Self-Organizing Load Balancing in Distributed Collaborative Cloud Applications

Bogdan Solomon
NCCT Lab, University of Ottawa
161 Louis Pasteur
Ottawa, Ontario, Canada
bsolomon@ncct.uottawa.ca

Dan Ionescu
NCCT Lab, University of Ottawa
161 Louis Pasteur
Ottawa, Ontario, Canada
dan@ncct.uottawa.ca

Crsitian Gadea
NCCT Lab, University of Ottawa
161 Louis Pasteur
Ottawa, Ontario, Canada
cris@ncct.uottawa.ca

ABSTRACT

In the past decade, cloud computing has become an integral technology both for the day to day running of corporations, as well as in everyday life as more services are offered running on public or private clouds or grids. At the same time, online collaboration tools are becoming more important as both businesses and individuals need to share information and collaborate with other entities. Previous work has presented an architecture for a collaborative online application which allows users in different locations to share various content including videos, images and documents while at the same time communicating with each other via video chat. The application's servers are deployed in a cloud environment which can scale up and down based on demand. Furthermore, the design allows the application to be deployed on multiple clouds which are deployed in different geographic locations. One area which the previous work left open was how the load balancing is achieved in an intelligent manner. In this paper a self-organizing system based on ant colony path finding and ant colony relocation is proposed in order to achieve such an intelligent load balancing and self-optimization system.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Theory, Design, Self-organization

1. INTRODUCTION

Cloud computing and grids have become an integral technology as more and more services make use of the capability to both use hardware resources as a service and to scale a system from a small number of users to millions of users easily. Companies can now make use of outside resources and simply rent the hardware as needed from cloud providers like Amazon, Rackspace, and Microsoft or build their own internal clouds/grids in order to offer services internally or run business intelligence. In many cases the same infrastructure is shared by multiple departments in the same company, each with its own uses and demands. In the past, scaling an application after release would take weeks as new servers would need to be purchased, configured, tested and finally set to production. With the advent of cloud computing, a new image of a server can be created and be ready to be used in seconds or minutes. This helps in both scaling a service up and down based on demand or deploying new versions of applications in parallel with the old for testing.

As businesses have expanded and have offices all across the world or do business with companies from other parts of the world technology to offer communication and sharing capabilities have become very important. Collaborative applications are important not just for businesses but also for normal persons who wish to communicate with friends or family who are far. Such applications must also be able to be deployed on a cloud and scale up and down based on user demands. In the past we have proposed such a collaborative application [12], [10] as well as a way to self-optimize the servers [11]. However, previous work only presented a simple approach to self-optimize the cloud and group of clouds.

At the same time, self-organizing systems have seen a large amount of research because of the intrinsic properties which they poses:

1. Adaptable - the ability to deal with changes in the environment which were not predicted at design time
2. Resilient - parts of the system can die or be lost but the remaining still perform their goal
3. Emergent - the complex behaviour arises from the properties and behaviour of the simple parts
4. Anticipation - the system can anticipate problems and solve them before they impact the whole system

Because of the above properties a self-organizing system would be perfectly suited to solve the problem of load balancing the servers for the collaborative application. Because the size of the cloud is not determined and can scale up and down, and because at peak demands a large number of servers are up, a self-organizing system can better cope with the self-optimization than a top-down control system.

A number of biological systems have been researched in order to build self-organizing systems, from ant colonies and bee hives [2], [6] to glowworms [4] and bats [7].

DCC '16, July 25, 2016, Chicago, Illinois, USA

In this paper, the proposed self-organizing system makes use of two ant self-organizing approaches in order to obtain the load balancing of the servers. The first is the ant colony route finding self-optimization algorithm. This is combined with the ant colony nest relocation self-optimizing algorithm to achieve the desired self-optimization of the cloud application.

The rest of the paper is organized as follows. Section 2 introduces the problem to be optimized. Section 3 presents the self-organizing approach and section 4 presents some conclusions and future work.

2. COLLABORATIVE APPLICATION

In previous work [12], [10] and [11] we have presented a collaboration application which can be used by users in different locations to share media like video, images and documents in a synchronized manner while also communicating through video and text chat. The application was deployed initially in a single cloud, where servers can scale up and down based on user demand. The application was then extended to be deployed on multiple clouds where users would connect to the best cloud in terms of QoS but were still able to communicate with users connected to other clouds.

Each of the servers was extended to have with its own control loop for self-optimization such that when demand became too high the server stopped receiving connections. The control loop is developed using principles from flow-based models with a leaky bucket in which incoming requests are viewed as an incoming flow of water into the bucket and served requests are viewed as water leaking out of the bucket.

A fluid model or a stochastic fluid flow model is a mathematical model used to describe the fluid level in a reservoir subject to randomly determined periods of filling and emptying. Fluid flow models are widely used in the performance evaluation of computing processes, and in the transmission of packets in high-speed communication networks. The fluid model is an idealized deterministic model. In a communication network a unit of "fluid" corresponds to some quantities of packets. A fluid model is often a starting point to build a model for the impact of topology, processing rates, and external arrivals on network behavior.

In a computer environment the fluid represents the tasks stirred in a buffer of tasks; the tasks wait to be entered into execution and terminated, though in this flow the tasks might be interrupted and brought back into execution. Those latter tasks are called to be re-entrant. The arrival and service processes are modulated by a random external environment, and the quantity of interest is the behavior of the occupied size of the buffer.

The arrival and service processes are modulated by a random external environment, and the quantity of interest is the behavior of the occupied size of the buffer. Various aspects of the fluid flow model were thoroughly investigated in the scientific literature such as [1], [5], [13], [3], [8]. The above references, considered the case where the random external environment is a Continuous-Time Markov Chain (CTMC) with a finite or infinite state space.

The literature about modeling communications of computer processing based on fluid flow models for queuing processes abounds in a variety of models which some are better than others. However, all the models have as the starting point the modeling of processes of level control.

A level control process in a tank is fueled by one or many flows flowing through a number of pipes, while the liquid/fluid is evacuated from the tank via another number of pipes through which the fluid flows with different speeds/quantities as in Figure 1. This model is considered in what follows. The tank level control is modeled by the equation 1.

$$\begin{aligned} A\tfrac{dh}{dt} &= q_{in} - q_{out} \\ q_{out} &= a\sqrt{2gh} \end{aligned} \tag{1}$$

where: $h \in \mathbb{R}$ is the level in the tank, $g \in \mathbb{R}$ is the acceleration of gravity, $A \in \mathbb{R}$ is the area of the tank, $q_{in} \in \mathbb{R}$ is the flow of fluid pumped into the tank, $q_{out} \in \mathbb{R}$ is the flow of fluid pumped out of the tank, and $a \in \mathbb{R}$ is the area of the of the hole in the tank through which the fluid is pumped out of the tank.

$$\begin{aligned} \tfrac{dh}{dt} + k_1\sqrt{h} &= k_2 q_{in} \\ q_{out} &= k_3\sqrt{h} \end{aligned} \tag{2}$$

where:

$$\begin{aligned} k_1 &= \tfrac{a\sqrt{2g}}{A} \\ k_2 &= \tfrac{1}{A} \\ k_3 &= \sqrt{2g}a \end{aligned} \tag{3}$$

where $k_1, k_2, k_3 \in \mathbb{R}$ are constant parameters expressed in the physical units of MKS.

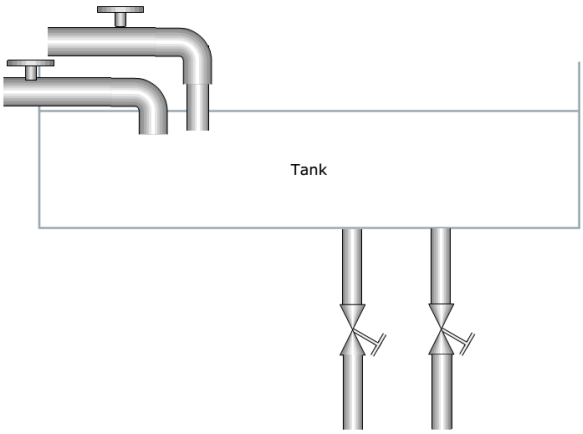

Figure 1: Level/Fluid Flow Model

In order to achieve the global self-optimization of the system, previous work presented a simple voting system where each of the servers votes on whether to add or remove servers based on its own local knowledge of the system. A server which does not accept requests votes to have new servers added, while an under-loaded server votes to have servers removed from the cloud. This approach exhibits some self-organizing behaviour however there is still a need for a centralized vote counter which makes the decision of adding or removing servers. At the same time, the voting approach requires a threshold of votes to be passed in order to add/remove servers and results in a system with low emergence possibilities.

In this paper, we propose to replace the voting system with a self-organizing system based on two ant colony optimization algorithms which will be presented in the next section.

3. ANT COLONY SELF-OPTIMIZING SYSTEM

Ant colony food finding is a well know self-organization algorithm used to find best routes in a network [2] or solve load balancing problems [9]. The ant colony relocation algorithm is another self-organization algorithm used to find a new solution once the previous state of the system is no longer valid []. In this paper we proposed to combine the two ant colony self-organization algorithms to achieve the self-optimization of the cloud of collaborative applications. The ant colony food finding algorithm is used in order to continuously monitor the state of the cloud and determine when an SLA breach is in danger of happening. The ant colony relocation algorithm is then used in order to find a new optimum for the cloud of servers.

3.1 Ant colony food finding

The ant colony food finding algorithm [2], [9] can be used in order to load balance clouds or find the best route in a network. At a high level the algorithm uses a number of simple agents called ants which traverse the network and leave a trail of pheromones which other ants can then follow. Good paths or solutions are reinforced by having more ants traverse them and leaving more pheromones. Once a path or solution is no longer good, less ants travel it and another path is reinforced as more ants travel that path.

Assume that the leaky token bucket adds tokens for each request received and removes tokens at a predefined rate. Thus an empty token bucket represents a server with no load and a full bucket represents an overloaded server.

For the collaborative application cloud, whenever a server starts it creates a new ant and sends it through the network of servers. If the server is the first server and has no knowledge of other servers it sends no ant. Thus the total number of ants is equal to $N-1$ where N is the number of servers. Because the new server joins the cloud of servers and receives information about the other servers in the cloud randomly it will send the ant to the first server which replies to it.

As ants reach other servers they deposit pheromone at the server they arrive at a rate inversely proportional to the load of the server. At the same time ants wait at the server a time proportional to the load of the server. Thus overloaded servers will have less pheromone deposited when compared to underloaded server. By having ants wait a longer time at overloaded servers, the overall amount of pheromone in the network will further decreases.

Assume that an ant k deposits an amount of pheromone τ_k when it reaches a node which has 0 load - represented by an empty leaky token bucket, and waits at the node $15s$. An ant k which reaches a node where the leaky bucket is 50% full will deposit only $\tau_k * (1-p)$ where p is the bucket fullness as a percentage and wait at the node a time of $15s/(1-p)$ with a maximum wait time of $60s$ to avoid waiting an infinite time when p approaches 100%. At the same time, the pheromone left by the ants decays at a rate of ρ every $15s$. As such, the amount of pheromone at any node can be seen as:

$$p_n^t = p_n^{t-1} + \sum_{i=1}^{K}(\tau * (1-p)) - \rho \quad (4)$$

where p_n^t is the amount of pheromone at node n at time t where t can be considered discrete in $15s$ increments and K is the amount of ants arriving at the node in the time frame between $t-1$ and t.

Server	Time since last visit (s)
Server 1	15
Server 2	20
Server 3	5
Server 4	35
Server 5	0

Table 1: Ant routing knowledge prior

Server	Probability (%)
Server 1	21.43
Server 2	28.57
Server 4	50
Server 5	0

Table 2: Ant routing probability

Ants store information about which servers they have visited and time passed since the last visit. When an ant decides which server to go to, it uses a random function which is proportional to the time since it has not visited a server. Thus the ant will give preference to the servers it has not visited in a long time, and especially the servers it has never visited. Because the server structure is not stable and servers can join and leave at any time, ants decide the next server to visit based on the servers known by the current server the ant is at. Assume a cloud with 5 servers and an ant which has the following information in it's visit history table.

Based on the table, the ant computes the probability of visiting each server as

$$P_s = t_s / \sum_{i=1}^{N} t_i \quad (5)$$

where P_s is the probability of visiting server s and t_s is the time since it has visited server s last time. These probabilities are computed only on the servers known as being up by the server the ant is at. Since the server the ant is at has a time since last visit of 0, it does not need to be excluded from the calculations. Let us assume also that Server 5, which is the server the ant is at currently does not yet know about server 3. As such, the visit probability table looks as follows:

As such, the ant will roll a random value between 0 and 1, and choose which server to go to. A value between 0 and 0.5 means Server 4, between 0.5 and 0.78 means Server 2 and between 0.78 and 1 means Server 1. Assuming the value the ant rolls is 0.8, and that the ant waits at Server 5 for 5s, the routing table after the jump will be:

In order to avoid having all ants visit a new node all at once, whenever an ant discovers a new server it initializes the time since it visited with a random value. Assume that after the ant reaches Server 1, it discovers a new server which was unknown before - Server 6. The time since last visiting Server 6 will be initialized with a random value between 0 and the maximum time since last visiting a server which is known to be alive as in Equation 6. If Server 1 only knows Server 2, 5 and 6 then the random value will be between 0

Server	Time since last visit (s)
Server 1	0
Server 2	25
Server 3	10
Server 4	40
Server 5	5

Table 3: Ant routing knowledge posterior

and 25s.

$$t_{new} = random(0, max(t_{known})) \quad (6)$$

This algorithm is used in order to decide when the system is about to breach its SLAs. Once it appears that the system's SLA is about to be breached the ant relocation algorithm kicks in in order to find how many servers should be started. As such, each ant is also responsible for measuring the pheromone level at each node it visits. Ants store a history of the last N nodes that they visited and the pheronomone level at each of these nodes. Every time an ant moves to a new server, it removes the oldest pheronomone value from this list and adds the pheronomone at the current node. It then computes an aggregate metric of the pheronomone at the last N nodes visited, which is a simple average:

$$p_{ant} = \sum_{i=1}^{N} p_N/N \quad (7)$$

where p_N is the pheromone at server N. Once the pheromone level of an ant goes under a predefined value, the ant morphs into an ant relocator which searches for the new count of servers which should be added to the cloud. Similarly, if the pheromone level of an ant becomes too high, the ant morphs and starts searching for the number of servers which should be removed from the cloud.

3.2 Ant colony relocation

The ant colony relocation algorithm can be used in order to optimize the number of servers in the cloud, by having ants search for the optimal number of servers which should exist in the cloud. This algorithm works by having each ant search for a new nest, where a nest is represented by a new optimum number of servers in the cloud. Once more than half of the ants have reached the same optimum value, the servers are added or removed as desired and all ants are morphed back into food foraging ants.

When an ant morphs into an ant relocator it initializes itself with a range of possible solutions for the new size of the cloud. These possible solutions are computed as permutations of the current size of the cloud as known by the server the ant is at. If the ant was morphed because of lack of pheromone, then the ant generates a random value which would represent the number of servers to be added. Each random value is proportional to the size of the cloud and the pheromone level across the last N servers. For example, if the cloud contains only 2 servers, then the ant is initialized with a random value between 1 and 2. However, if the cloud contains 100 servers, then the ant would be initialized with a random values, which is defined as:

$$ServerCount_{add} = 1 + rand * N/(p_{ant}) \quad (8)$$

such that the lower the pheronomone value, the higher the probability of more servers being added. The random value has the effect of generating multiple possible solutions across different ants.

If however the ant was morphed because of too much pheromone, then it does a similar initialization, but in this case it randomly chooses X servers and hides them from its knowledge of the cloud. The servers are still in the ants history but it does not consider them when computing the solution fitness. Each ant will create a different such permutation of servers which are hidden.

The ant relocation algorithm is performed in rounds. In the first round all relocator ants search for a new nest by examining their solution space. This is achieved by having each ant simulate the effect of its solution on the pheromone level. If the ant is a server adder then the ant simulates that the $ServerCount_{add}$ servers are up and with no load. It picks K servers from the list of the last N servers it has visited and hides them, and replaces them with servers with no load. It then uses this information in order to simulate the pheromone level. If the ant is a subtractor, it hides K servers where K is defined as part of its random solution and simulates the effect this would have on its pheromone level.

In the second round ants with a solution with poor fitness give up on their 'nests' but wait a single round at the rejected nest before going back home. Ants with solutions with good fit criteria come back to the home 'nest' and recruit other ants. The recruitment process is completely random, with an ant either recruiting or being recruited. Ants then go together to the nest of the recruiting ant. While two ants can go together to the same 'nest' and simulate the same number of servers, the hidden servers will be different so the ants will have different pheronome levels.

In the third round ants which have given up on their nests go back home where they wait to be recruited by other ants. The other ants do tandem runs, where two ants go together to a recruiting ant's nest. The ants arrive at the nest and compare the amount of ants at the nest in the previous round with now. If the number of ants at the nest has decreased then the nest is abandoned, the ants wait a round at it and then go back home to be recruited.

The process in rounds two and three is repeated until a single candidate nest remains. Such a nest is represented by a nest to which more than half of the ants are committed.

One thing to note is that not all ants will morph at the same time. As ants discover that the environment is not meeting the SLA anymore, due to either too much load represented by insufficient pheromone levels or too many servers represented by too much pheronomone in the system they will morph. As such, a single ant or a small subset of ants will not cause the entire system to modify itself. If morphing ants pass a number of rounds without having found a new best nest, then they give up and morph back into food finding ants, as the majority of the ants still finds the current environment to be stable.

3.3 Advantages of proposed approach

There are a number of advantages which the proposed approach offers. first of all because it takes time for ants to find that the system is over or under loaded, there is no need to implement a dead time period in which no control decisions are taken. Because of this the system can not

exhibit oscillations.

Second of all, the ants act in a completely distributed manor and share no knowledge between them, thus the autonomic system is extremely resilient to failures in the infrastructure. The worst case that can happen is that a server crashes while hosting a large number of ants, thus causing the pheromone level to drop because of a lack of ants in the system. To avoid such issues from happening, a server receives periodically pings from the ant it has created. If too long a time passes without any pings, the server creates a new ant and sends it in the system. The pinging of messages from ants to servers goes both ways - a server sends a reply to an ant upon reception of a ping, and it also used to allow ants to die when their respective server crashes or is shut down. If a network failure happens and upon restoration a server notices pings from more than one ant, then it randomly kills all but one of its ants.

It is possible in this scenario for an ant to send a ping message to its server and then move to another server before receiving the reply from its server. Because of this, an ant will only die if two conditions are met - it has not received a ping back from its server and the last N servers it has traversed had no knowledge of the ant's server.

4. CONCLUSIONS AND FUTURE WORK

This paper introduced a novel approach for the autonomic control of a cloud based application responsible for providing user collaboration tools, which uses principles from network control and combines them with self-organizing mechanisms in order to ensure the self-optimizing function for the cloud due to changing demand. The self-organizing approach is based on two ant based self-organizing algorithms which are combined in order to achieve the desired results.

Future work will focus on providing a full implementation of the self-organizing system described in this paper as well as using various load simulations in order to provide performance results for the system.

5. REFERENCES

[1] D. Anick, D. Mitra, and M. M. Sondhi. Stochastic theory of a data handling system with multiple sources. In *ICC'80; International Conference on Communications, Seattle*, volume 1, 1980.

[2] C.-H. Chu, J. Gu, and Q. Gu. A heuristic ant algorithm for solving qos multicast routing problem. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1630–1635, Washington, DC, USA, 2002. IEEE Computer Society.

[3] V. G. Kulkarni. Frontiers in queueing. In J. H. Dshalalow, editor, *Frontiers in queueing*, chapter Fluid models for single buffer systems, pages 321–338. CRC Press, Inc., Boca Raton, FL, USA, 1997.

[4] X. Lu and W. Sun. An improved self-adapting glowworm swarm optimization algorithm. In *Signal Processing, Communication and Computing (ICSPCC), 2013 IEEE International Conference on*, pages 1–4, Aug 2013.

[5] D. Mitra. Stochastic theory of a fluid model of producers and consumers coupled by a buffer. *Advances in Applied Probability*, 20(3):pp. 646–676, 1988.

[6] P. Navrat, T. Jelinek, and L. Jastrzembska. Bee hive at work: A problem solving, optimizing mechanism. In *Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 122–127, Dec 2009.

[7] T. Niknam, F. Bavafa, and R. Azizipanah-Abarghooee. New self-adaptive bat-inspired algorithm for unit commitment problem. *IET Science, Measurement Technology*, 8(6):505–517, 2014.

[8] B. Sericola and B. Tuffin. A fluid queue driven by a markovian queue. *Queueing Systems*, 31(3-4):253–264, 1999.

[9] K. M. Sim and W. H. Sun. Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 33(5):560–572, Sept 2003.

[10] B. Solomon, D. Ionescu, C. Gadea, and M. Litoiu. *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, chapter Geographically Distributed Cloud-Based Collaborative Application. IGI Global, 2013.

[11] B. Solomon, D. Ionescu, C. Gadea, S. Veres, and M. Litoiu. Self-optimizing autonomic control of geographically distributed collaboration applications. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, pages 28:1–28:8, New York, NY, USA, 2013. ACM.

[12] B. Solomon, D. Ionescu, C. Gadea, S. Veres, M. Litoiu, and J. Ng. Distributed clouds for collaborative applications. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 218 –225, may 2012.

[13] T. E. Stern and A. I. Elwalid. Analysis of separable markov-modulated rate models for information-handling systems. *Advances in Applied Probability*, 23(1):pp. 105–139, 1991.