Distributed Agent-Based Load Balancer for Cloud Computing

Sliwko, L., Getov, V. and Bolotov, A.

DISTRIBUTED AGENT-BASED LOAD BALANCER FOR CLOUD COMPUTING

Leszek Sliwko, Vladimir Getov, Alexander Bolotov
University of Westminster, 101 New Cavendish Street, London W1W 6XH
w1518355@my.westminster.ac.uk, V.S.Getov@westminster.ac.uk, A.Bolotov@westminster.ac.uk

Abstract. In this paper we present a concept of an agent-based strategy to allocate services on a Cloud system without overloading nodes and maintaining the system stability with minimum cost. To provide a base for our research we specify an abstract model of cloud resources utilization, including multiple types of resources as well as considerations for the service migration costs. We also present an early version of simulation environment based of workload traces from Google Cluster Data project and a prototype of agent-based load balancer implemented in functional language Scala and Akka framework.

Introduction. Modern day applications are often designed in such a way that they can simultaneously use resources from different computer environments. System components are not just properties of individual machines and in many respects they can be viewed as though they are deployed in a single application environment. Distributed computing differs from traditional computing in many ways. The sheer physical size of the system itself means that thousands of machines may be involved, millions of users may be served and billions of API calls or other requests may need to be processed. In recent years the most advanced technologies offer cloud solutions. A cloud system connects multiple individual servers and maintains the communication between them in order to process related tasks in several environments at the same time. Clouds are typically more cost-effective than single computers of comparable speed and usually enable applications to have higher availability than a single machine. This makes the software even more attractive as a service and is shaping the way software is built today. Moreover, companies no longer need to be concerned about maintaining a huge infrastructure of thousands of servers just so they have enough computing power for those critical hours when their service is in highest demand. Instead companies can simply rent thousands of servers for a few hours.

Cloud systems. As of today, the biggest cloud systems offering elastic resource allocation are Amazon Web Services, Google App Engine, Microsoft Azure, Rackspace, Digital Ocean and GoGrid. A few well-known examples of services backed up by cloud computing are Dropbox, Gmail, Facebook, Youtube and Rapidshare. This elasticity of resources, without paying a premium for a large-scale usage, is unprecedented in the history of IT (Fox et al, 2009). However, it introduces a new set of challenges and problems, which need to be solved. The cloud systems are usually made up of machines with very different hardware configurations and different capabilities. These systems can be rapidly provisioned as per the user's requirements thus resource sharing is a necessity. Resource management has been an active research area for a considerable period of time and those systems usually feature specialized load balancing strategies.

Focus on stability. Currently existing distributed load balancers (mainly from Cluster environments, e.g.: Maui (Etsion and Tsafrir, 2005), Moab (Young, 2014), LoadLeveler (Kannan, 2001), Globus (Foster, 2005), Mesos (Hindman et al., 2011), Torque (TORQUE Resource Manager, 2014b), etc.) primary focus on features like: performance, responsiveness, locality, fairness, etc. However, the main purpose of commercial Cloud systems is to keep third party operations working continuously and with a minimal disturbance (i.e.: ‘stability’). Therefore, our resources utilization model focuses on maintaining system stability with minimum cost. We assume in the majority of problem instances, the system already has capacity to process all current jobs, although the system might detect that existing resources are insufficient. The main challenge is to allocate those jobs properly, so no single node is overloaded. Our research will focus mainly on providing system stability combined with optimal minimal cost. Other features, such as fairness and performance will be considered to be a secondary objective.

Resource utilization model. Our model consists of nodes and services where the load balancer task is to keep a good load balance through resource vector comparisons. In considering what is actually constituted as a ‘service’ in a Cloud environment an example may be seen in a popular Cloud environment such as Amazon’s EC2, where applications are deployed within the full Operating System Virtual Machine (VM). Depending on design, service might come also with preinstalled local database such as MySQL. One might argue the effectiveness of this approach, however this schema has many benefits such as the almost complete separation of execution contexts (although services might still share the same hardware if they are deployed on the same node) and complete control over local system environment parameters. Amazon EC2 uses the templates in Amazon Virtual Image format (Beach, 2014). Currently there are more than 20000 images to select from.

Problem formulation. Let us define $\Lambda = (\tau, \eta, \psi, a, r, c)$ as a problem space and system as a twice (Λ, μ). In the ***d-resource system optimization problem***, we receive a set τ of l mobile services $\tau = \{t_1, t_2, ..., t_l\}$ and a set η of m fixed nodes $\eta = \{n_1, n_2, ..., n_m\}$. We call $\mu : \tau \to \eta$ as a *service assignment* function, where each service has to be assigned to the node.

We also consider:

- $\psi = \{i_1, i_2, \ldots, i_d\}$ as a set of all different kinds of resources. To illustrate, for $d = 3$ we could define $\psi = \{CPU, memory, network\}$.
- $a : \psi \times \eta \to \mathrm{N} \cup \{0\}$ as a fixed *available resources* on the nodes. $a_i(n)$ is the available level (integer value) of a resource i on the node n.
- $r : \psi \times \tau \to \mathrm{N} \cup \{0\}$ as a fixed *required resources* for services. $r_i(t)$ is the required level (integer value) of a resource i of service t.
- $c : \tau \to \mathrm{N} \cup \{0\}$ as a *service migration cost* function. $c(t)$ means cost incurred migrating service executables and its state and preparing service environment.

For every node $n \in \eta$ we define a set $A_n = \{t \in \tau : \mu(t) = n\}$ of all services assigned to the node n. We also define $f : \psi \times \eta \to \mathrm{N} \cup \{0\}$ as *remaining resources* on the nodes: $f_i(n) = a_i(n) - \sum_{t \in A_n} r_i(t)$.

We consider system (Λ, μ) as *stable*, if:

$$f_i(n) \geq 0, \text{ i.e.: } \sum_{t \in A_n} r_i(t) \leq a_i(n), \text{ for every } n \in \eta,\ i \in \psi \qquad (1)$$

Otherwise the system (Λ, μ) is *overloaded*.

Each service t is initially assigned by *service assignment* function μ_0 to some node η. During the *system transformation* $(\mu_0 \to \mu_1)$ service $t \in \tau$ can be reassigned to any different node $n \in \eta$. The process of moving the service to a different node is referred to as *service migration* and this feature generates a *service reassigning cost*:

$$c_{(\mu_0 \to \mu_1)}(t) = \begin{cases} 0, & \mu_0(t) = \mu_1(t) \\ c(t), & \mu_0(t) \neq \mu_1(t) \end{cases}$$

Every *system transformation* process $(\mu_0 \to \mu_1)$ has its *system transformation cost*:

$$c_{(\mu_0 \to \mu_1)} = \sum_{t \in \tau} c_{(\mu_0 \to \mu_1)}(t) \qquad (2)$$

Consider initial *service assignment* μ_0; *service assignment* μ^* is optimal for μ_0, if μ^* renders system (Λ, μ^*) *stable* and:

$$c_{(\mu_0 \to \mu^*)} \leq c_{(\mu_0 \to \mu)}, \text{ for every } \textit{stable} \text{ system } (\Lambda, \mu).$$

N.b.: when (Λ, μ_0) is *stable* for initial *service assignment* μ_0, the *system transformation cost* equals 0 as it is considered optimal.

Agent-based load balancer. In order to correctly assign new services to nodes as well as to provide adequate execution environments to existing ones, we have designed a load-balancing system, where autonomous agents continuously negotiate between themselves the placement of tasks.
Every node is represented by an agent. An agent has several tasks:

- Periodically update central database about currently available and utilized resources on his node
- Accept or reject task placement requests from other agents (depending on available resources and scheduled tasks which will run in near future)
- Keep track of all tasks currently being executed or scheduled to be executed (i.e.: task might be currently being downloaded)
- Select and evict tasks, which consume too much resources
- Find alternative node and negotiate with its agent evicted task migration

In current implementation decision algorithms used in every of the above tasks are simplistic. The agent reacts only when the level of any of utilized resources exceeds available levels and selects the most resource-consuming service for eviction. Then agent tries to find alternative node for this service:

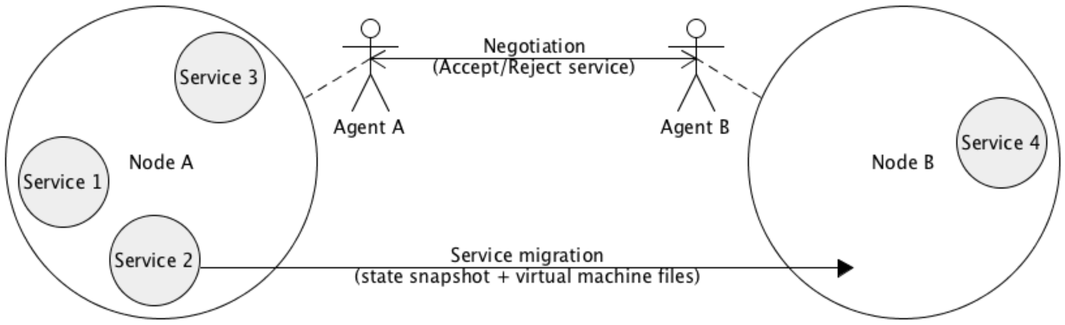

Figure 1: ???

Conclusions. In our current prototype, we have based implementation on Scala with Akka actors-system. Such setup is quite flexible and easy to test on single machine. From early experiments, we have noted that agent-based load balancer system is very unpredictable and might provide a wide spectrum of results. Nevertheless, thanks to distributed nature on agents we avoid 'blocking-head-of-queue' problem, which is prevalent in 'monolitic' schedulers (Schwarzkopf et al., 2013). In future research we plan use workload traces from Google Cluster Data project as realistic input data.

REFERENCES

1. "TORQUE Resource Manager. Administration Guide 5.0.1." Adaptive Computing Enterprises, Inc. November 7, 2014. Retrieved 15 November, 2014. Version 5.0.1.
2. Beach, B. (2014) "Amazon Machine Images" In *Pro Powershell for Amazon Web Services*, pp. 115-134. Apress
3. Etsion, Y. and Dan Tsafrir. (2005) "A short survey of commercial cluster batch schedulers." *School of Computer Science and Engineering, The Hebrew University of Jerusalem* 44221: 2005-13.
4. Foster, I. (2005) "Globus Toolkit Version 4: Software for Service-Oriented Systems." *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779, pp 2-13
5. Fox, A. Rean Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. (2009) "Above the clouds: A Berkeley view of cloud computing." *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28: 13.
6. Hindman, B. Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. (2011) "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." In *NSDI*, vol. 11, pp. 22-22
7. Kannan, S. Mark Roberts, Peter Mayes, Dave Brelsford, and Joseph F. Skovira. (2001) "Workload management with loadleveler." *IBM Redbooks* 2: 2.
8. Schwarzkopf, M. et al., (2013) "Omega: flexible, scalable schedulers for large compute clusters." In *Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 351-364. ACM
9. Young R. "Ten Reasons to Switch from Maui Cluster Scheduler to Moab HPC Suite - Comparison Brief", Adaptive Computing. January 6, 2012 . Retrieved 5 November, 2014.