Modeling and Simulation of Concurrent Workload Processing in Cloud-Distributed Enterprise Information Systems

Alexandru-Florian Antonescu, Torsten Braun
alexandru-florian.antonescu@sap.com, braun@iam.unibe.ch

1 SAP (Switzerland) Inc. Products & Innovation, Research
Althardstrasse 80, 8105 Regensdorf, Switzerland

2 University of Bern Communication and Distributed Systems (CDS)
Neubrückstrasse 10, 3012 Bern, Switzerland

ABSTRACT
Cloud Computing enables provisioning and distribution of highly scalable services in a reliable, on-demand and sustainable manner. Distributed Enterprise Information Systems (dEIS) are a class of applications with important economic value and with strong requirements in terms of performance and reliability. In order to validate dEIS architectures, stability, scaling and SLA compliance, large testing deployments are necessary, adding complexity to the design and testing of such systems. To fill this gap, we present and validate a methodology for modeling and simulating such complex distributed systems using the CloudSim cloud computing simulator, based on measurement data from an actual distributed system. We present an approach for creating a performance-based model of a distributed cloud application using recorded service performance traces. We then show how to integrate the created model into CloudSim. We validate the CloudSim simulation model by comparing simulation results using different VM configurations. We demonstrate the usefulness of using a cloud simulator for modeling properties of real cloud-distributed applications.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Distributed applications

Keywords
Cloud Computing; Distributed Applications; Performance Profiling; Modelling and Simulation

1. INTRODUCTION
Cloud Computing [9] enables provisioning and distribution of highly scalable services in a reliable, on-demand and sustainable manner. Often, the applications running in these distributed environments are designed to concurrently execute workload, making use of available multi-core computing resources. As example, distributed Enterprise Information Systems [11] often interact with large-scale distributed databases, requiring distributed processing of results coming from multiple systems.

In order to validate certain properties of distributed cloud systems, it is often the case that large-scale hardware test beds are used for observing the application behavior under conditions similar to those met in production environments. However, getting access to such large, distributed testbeds can be difficult, and reproducing results is often not possible, due to the use of shared physical computing and network resources. This leads to the need of modeling cloud applications and resources in order to perform simulations for testing “cloud-aware” software management algorithms. Examples of such complex interactions to be modeled and simulated are cloud infrastructure management [4], or Service Level Agreement-driven benchmarking of cloud services scaling[2].

We extend the work in [2] and [4] by building a simulation model for a distributed enterprise information system (dEIS) based on performance measurements gathered in a small-scale distributed testbed. We show how to analyze and model the distributed interactions between the dEIS entities in a state-of-the-art cloud simulator. We then perform multiple distributed experiments with varying concurrent workload. Finally, we compare the experimental results to the simulation, by running the same workload scenario.

Our main contributions can be summarized as follows. We present an approach for analyzing and structuring monitoring information belonging to distributed cloud services in order to build a model of the distributed application. We show how to integrate the application’s performance model in a cloud simulator, by converting from time-bound service performance indicators to CPU and network resource utilization. We describe an approach for increasing the scale of cloud simulation and for testing the distributed application’s behavior under different concurrent workloads.

The rest of our paper is organized as follows. Section 2 presents the related work in the field of cloud simulators as well as modeling and simulation of distributed cloud applications. Section 3 presents the algorithms used for building the simulation models and for modeling a distributed application in CloudSim. Section 4 presents the evaluation
results, and finally, Section 5 draws conclusions and gives future research directions.

2. CLOUD SIMULATOR

CloudSim [8] positions itself as a cloud simulator for both applications and infrastructure, as it allows modeling of hardware and software cloud resources. Among the modeled physical entities there are hosts, network links and datacenters, while the modeled software entities are virtual machines (VMs), brokers and cloudlets (tasks). The mentioned entities are manipulated using a Java API. The simulator is implemented using discrete event communication. CloudSim provides a wide selection of resource allocation policies for VM-to-host and task-to-VM mapping.

Buyya et al. [5] presented an approach for simulating large scale cloud environments by using CloudSim. While they described the steps required for simulating a large number of hosts and network connections, they did not show how to also model applications using the CloudSim simulator. Our work fills this gap by presenting a methodology for simulating a large-scale distributed application.

Garg et al. [7] extend CloudSim with more advanced network simulation capabilities. They model the datacenter network landscape by introducing switch and network packet simulation entities. The authors simulate a network application using four VMs and compare the results to the simulator, in a similar manner to our approach.

3. PERFORMANCE ANALYSIS AND MODELING OF A DISTRIBUTED SYSTEM

As representative dEIS distributed application we used the one described in [1, 2, 3]. The targeted dEIS system is composed of four core services: one or more Thin Clients (CS), a Load Balancer (LB), one or more Worker services (WK), and one or more Database Storage services (ST). Each service runs in its own VM and communicates asynchronously with the other services using a distributed service messaging bus. Fig. 1 presents the interactions between various dEIS services when executing a CS-initiated request. This also shows the breakdown of operations executed at each service, which serves as the foundation for building a simulation model of dEIS.

In order to simulate the dEIS application, we first recorded key performance indicators of dEIS services under constant workload. We then used these monitoring traces to create the dEIS’s simulation model composed of a large number of single requests’ execution measurements. For a given request we measured the round-trip times for the service calls displayed in Fig. 1, as well as the durations of the service-local calls.

For building the application performance profile we applied the following algorithm. We instantiated one VM for each (Consumer) CS, (Load Balancer) LB, (Worker) WK and (Storage) ST services. We initialized the concurrent load with 1 and the benchmark time duration with 10 minutes. Every 10 milliseconds the number of active requests ar executed by the system was compared to the target load, and if ar was below load, then a number of load – ar CS requests would be generated. For each end-to-end request, the following parameters were recorded: round trip durations, service-local operation durations and VM-level performance metrics (CPU, memory, network, disk). By recording all these parameters for each distributed request, a performance benchmark profile was generated under the constant workload load.

After running the benchmark described above, the dEIS performance profiles for constant concurrent load between 1 and max.load will be known and the performance profile will be generated with all service call round-trip times and service-local operation durations. This profile will then be used during the simulation, as described later in this section.

In order to build the application’s performance profile the following assumptions were made: (1) load balancing is applied at the LB service and (2) CS requests are independently executed by the dEIS services. This allows to extend the performance profile recorded with one VM per dEIS service to any number of VMs, as the performance of individual services will be influenced only by the level of concurrent workload and not by the number of running VMs.

3.1 dEIS Performance Profiling

We analyze performance profiles generated for each dEIS service while the system was sequentially executing requests
for 10 minutes, at a fixed concurrent workload. The physical setup consisted of three servers with Intel(R) Core(TM)2 Duo CPU E8400 CPUs, 4GB of memory and 1 Gbps network interface. CS and LB VMs were co-located on the same server, while WK and ST were each located on a separate server. A fourth server was used for running validation experiments, by using two additional VMs, one VM for the WK service and one VM for the ST service.

In Fig. 2 we show the performance profile of the dEIS CS service (thin client). Four application metrics were recorded for the CS service: (1) round-trip time for creating a LB session and receiving a confirmation acknowledgment, (2) round-trip time for sending a request to the WK service, (3) the time during which the response was queued before being selected for processing, and (4) time for validating the response. For the VM performance there were also four metrics gathered: (5) VM CPU utilization percentage, (6) memory usage, (7) network bytes read, and (8) network bytes written. The network byte read correspond to the messages received from LB (session creation confirmation, around 4KB) and WK (processing result, 10-30KB), and have a large variance. The CPU utilization has low values and is almost constant at around 20%, as expected for a thin client application.

In Fig. 3 we present the performance profile of the dEIS LB service. The application metrics recorded are: (1) session creation time, and (2) number of concurrent sessions. The LB VM metrics are the same as the ones gathered for CS. CPU utilization is also low at around 20%, justifying co-location of CS and LB VMs. Both network read and written bytes have a median of around 2KB and are caused by exchanging small sized monitoring and session creation/confirmation messages.

In Fig. 4 we present the application performance profile of the dEIS WK service. We gather the following monitoring information: (1) request scheduling duration - time a CS-request is queued until it is selected for processing, (2) round-trip duration of sending the WK-request to ST service, (3) ST-response queuing, (4) processing of ST-response, and (5) round-trip time for sending the response back to CS service and receiving the acknowledgment.

In Fig. 5 we display the WK's VM performance metrics. CPU utilization has high values because of the additional computing time required for processing the result from ST. Also, the size of the response sent to CS has a large variance (5-30KB), explaining the variance of the result processing time. Also, this rather large result size variance will influence the variance of the simulation results.

In Fig. 6 we present the ST's application performance metrics. The recorded metrics are: (1) time a WK-request is
queued until it is selected for processing, (2) time it takes to obtain a database connection for processing the data queries, (3) database query execution time, (4) SQL result set processing time, (5) size of the result in bytes, and (6) round-trip time for sending the result back to WK.

In Fig. 7 we show the VM performance metrics associated with the ST service. Both CPU and memory utilization are high due to the large amounts of processing required for retrieving and processing the data from the database.

Finally, in Fig. 8 we present how the various durations and round-trip times stack up to compose the complete end-to-end time of each chain of requests (CS-WK-ST) and responses (ST-WK-CS). This plot represents the foundation of the simulation model. During the simulation, vertical “slices” (service-local durations and round-trip times) will be used for simulating CloudSim cloudlets.

3.2 Modeling and Simulation

The performance-profiling traces previously gathered can be represented as shown in Equation 1. $\text{Profile}_{\text{load}}$ is a matrix with all measurable time values of applications’ operations under concurrent load. $RT(S_i, S_j)$ is the round-trip time of a remote service call on service $S_i$ from service $S_j$.

\[ D_i(\text{Op}_k) \] is the time of performing service-local operation $k$ on service $S_i$ and $(\text{cpu}|\text{mem}|\text{net})_{CS|LB|WK|ST}$ is the utilization level of CPU, memory and network on the VMs corresponding to CS, LB, WK and ST services. Each line of this matrix corresponds to a single chain of requests from CS, to WK, to ST, and then back to CS.

\[ \text{Profile}_{\text{load}} = \begin{bmatrix} RT(S_i, S_j), D_i(\text{Op}_k), (\text{cpu}|\text{mem}|\text{net})_{CS|LB|WK|ST} \end{bmatrix} \] (1)

By combining performance profiles for concurrent workload between 1 and $\text{max.load}$ end-to-end requests, we form the deEIS performance model $\text{Profile}_{\text{load}}$ as dictionary with key load and corresponding values $\text{Profile}_{\text{load}}$.

In order to convert the created performance profile to CloudSim entities we transform the durations in milliseconds to their equivalent MIPS rating. We also map the VM resource consumption for CPU, memory and network to CloudSim characteristics of the created cloudlets - CPU and memory utilization models and network input and output payloads. The conversion of milliseconds to MIPS is done according to Equation 2.

\[ \text{instr} = \frac{\text{ms}}{1000.0} \cdot \text{CPU}_\text{MIPS} \] (2)

\[ \text{instr} \] is the length in instructions of an operation with a duration of $\text{ms}$ milliseconds when executed on CPU with a MIPS rating of $\text{CPU}_\text{MIPS}$. This equation is valid only when there is only one task being executed on the CPU. If there are multiple tasks in execution on the same CPU, the tasks’ length in number of CPU instructions needs to be reduced by the CPU concurrency level in order to keep the same execution deadline. As our performance profile contains operations’ durations measured under constant workload, we apply Equation 3 in order to convert the operation’s duration in milliseconds to MIPS.

\[ \text{instr}^* = \frac{\text{instr}}{c_{\text{new}} + \text{concurrency}_M} \cdot (1 - \text{CPU}_\text{OS}) \] (3)

\[ \text{instr}^* \] is the task’s length in MIPS considering VM concurrent load, $\text{instr}$ is the value calculated by Equation 2, $c_{\text{new}}$ is the number of cloudlets newly submitted to the VM.
for execution, \(\text{concurrency}_{VM}\) is the number of cloudlets already executed by the VM, and \(\text{CPU}_{OS}\) is the average CPU utilization caused by independent OS-level kernel tasks. This calculation is performed in each CloudSim Datacenter Broker corresponding to CS, LB, WK and ST services for each batch of cloudlets submitted for execution. For running the simulation in CloudSim, a Datacenter Broker is created for each dEIS service and also at least one VM per dEIS service is added to CloudSim.

A simulation scenario is given as a list of pairs of simulation time and the corresponding expected number of concurrent CS-requests at that time. The list is then linearly-interpolated for determining the expected concurrency for each simulation time slot. For each simulation time step, the current number of running CS-requests \(cl\) is compared to the simulation plan values \(cl_{sim}\), and if \(cl < cl_{sim}\), then \((cl_{sim} - cl)\) CS-cloudlets are created.

\[
D_{n,\alpha'} = \sup_x |F_{1,\alpha}(x) - F_{2,\alpha'}(x)|
\]

where \(F_{1,\alpha}\) and \(F_{2,\alpha'}\) are the empirical distribution functions of the first and the second sample respectively.

In Fig. 9 we show the comparison between the simulator input values \((\text{Profile})\) performance traces and the output produced by CloudSim, showing very similar density distributions of the execution times. The difference between the distribution of values is low, as indicated by a \(D\) value of 0.0855 of the KS test.

We also validated the ability of the simulation model to correctly reproduce end-to-end response times under constant concurrent workload. In Fig. 11 we show the KS test values and linear dependency between the values gathered in experiment and simulation. The graphs show a good correlation with a value of 0.3 of KS statistic (maximum
difference between CDF functions), and a linear coefficient of 1.05 between experiment and simulation results (indicating that the simulation values differ with only 5% from the experiment values).

In order to test the simulation model under varying workload conditions and with a different number of VM instances for the deIS services, we ran two additional experiments, as shown in Fig. 10. In the first experiment still used one VM per service. However, we varied the concurrent workload from 1 to 17 and back to 1 parallel requests. In the second experiment we used 2 VMs for each of WK and ST services, using in total 4 servers, while the concurrent workload was varied from 1 to 20 parallel requests, with 5 seconds pause before switching to a new workload level. The experiments were performed with real resources and then the results were compared with the same scenario running in the simulator.

In case of experiment 1, simulation results shown in Fig. 12 validated the modeling of the dependency between service tasks durations and the corresponding number of executed CPU instructions, as the simulated end-to-end response time closely followed the measured deIS response times. The increased variance in simulation results was caused by the simulator’s event communication overhead and by the fact that the simulator randomly chooses the requests to be simulated.

Experiment 2 validated the modeling of concurrent computing workloads distributed across multiple VMs. The simulation results closely matched the ones measured during the experiment experiment, as can be seen in Fig. 13.

5. CONCLUSIONS

We have presented an approach for building a simulation model of a distributed application for concurrent workload processing, by analyzing application’s performance traces gathered in small-scale real deployments. We then showed how to integrate the simulation model into CloudSim and how to build a simulation with varying concurrent workload. We have also presented an approach for modeling concurrent computing workloads using the CloudSim simulator. Finally, we validated our models by running experiments with varying workload and with different VM scaling factors.

Our results show that it is possible to accurately model the behavior of a distributed enterprise information system using CloudSim, although extensions for supporting concurrent task processing need to be added. Also, due to variance in the execution times of distributed requests, the simulation models cannot perfectly represent the real distributed systems, however, a good-enough representation of key application-level metrics has been shown to be obtainable.

6. REFERENCES