbours; the only condition being that the *matchmaker* is of a different type.
2. The *matchmaker* node then causes a link to be formed between one of the *matchmaker's* neighbours that match the type of the *initiator* node.
3. The *matchmaker* removes the link between itself and the *initiator*.

This algorithm was studied extensively in [17], showing the organisation of a complex network towards a steady state. Further works in [19] and [20] adapted the *active* algorithm; the *fast* algorithm does not allow the removal of a link between like nodes, whereas the *accurate* algorithm maintains the number of links and enforces that links between non-like nodes can only be added if another link between heterogeneous nodes is removed. Full details of the complete algorithm that switches between *active*, *fast* and *accurate* as circumstances dictate may be found in [20].

IV. The Experiment

This paper's motivation arose from a study of optimisation of network topology (a representation of the resource layer), using clustering, following exploitation of the honeybee foraging algorithm at the application layer [13]. In order to compare the described algorithms, a series of experiments were conducted using simulations set up in Repast.NET [21]. The experiments were set up replicating the domain described in [20], to allow as direct a comparison of results as possible. The simulations were repeated 20 times each.

For the Honeybee Foraging algorithm experiment, the server colony consisted of *M* virtual server *types* with *N servers* (representative of bees). The round robin method is used to control the advert board reading process. The probability (p_x) that a server reads the advert board (i.e. *forages*) is initially set to 0.2 whilst the probability (p_r) that a successful server writes to the advert board (i.e. performs a *waggle dance*) is 0.8. An advertisement's lifespan on the advert board is equal to 10 ticks. A metric used in [20] was

considered here as a performance measure: *job throughput*, which is the number of completed jobs per elapsed time.

Similarly the *Active Clustering* algorithm for load balancing is assessed. In this experiment a scale free network consisting of *N* nodes is constructed. Again, there are *M* job or server types, and jobs are inserted with the same parameters for the previous experiment. It should be noted here that the results show a throughput variation depending on network organization (nodes of each type are generated and connected randomly in the scale free network so *active adaptive clustering* affects the results).

For the remaining approach, Biased Random Sampling, the parameters are identical to the previous experiments. Here, the node with the greatest free resources on a walk is preferred; to receive the new job, its resources must be greater than or equal to those of the last node on the random sampling.

The results that receive scrutiny in the following section are based on two phases of experiments as described above. The first phase measured *throughput* against *diversity*; all experiments ran for *N=1000*, and an increasing value of *M* (10 to 800) for each iteration. The second phase measured *throughput* against *available resources*; experiments ran for *M=10*, and increasing values of *N* for each iteration (100 to 1000).

V. Significant Results and Discussion

Fig. 2 shows the comparative performance on a simulated heterogeneous system, with the *x* axis showing the effect of increased system diversity on performance:

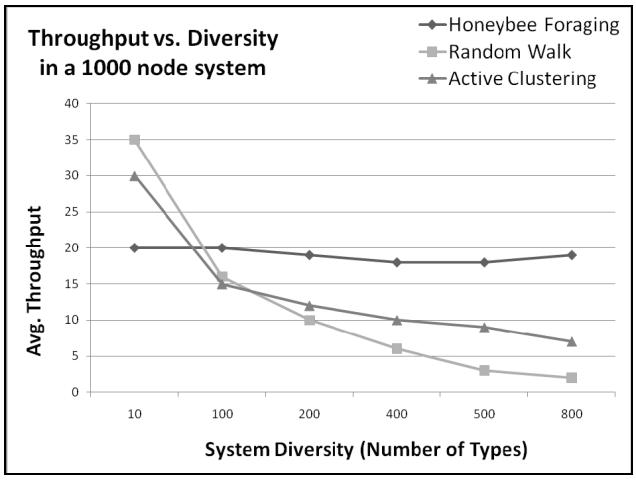

Figure 2. Throughput vs. Diversity in a fixed size (1000 node) system

This graph demonstrates that the honeybee algorithm performs consistently well as system diversity increases. However, despite performing better with high resources and low diversity, both the random sampling walk and active clustering *degrade* as system diversity increases.

Fig. 3 shows the comparative performance on a simulated system where this time the *x* axis increase shows the effect of increased system size (resource availability) on system performance:

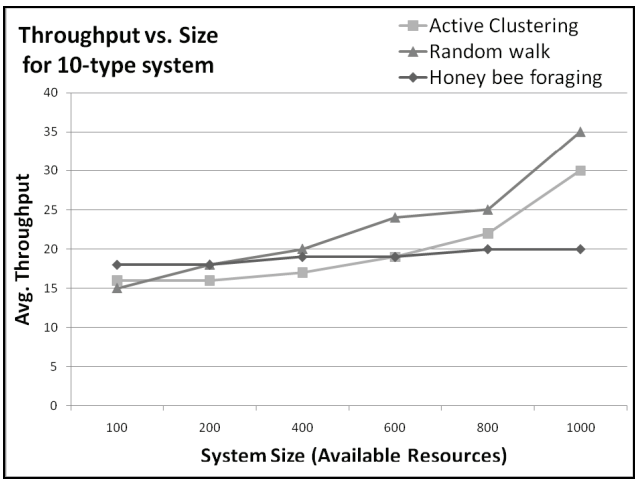

Figure 3. Throughput vs. Size for a fixed-diversity (10 type) system

This graph demonstrates that the honeybee algorithm again performs consistently, but does not increase throughput in line with system size. However, the other approaches are able to utilise the increased system resources more effectively to increase throughput.

The results indicate that the honeybee-based load-balancing approach gives better performance when a diverse population of service types is required. Secondly, results indicate that the random sampling walk performs better in conforming, similar populations, and quickly degrades as the population diversity increases.

The results surrounding active clustering suggest it copes marginally better than random walk with population diversity; though it is affected by the initially created topology. This set of experiments did not set out to evaluate the potential benefit surrounding the application of a number of techniques simultaneously or sequentially. Equally, it is noted that none of the load balancing methods are mutually exclusive; as such, combinations could be used to further improve the algorithms' performance. For example, in the case of a heavily-heterogeneous resource pool (e.g. many service types) it would seem that the inherent diversity causes the Random Sampling approach to perform particularly badly. Application of the Active Clustering rewiring process to create a clustered virtual graph may prove a useful pre-processing step to the Random Sampling walk.

VI. Conclusions and Further Work

This paper presented a comparative study of three distributed load-balancing algorithms for Cloud computing scenarios. It was noted that current commercial offerings based on centralised allocation will cease to be scalable as demand outstrips attempts at apportioning load across all

nodes in a system. Therefore a decentralised approach is required whereby load balancing emerges as a global outcome from local node interactions. As such, three representative algorithms were chosen for this comparative study. The first was directly based on a naturally occurring phenomenon, honey bee foraging, the second sought to engineer a desired global outcome from biased random sampling, while the third used system rewiring (termed Active Clustering) to improve the load balancing performance. Active Clustering was introduced and evaluated in [17], [19] and [20]; in this work the aim has been to reproduce the conditions in a new comparison-based simulation using a parameter from [20]. When resource scale and system diversity are considered, new issues arise.

Active Clustering and Random Sampling Walk predictably perform better as the number of processing nodes is increased; although the latter shows significant variation in performance as the diversity alters. Conversely, increasing resources does not cause an equivalent throughput improvement on the Honeybee Foraging approach. The strength of this approach is shown in a comparative increase in throughput performance when the variety of nodes (more types) is increased. This is the scenario in which both Active Clustering and the Sampling walk are shown to perform comparatively poorly. However, the deliberately-rewired balancing graph of the Active Clustering technique may be more suitable for a second-pass Random Sampling owing to its clustering of like types, resulting in less "wasted" walks, which traverse fewer *appropriately-typed* server nodes.

As such, future work is required to study further the interplay and trade-offs in these large-scale complex systems, and the ways in which it can be exploited. There appears to be a variation in the best algorithms to use depending on the composition and topology of the server network. Virtualisation is considered a crucial component of Cloud Computing; though the management of underlying systems responsible for provision has yet not been thoroughly investigated at this scale or diversity. It is not known how to select appropriate balancing techniques for given applications that will provide a suitable configuration for the application – and provide it in a timely manner. The combination of algorithms is crucial to this process and further work must include sensing of trade-off factors (e.g. system topology or strength of diversity), automated switching and ordered composition of load balancing solutions.

VII. REFERENCES

[1] C.Schroth and T. Janner, (2007). Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. IEEE IT Professional Vol.9, No.3 (pp.36-41), June 2007.

[2] P.A Laplante, Zhang Jia and J.Voas, "What's in a Name? Distinguishing between SaaS and SOA," *IT Professional* , vol.10, no.3, pp.46-50, May-June 2008

[3] W.M.P. van der Aalst, Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 18(1):72-76, 2003.

[4] R. Ruggaber, Internet of Services SAP Research Vision. In 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007), pp: 3.

[5] G. Briscoe, P. De Wilde (2008) Digital Ecosystems: Optimisation by a Distributed Intelligence. In Proceedings of the 2nd IEEE International Conference on Digital Ecosystems and Technologies, Phitsanulok, Thailand.

[6] R. Sterritt (2005) Autonomic Computing. Innovations in Systems and Software Engineering, 1(1), pp: 79–88.

[7] Martin Randles, A. Taleb-Bendiab and David Lamb, Scalable Self-Governance Using Service Communities as Ambients. In Proceedings of the IEEE Workshop on Software and Services Maintenance and Management (SSMM 2009) within the 4th IEEE Congress on Services, IEEE SERVICES-I 2009 - July 6-10, Los Angeles, CA (To appear).

[8] IBM. IBM Introduces Ready to Use Cloud Computing. IBM Press Release, 15th November 2007.
http://www-03.ibm.com/press/us/en/pressrelease/22613.wss

[9] R.L. Grossman, "The Case for Cloud Computing," *IT Professional*, vol.11, no.2, pp.23-27, March-April 2009

[10] Amazon Elastic Compute Cloud (EC2),
http://www.amazon.com/gp/browse.html?node=201590011

[11] P. Miseldine and A.Taleb-Bendiab, A Programmatic Approach to Applying Sympathetic and Parasympathetic Autonomic Systems to Software Design, in Self-Organisation and Autonomic Informatics (1) (Ed: H. Czap et al) pp: 293-303, IOS Press, Amsterdam, 2005.

[12] A. Azeez, Auto-Scaling Axis2 Web services on Amazon EC2. ApacheCon Europe 2009, Amsterdam, March 2009.

[13] Martin Randles, A. Taleb-Bendiab and David Lamb, Cross Layer Dynamics in Self-Organising Service Oriented Architectures. IWSOS, Lecture Notes in Computer Science, 5343, pp. 293-298, Springer, 2008.

[14] S. Nakrani and C. Tovey, On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers. Adaptive Behavior 12, pp: 223-240 (2004).

[15] O. Abu- Rahmeh, P. Johnson and A. Taleb-Bendiab, A Dynamic Biased Random Sampling Scheme for Scalable and Reliable Grid Networks, INFOCOMP - Journal of Computer Science, ISSN 1807-4545, 2008, VOL.7, N.4, December, 2008, pp. 01-10.

[16] P. Erdős and A. Rényi, On Random Graphs, Publicationes Mathematicae Vol 6 pp: 290-297, 1959

[17] F. Saffre, R. Tateson, J. Halloy, M. Shackleton and J.L. Deneubourg, Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications. The Computer Journal, March 31st, 2008.

[18] G. Cybenko, Dynamic Load Balancing for Distributed Memory Multiprocessors. Journal of Parallel and Distributed Computing Vol. 7(2), pp: 279-301, 1989.

[19] E. Di Nitto, D.J. Dubois and R. Mirandola, Self-Aggregation Algorithms for Autonomic Systems. In Proceedings of the 2nd Internation Conference on Bio-Inspired Models of Network, Information and Computing Systems, 2007 (Bionetics 2007), pp.120-128, 10-12 Dec. 2007

[20] E. Di Nitto, D.J. Dubois, R. Mirandola, F. Saffre and R. Tateson, Applying Self-Aggregation to Load Balancing: Experimental Results. In Proceedings of the 3rd international Conference on Bio-inspired Models of Network, information and Computing Systems (Bionetics 2008), Article 14, 25 – 28 November, 2008.

[21] Repast Organization for Architecture and Development, http://repast.sourceforge.net, 2003 (Accessed 10th June 2009).