# An Implementation of the WS-Agreement Negotiation Standard in CloudSim

Benedikt Pittl
Faculty of Computer Science
University of Vienna, Austria
Email: benedikt.pittl@univie.ac.at

Werner Mach
Faculty of Computer Science
University of Vienna, Austria
Email: werner.mach@univie.ac.at

Erich Schikuta
Faculty of Computer Science
University of Vienna, Austria
Email: erich.schikuta@univie.ac.at

***Abstract*—The currently dominating market mechanism for selling IaaS resources is the so called *supermarket approach*: Consumer buy resources from providers without negotiation. The recent development of Amazon EC2 spot market shows that dynamic market mechanisms are becoming increasingly popular. The so called *Bazaar approach* is such a dynamic market mechanism. On a Bazaar-base market consumer and provider can negotiate all characteristics of IaaS resources described by SLAs. The WS-Agreement Negotiation standard describes concepts for such bilateral negotiations on Bazaar-based markets.**

**The scientific simulation framework CloudSim is not able to simulate Bazaar-based markets. Thus we extended the framework by developing the so called *Bazaar-Extension* which implements the basic principles of the WS-Agreement Negotiation standard. Using the Bazaar-Extension consumers and datacenters are created, strategies are assigned to them and the result of the bilateral negotiation simulation are analysed.**

## I. Introduction

A digital service market may be envisaged as the culmination point of stake-holders providing services used along each link in a value chain. Currently such service markets are evolving dramatically [1].

Service markets allow providers to sell their resources to consumers. Therefore, they form contracts which are technically realized by SLAs (Service Level Agreements). As described in [2] cloud providers can make use of SLA technology to advertise and offer their services capabilities while consumers are able to formalize their service level objectives through SLAs. SLA contracts are described in protocols like WS-Agreement [3].

In this paper we consider IaaS (Infrastructure as a Service) as an example of a cloud service. Thus we use the terms IaaS provider and cloud provider synonymously.

Usually cloud providers run data centers providing IaaS resources. Providers sell these resources to enterprises in forms of virtual machines (VMs). Virtual machines are not commodities which are totally interchangeable. Virtual machines are goods which have several characteristics such as: (i) processing power (ii) storage (iii) RAM (iv) price. Today, these machines are mainly traded directly for fixed prices from provider to consumer [4].

For our research project focusing on economical aspects of Cloud Computing we needed an adaptive cloud simulation environment which is able to run our envisioned Bazaar-based market ecosystem. The basic requirement for simulating our Bazaar-based market ecosystem is the simulation of resource allocation mechanisms based on bilateral negotiations. We found several relevant frameworks: (i) The simulation environment greenCloud [5] was developed by the University of Luxembourg. It focuses on the simulation of energy consumption of Cloud infrastructures. (ii) iCanCloud [6] is a Cloud simulation framework for analysing trade-offs between cost and performance of a given set of applications executed in a specific hardware. (iii) The CloudSim framework [7] was developed by the University of Melbourne. A lot of other researchers have extended this framework.

None of the frameworks we found is able to simulate service markets (and consequently negotiation based resource allocation mechanisms). However, for the simulator CloudSim [7] two market extensions (including scientific papers) are existing using CloudSim for simulating resource allocations.

- The extension introduced by [8] allows to run a double combinatorial auction. It is published on the official CloudSim website (http://www.cloudbus.org/cloudsim/).
- The authors of [4] described that they extended CloudSim for running procurement auctions.

To the best of the authors knowledge no bilateral negotiation extension exists enabling Bazaar-based markets. Therefore, we developed the **Bazaar-Extension** for CloudSim as this framework (i) is well known by the community (ii) offers a lot of extensions (iii) is appropriate for introducing market based components like the Bazaar-Extension.

The remainder of the paper is organized as follows: Section II describes the high level architecture of the Bazaar-Extension. The paper is closed by a description of further research in section III.

## II. Architecture Overview

We designed the Bazaar-Extension so that the existing CloudSim architecture as well as the classes need not be modified.

Figure 1 illustrates a high level view of the Bazaar-Extension architecture including a rough communication sequence. The Bazaar-Extension is based on the CloudSim framework and uses the framework fxyz for the visualization of the negotiation results in 3d plots.

The Bazaar-Extension processes so called negotiation messages, which we developed during the creation of the Bazaar-

978-1-4673-9933-3/16/$31.00 ©2016 IEEE

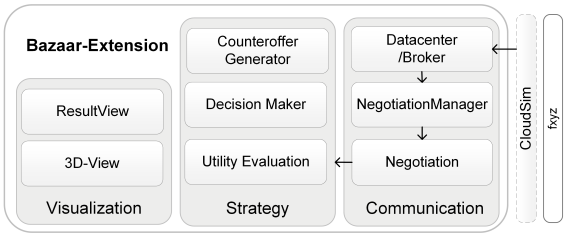
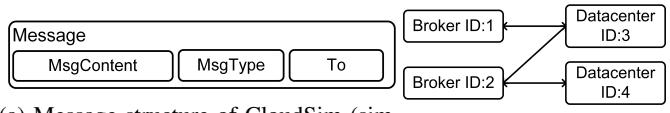



Fig. 1: High level architecture of the main components




(a) Message structure of CloudSim (simplified)

(b) Negotiation between consumer and provider

Fig. 2: CloudSim structure

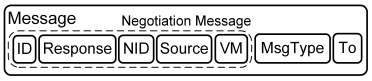



Fig. 3: Message containing a negotiation message

Extension. These negotiation messages contain offers and represent the messages defined in the WS-Agreement negotiation standard.

For the Bazaar-Extension a negotiation datacenter and a negotiation broker was created. The negotiation broker and the negotiation datacenter are entities. They inherit the basic behavior of the entity class defined in CloudSim. Each datacenter and each broker has exactly one negotiation manager. All messages send to a broker or a datacenter are passed to their negotiation manager. So the negotiation manager can be considered as a gateway inter alia responsible for forwarding messages to negotiations. Further, the negotiation manager checks if the received message is a negotiation message or a usual CloudSim message. If the received message is not a negotiation message then it is not processed by the negotiation manager. Negotiation objects encapsulate negotiation relevant data and use a negotiation strategy which decides how to respond to received offers.

### A. Negotiation Messages

During a simulation flow in CloudSim events are exchanged. We use terms event and message synonymously. The events used by CloudSim are described in the Java class $SimEvent.java$ in the CloudSim framework [9]. A event contains three fields which are relevant within this paper as shown in figure 2a.

- The MsgContent is of type $Object$. Therefore the message content can contain an object of any type.
- The Msg Type field is an integer indicating what content is send. These integers are also called tags. The tags represent message types. CloudSim defines several message types. For example message type $Register\_Resource$ has the value 2. It is used by datacenters to register at the CIS (Cloud Information System). The class $CloudSimTags.java$ contains tags which are defined as public static final integers.
- In CloudSim each entity is identified by an unique integer. This id is necessary to send a message to another entity via CloudSim. In figure 2a the id of the destination is represented by the $To$ field.

For the Bazaar-Extension we created Bazaar-Tags representing the offer states defined in the WS-Agreement Negotiation standard. Further an internal tag triggering the evaluation of datacenters and brokers was defined. Such an internal tag is necessary because of the round based simulation flow of CloudSim. The CloudSim messages were used by the Bazaar-Extension for exchanging negotiation messages.

The simplest way for exchanging negotiation messages is to use the $MsgContent$ field. A sender could put an offer for a VM including a price into this field and send it to a negotiation partner. CloudSim adds the id of the sender to messages so that the receiver knows the message creator. This id is also called the source. Using the source the consumer as well as the provider is able to distinguish between messages received from different entities. So consumer and provider are able to negotiate with several entities at same time as illustrated in figure 2b. This figure shows several datacenters and brokers including their id. Broker 2 is negotiating with datacenter 3 and 4 at the same time. Broker 2 is able to distinguish between offers received from the datacenters 3 and 4 as it knows the source. Similar, datacenter 3 is negotiating with two brokers at the same time is able to distinguish between offers received from broker 1 and 2 using the source.

If a consumer needs two VMs (for example a small and a large VM) its broker starts negotiating with a datacenter for each VM. Thus the broker has two negotiations with a datacenter at the same time. A receiver of a message is able to use the source for distinguishing between different entities. However, the receiver is not able to distinguish between different negotiations with the same entity. The source identifies the entity but there is no so called negotiation id. Therefore a negotiation message class was created. Instances of this class can be exchanged between entities via the message content field. An overview is depicted in figure 3. So the priced VM stored in the MsgContent field is replaced by a negotiation message object containing a VM which is offered to the negotiation partner.

The negotiation message introduces a negotiation id ($NID$) which is a $UUID$. Consumer and provider use the same $UUID$ for identification of a negotiation. The consumer generates an UUID and sends it to the provider in exchange for its templates. By using the $NID$ the receiver of a message is able to distinguish between parallel running negotiations of an entity.

A negotiation message contains several additional fields.

- ID: The ID is an UUID identifying the message. Each message has an unique ID.

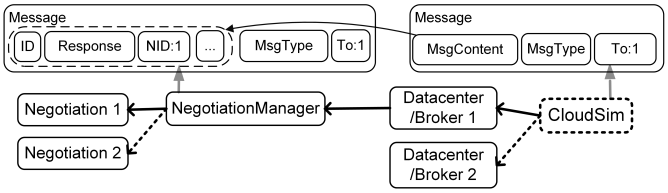



Fig. 4: Message passing

- Response: Each message has an reference to its predecessor message. If several messages are exchanged the reference helps to find out to which message the received message belongs. The template has not a reference as it is the initial message.
- The source field contains an integer representing the entity id of the sender of the message. This id was added into the negotiation message to simplify negotiation message processing.
- The VM field contains the description of a virtual machine including its price.

A broker requiring a VM creates a negotiation id and contacts several datacenters with a Resource Characteristic Request. So the broker sends the same negotiation ID to all datacenters. The negotiation id and the source field form the so called interaction id. The interaction id identifies a negotiation with an entity.

The Bazaar-Extension uses the negotiation messages as message content for running negotiations. Therefore the CloudSim framework needs not to be modified for running negotiations

### *B. Negotiation Manager*

The two main tasks of the negotiation manager are to forward negotiation messages and create new negotiations. Figure 4 shows an example how message forwarding works. CloudSim forwards messages to entities using the destination field (To field) stored in the message. If the entity receiving the message is a negotiation entity it forwards the received message to its negotiation manager. The negotiation manager uses the message type to determine if the received message is a negotiation message. If the received message is a negotiation message it casts the message content referenced as an object to a negotiation message.

A negotiation manager is responsible for several negotiations. The negotiation message contains a negotiation id. This negotiation id is used by the negotiation manager to forward the message to the corresponding negotiation.

A new negotiation is created if a negotiation message was received containing a negotiation id which is unknown by the negotiation manager. For creating a new negotiation a negotiation factory is used. This factory encapsulates the creation of a new negotiation including a strategy. The factory is necessary as strategy creation can be a complex task. So a strategy may for example contain a counteroffer generator or an utility evaluator for ranking received offers. Such a utility evaluator was described by us in [10].

The tasks of a negotiation manager used by the consumer and datacenter are identical. Both forward messages to negotiations and use a factory for creating new negotiations. Consumer and provider use different factories. This is because consumer and datacenter use different strategies in the negotiation objects delivered by the factory.

### *C. Negotiation*

The negotiation class is a container for storing negotiation relevant data. Negotiation relevant data are the negotiation id and a tree tracing the negotiation history for each negotiation partner. Further a reference to the used strategy for the negotiation is stored.

The strategy as described in the following section creates messages which are sent to negotiation partners. So the strategy passes the messages to the negotiation which passes the messages to the negotiation manager. The negotiation manager adds entity data to the negotiation message and passes the message to the CloudSim framework which delivers the message.

### *D. Strategy*

The strategy is responsible for responding to received negotiation messages. It sets for example conditions for accepting or rejecting received offers. A goal of the Bazaar-Extension is to create and test new negotiation strategies. Thus we tried to keep the strategy creation as flexible as possible within the Bazaar-Extension. Basically, a strategy has to extend an abstract negotiation strategy class which enforces only to implement an evaluation method. This method is called by the responsible negotiation object after all messages from the negotiation partners were received. The negotiation strategy has access to the negotiation history.

We have already implemented initial negotiation strategies for testing the Bazaar-Extension. The strategies make use of the high level strategy process depicted in figure 5. Of course it is possible to extend this basic strategy or create new strategies in the Bazaar-Extension. The dark boxes represent the strategy components depicted in figure 1. The basic strategy process is identical for consumer and provider.

First, the received offers in a round (see [10] for a description of the term round) are collected. Then they are evaluated in order to rank them. We used the utility function described in [10] for doing the ranking. Afterwards the ranked offers are handed to a decision maker. It decides how to respond to the received offer. In case in which the decision maker decides to create a counteroffer the received messages are handed to the counteroffer generator. In all the other cases the corresponding responses are created.

In our initial experiments we were using thresholds in the decision makers. This means that if e.g. a offer was received which has an utility exceeding the used threshold, then it is accepted. The most complex component in the process seems to be the counteroffer generator. Computational intelligence

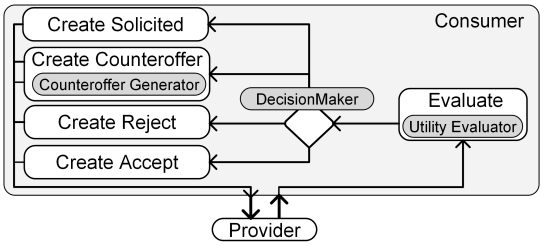



Fig. 5: Strategy process

techniques like neural networks or genetic algorithms may be suitable for this task.

With the Bazaar-Extension new negotiation strategies can be developed and evaluated.

### E. Scenario

In the Bazaar-Extension scenarios are executed. A scenario has a list of datacenters and a list of brokers attending the scenario. A scenario is responsible for assigning strategies to the entities. After running the scenario the datacenters and brokers are accessed in order to analyse results.

### F. View

The created view was developed using JavaFX determining the basic architecture of the view. JavaFX is considered as an eligible alternative for Swing [11]. As the view is visualizing the negotiation results it is called result-view. Exactly one scenario can be visualized by the view. The basic structure of the view is shown in figure 6. The numbers in the figure mark three areas which were realized using three fxml files. Together these files form the result-view. Area 3 is the basic view containing a menu bar as well as the other two view areas. By selecting a scenario its data is loaded into area 1. This area is the so called scenario area as it visualizes the entities participating in a scenario. By selecting a negotiation of an entity negotiation details are loaded into the view area 2. It consists of two diagrams:

- The tree list shows the messages which were exchanged during negotiation. Each entry in the tree list shows the characteristics of the VM of the received/sent offer.
- The utility-utility plot visualizes the tree list. The ordinate shows utility of the offers for the consumer while the abscissa shows the utility of the offers for the provider. The utility evaluator is responsible for assigning utility values to offers. The different colors indicates in which iteration the offer was received/sent. It is possible to calculate the Pareto-Boarder which shows how efficient the used negotiation strategies worked.

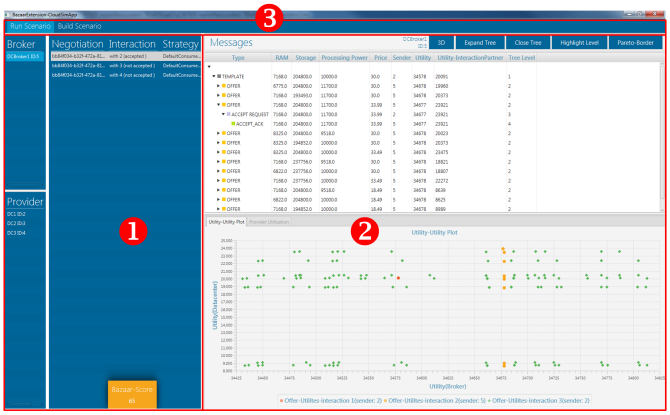



Fig. 6: Screenshot of the GUI based on JavaFX

## III. Conclusion and Further Research

We envision a comprehensive framework for autonomous, dynamic and adaptive negotiation processes of IaaS resources.

Current papers in the field of cloud economics consider a simplified consumer-provider market. Thus additional market participants and market components are neglected which limits the evidence of the outcomes derived from such models.

In our further research we will develop further components based on the Bazaar-Extension: For example taxes on cloud markets have not been considered yet by the scientific community.

## References

[1] W. Mach, B. Pittl, and E. Schikuta, "A Forecasting and Decision Model for Successful Service Negotiation," in *Services Computing (SCC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 733–740.

[2] P. Hasselmeyer, H. Mersch, B. Koller, H. N. Quyen, L. Schubert, and P. Wieder, "Implementing an SLA negotiation framework," in *Proceedings of the eChallenges Conference (e-2007)*, vol. 4, 2007, pp. 154–161.

[3] H. Ludwig, "WS-Agreement Concepts and Use Agreement-Based Service-Oriented Architectures," *IBM Research Division, Technical Report*, 2006.

[4] P. Bonacquisto, G. D. Modica, G. Petralia, and O. Tomarchio, "A Strategy to Optimize Resource Allocation in Auction-Based Cloud Markets," in *Services Computing (SCC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 339–346.

[5] D. Kliazovich, P. Bouvry, and S. U. Khan, "Greencloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[6] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "icancloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[7] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities," in *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*. IEEE, 2009, pp. 1–11.

[8] P. Samimi, Y. Teimouri, and M. Mukhtar, "A combinatorial double auction resource allocation model in cloud computing," *Information Sciences*, 2014.

[9] "The CLOUDS Lab: Flagship Projects - Gridbus and Cloudbus." [Online]. Available: http://www.cloudbus.org/cloudsim/

[10] B. Pittl, W. Mach, and E. Schikuta, "A negotiation-based resource allocation model in iaas-markets," in *8th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2015, Limassol, Cyprus, December 7-10, 2015*, 2015, pp. 55–64.

[11] JavaFX FAQ. [Online]. Available: http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#6