Distributed Management of Cloud Computing Applications

Michael Roth
Universität Augsburg, Germany
E-Mail: michael.roth@informatik.uni-augsburg.de

I. Introduction

Cloud Computing allows customers to pay for servers as needed. Customers can deploy large web applications without an upfront investment in hardware. Most web applications have a high peak load in daytime and fewer visitors at night. Because of the ability to change the number of running servers in a matter of minutes the number of servers can be reduced during times of low traffic, and it can be increased during peak times. By using Cloud Computing developers can also start a small web application which can easily scale up if more users discover the application. For these reasons Cloud Computing has become very popular over the last years.

Large Cloud Computing applications consist of thousands of servers and are too dynamical to manage them manually. Therefore, an automated way to manage these applications is needed. Organic Computing aims to control complex systems by introducing self-x abilities. Organic Computing systems posses the abilities to self-configure, self-optimise, self-heal and self-protect themselves.

We are developing an Organic Computing Middleware [1] which is capable of managing large Cloud Computing applications on its own by using these self-x properties. Our middleware is completely decentralised and therefore possesses no single-point-of-failure. An Organic Manager is the key component of our middleware. The manager uses a MAPE Cycle [2] to observe and control the system. The planing in our middleware is done by an automated planer [3]. We investigate the monitor and analyse phases in this paper.

II. Scenario

In our scenario a company develops a new web application and uses Cloud Computing to run the application. The company rents virtual machines from a Cloud Computing provider. The application requires web servers to process user requests and database servers to store information.

We use Amazon Web Services (AWS) [4] as an example Cloud Computing provider. Other providers have similar services, therefore our research can be used with them as well. Users can rent Virtual Machines (VM) and pay for them by the hour. VMs have different computing capabilities. The more computing power a VM possesses the higher is the price. Currently the prices for one hour range from \$0.02 to \$4.60 for the AWS Data Centre in North Virginia. Additionally, customers are charged for generated network traffic.

The customers have only limited influence on the VM placement. AWS currently operates 8 data centres, called Regions, all over the world. Each Region is divided in two to four Availability Zones. A customer can choose the Region and the Availability Zone for new instances.

Amazon also offers some additional services to manage the cloud application. Amazon's monitoring service CloudWatch [5] is a centralised monitoring system which can monitor instances in one Region. It is not possible to monitor the whole application over multiple Regions. The CloudWatch server collects information from the instances over SNMP. The user can also define other information sources.

With Amazon's Auto Scaling new VMs can be deployed automatically if an observed metric reaches a predefined threshold. It is also possible to stop instances if an observed metric falls below predefined value. The Auto Scaling service uses the data collected from CloudWatch. It is only possible to define thresholds which trigger actions.

Elastic Load Balancing [6] is a service from Amazon which manages user requests and distributes them to different servers. Users send their requests to the load balancer. The load balancer distributes the requests to the web server pool. Therefore it is possible to change the number of web servers without the user noticing. The Amazon load balancer can only handle traffic for one region.

In our work we use an example web application which is used by customers all over the world and is therefore hosted in all 8 AWS regions. To handle the failure of an entire region the monitoring system must observe all VMs in all regions. It is not necessary to know the exact information for each VM in other regions but the overall state must be known. We use the response time of the web server as indicator for the application's health. Users expect an answer in a reasonable time and are not interested on the CPU load or RAM usage of the instances. If the response time is too high the middleware must determine the cause. A high response time can be caused by the web server or by the database server. If only a single web server is slow, succeeding requests can be routed to other web servers by the load balancer until the slow web server recovers. If more web servers have a high response time the middleware must decide if new instances must be launched. To save money the company wants only to rent the number of servers which are actually required to handle the user requests within a given response time interval. If the workload decreases and the user requests are answered with a very low response time the middleware can shut down instances to save money.

The middleware must also detected failed instances and decide whether the VMs must be replaced or shut down. Such failed instances can respond fast to user requests with an error page or a partial HTML page. Therefore not only a low response time must be guaranteed. The middleware must also ensure that the web servers work correctly.

III. Monitoring Phase

Amazon Web Services use a centralised monitoring system like most Cloud Computing providers. In such a system all monitoring data is sent to the central monitoring instance. To avoid a singe-point-of-failure the monitoring instance is often replicated which leads to even more traffic. Amazon CloudWatch is not suitable for our scenario since it can not monitor system in different AWS regions.

Our goal is to develop a decentralized monitoring system for a self-managing Cloud Computing Middleware. The monitoring system must collect enough information to observe the status of the web application. With the monitoring information the middleware can decide if the system must be adjusted. Some instances in the web application monitor the system. These instances are chosen by the middleware. We want to spread these observer instances over the entire network. Each observer is responsible for close-by instances but knows also the approximated status of the entire network. If an observer fails or the system changes, the middleware can change the number of observers or relocate the observer to another VM.

The instances are not aware of the network structure. To enable the instances to send monitoring data to all observers, all components form a distributed hash table (DHT). The DHT allows a structured decentralised forwarding of information for all participants in the middleware. More details on the used DHT based information dissemination algorithms can be found at [7].

To allow the observer instances to receive the required information a publish/subscribe protocol is used. We use the DHT network to forward the subscriptions and information. Neighbours in the DHT network are chosen by their network distance. The node information of different nodes is combined on its way to the observer. Therefore each node in the information path has an partial view an the network. Only the observer receives all information and has therefore a more complete view of the network. Because of the different transit times the observer does not possess a consistent view.

IV. Analyse Phase

Because of the decentralized nature of our middleware and to save network bandwidth we want to analyse the information on its way to the monitoring instances and send only aggregated information. We also want to investigate if it is possible to trigger actions before the information reaches the monitoring instance if the aggregated data shows a fatal problem. This can be done by the nodes forwarding the information to the observer. In this case the observer is only responsible for optimizations that cannot be performed by the other instances. The monitoring system must also observe the actions taken from aggregated information and control if these actions guide the system into a valid state.

A. Fuzzy Logic

One interesting way to analyse the monitored data is Fuzzy Logic. In Fuzzy Logic variables have truth values which can adopt a value between 0 and 1. Member Functions describe the truth values of variables for given input values. Rules are used to map the variables to output variables. The truth values of the input variables and the rules are used to calculate the truth values of the output variables. By using the different truth values of the output variables an output value is calculated.

Usually the input and output variables are named to be easily understood, e.g. *low utilization*, *high temperature*, *moderate latency*, *extreme high throughput*, *start few instances*, *start many instances*. The rules are therefore very easy to understand, e.g. IF *high latency* AND *high throughput* THEN *start few instances*. The idea is that these simple rules can be entered by users without knowing the exact values representing good and bad latencies. The definition of the Member Functions must be done by an expert whereas the rules can be generated by users without specific domain knowledge. Because of this simple rule language fuzzy logic is a good match for our organic computing concept.

B. Time Series Analysis

Most web applications have a peak load every day at the same time and a different load on weekends and holidays. With time series analysis and time series forecasting we want to analyse the previous behaviour to predict such peak times. If possible such analysis should be done by each instance. The instances send recommendations to the monitoring system. These recommendations can be calculated on the source instances and are sent to the monitoring instances. The monitoring instances can trigger actions on behalf of these recommendations.

We want to investigate if it is possible for single instances to calculate reliable forecasts. Each instance knows only the local load information. If the VM has recently been started there is not enough information available. If the local knowledge is not sufficient enough the load information is sent to the monitoring instance. We want to determine how many data is needed to calculate a reliable forecast. The data sent to the monitoring system can be aggregated on its way. A forecast can be calculated if the aggregated data is significant so that only the forecast for a group of instances is sent to the monitoring system. By doing so we can save bandwidth.

V. Evaluation

Because of the size of such web applications we are unable to test our scenario in the real world evaluation. We use the ns3 [8] network simulator to generate a simulated Cloud Computing environment. The work of Barroso and Hölzle [9] are used to model a Cloud Computing data centre. Within this environment we will simulate user interactions. We are looking for real performance information to simulate the behaviour of the VM instances and the visitors. We will induced VM failures and fluctuations in the number of visitors. To judge the efficiency of our middleware we will compare our approach with traditional centralised solutions. The objective of our evaluation is to see if the example web application can be reached with a low response time and at the same time the number of used servers is kept low. To measure the costs we will monitor the used network bandwidth. Also the time required to counteract disturbing influences will be measured.

VI. Summary

We presented a distributed middleware to manage cloud computing application. The middleware uses an organic manager to control the application. The manager uses a MAPE cycle. In this paper we focus on the monitoring and analyse phase.

For information dissemination the instances of the application build a distributed hash table network. Information can be spread throughout the entire network without knowing the network topology or many other active instances by using the DHT network. We use a publish/subscribe protocol over the DHT network to send the node information only to the observer instances. All other nodes collect and forward the information to these observers.

A distributed analyse phase aggregates the information on its way to the observer. If a problem is detected which can be solved with local knowledge actions are taken. The observer must only interact with the system if a bigger problem is detected.

We will evaluate our research with a network simulator. To get accurate results we model the network after current cloud computing centres and use real performance information for modelling the instances behaviour.

References

[1] M. Roth, J. Schmitt, R. Kiefhaber, F. Kluge, and T. Ungerer, "Organic Computing Middleware for Ubiquitous Environments," in *Organic Computing A Paradigm Shift for Complex Systems*, ser. Autonomic Systems, C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds. Springer Basel, 2011, pp. 339–351.

[2] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, pp. 41–50, jan 2003.

[3] J. Schmitt, M. Roth, R. Kiefhaber, F. Kluge, and T. Ungerer, "Using an Automated Planner to Control an Organic Middleware," in *Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on*. IEEE, 2011, pp. 71–78.

[4] Amazon, *Getting started with Amazon Web Services*, 2013 (accessed April 22, 2013). [Online]. Available: https://aws.amazon.com/en/documentation/gettingstarted/

[5] ——, *Amazon CloudWatch Documentation*, 2013 (accessed April 22, 2013). [Online]. Available: https://aws.amazon.com/en/documentation/cloudwatch/

[6] ——, *Amazon Elastic Load Balancing Documentation*, 2013 (accessed April 22, 2013). [Online]. Available: https://aws.amazon.com/en/documentation/elasticloadbalancing/

[7] M. Roth, J. Schmitt, F. Kluge, and T. Ungerer, "Information dissemination in distributed organic computing systems with distributed hash tables," in *Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*. IEEE, 2012, pp. 554–561.

[8] N. Baldo, M. Requena, J. Nunez, M. Portoles, J. Nin, P. Dini, and J. Mangues, "Validation of the ns-3 IEEE 802.11 model using the EXTREME testbed," in *Proceedings of SIMUTools Conference, 2010*, March 2010.

[9] L. Barroso and U. Hölzle, "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.