Automated Design in the Cloud

Georgios Andreadis, Konstantinos-Dionysios Bouzakis, Athina Vakali, and Konstantinos Stamos

***Abstract*—Cloud Computing can accelerate industrial and mechanical design. It can also bring people and ideas together to design products collaboratively. Its adoption by the enterprise community is increasing opening opportunities for affordable scalable IT infrastructures even for small companies. In this work we study a system that exploits the cloud computing paradigm to realize a scalable CAD service for designing lifts. In this context we present and dissect the architecture and discuss several issues. The system is designed with the principle of reducing the cloud size (cost) while maintaining reasonable QoS and considerable effort has been done to be technology agnostic.**

***Index Terms*—CAD, cloud computing, lift, manufacturing.**

I. Introduction

Cloud Computing can accelerate industrial and mechanical design. It can also bring people and ideas together to design products collaboratively. At an enterprise level it can boost on-demand the infrastructure size at a manageable monetary cost. In particular a Cloud can be treated as a set of a uniform standardized virtual hardware and software resources that increase or decrease in size on-demand. A number of studies [1]-[3] formalize and categorize the Cloud Computing principles and architectures under a consistent scheme. Usually the various categories are being named as XaaS where S stands for Service and X stands for either Hardware, Software, Platform, Infrastructure, Desktop, Database, Framework, Organization or any other virtualized technology.

Our view of the Cloud in this paper is a set of VMs (Virtual Machines) that can be spawned, killed, and customized on demand. Given this scheme and the almost infinite long-term potential of the Cloud, we present a case study for a scalable architecture that manages a fauna of VMs for CAD (Computer Aided Design) of hydraulic lifts in an attempt to reduce the infrastructure cost, automate the design process, and minimize the human error. This case study is performed and implemented on behalf of a reputable company.

The CAD systems became a key asset in every product design and construction process the last decades. Important contributions of the CAD modeling include a strict description of the illustrated prototype products, ability for further customization and easily managed and distributed product information. In the context of the lifts engineers and designers have to be employed for calculating, studying and designing annually hundreds of design layouts at offering and ordering level, supporting the work of the sales people. Additionally, investments in expensive hardware infrastructure are necessary as the deployment of the many CAD software licenses dictate. Consequently, the manufacturing cost of the final product is significantly increased. Promptness, accuracy and fullness of the order/offer, determine to a great extent the success of the sale and a company's reputation. However, with such excessive parameterization of the designs the human error factor is not negligible. In fact, an estimation of the human computing/designing mistake is around 10% (internal report 2007) of the total sales. Therefore, the need for an automated system, that customizes and designs a product automatically, becomes evident. Motivated from the above challenges and the potential of Cloud Computing as an enterprise level solution, our primary contributions are:

1) Propose a technology agnostic architecture for managing a set of Virtual Machine workers. The life-cycle of each VM involves its creation, customization, operation, and discard. Each VM offers a CAD environment as a service.
2) We study the case of a lift design by identifying building blocks, templates, and design rules.
3) Suggest a scheduling algorithm for managing the requests on the Cloud. This is designed to be scalable, resilient, and concurrent.
4) We discuss issues related to the advantages/disadvantages of the system in practice, and we offer insight for future improvement.

II. Related Work

The recent advances in Cloud Computing and virtualization [4], [5] are attractive, affordable, and scalable solutions for potential deployment of CAD software, and eventually setting up an entire industrial-scale infrastructure. The outline of the Cloud computing technology is that there is an "infinite" amount of processing power and hardware available on demand, available in affordable prices. Computing is being transformed into a model consisting of services that tend to resemble to traditional utilities such as water, electricity, gas, and telephony. Cloud Computing is an emerging technology which enables the on-demand availability of virtualized resources. Cloud Computing effectively can turn a small-to-medium company into a company having large-scale infrastructure on-demand without have to buy a single PC.

Traditionally there is a Cloud provider (e.g. Amazon, MS, Hp) that offers a set of services in the form of an API (Application Programming Interface). Via this API one may write software applications that manage a set of Virtualized

Manuscript received September 9, 2014; revised March 6, 2015.

G. Andreadis and K.-D. Bouzakis are with the Mechanical Engineering Department, Aristotle University of Thessaloniki, Greece (e-mail: andreadi@eng.auth.gr, bouzakis@eng.auth.gr).

A. Vakali and K. Stamos are with the Department of Informatics, Aristotle University of Thessaloniki, Greece (e-mail: avakali@csd.auth.gr, kstamos@csd.auth.gr).

recourses. A Virtualized resource is actually a Virtual Machine (VM) which is essentially a machine with CPU, hard drive, RAM and operating system being virtualized by software. Inside a VM there can be installed preconfigured software to perform actual, "real" complex tasks.

A Cloud in this context can be considered as a pool of such VMs which can be monitored, stopped, started, and deleted on demand as needed. The primary challenge we face in this paper is how to take into advantage such enormous amount of processing resources for the benefit of the manufacturing process.

Motivated by the potential the Cloud Computing has to offer, we suggest an architecture ready to accommodate tasks concerning the designing process. More specifically the proposed architecture is developed with the following principles in mind:

Extensibility Sets of rules are defined that describe classes of the objects to be designed. This includes dimensions and relative positions of design blocks. This generic approach may lead to any product.

Scalability Depending on the workload the system is read to scale on-demand. From a single VM up to hundreds. This dynamic expansion and shrinking of the infrastructure facilitates the optimal usage of the resources within the Cloud and reduces effectively the final cost while maintaining acceptable response times.

Quality of Service (QoS) The architecture supports configuration about response times; the actual time span within a document must be prepared. Furthermore, the delivery of the documents is ensured since upon a failure of any VM the design can be prepared somewhere else transparently.

Monitoring the pending and running design tasks can be monitored, retrieved and canceled on-demand. Furthermore the active VMs are monitored periodically for failures and updates.

Security The communication among the various elements of the infrastructure is performed with respect to privacy and enhanced security via secured connections and protocols.

We bind together the technology of Cloud Computing and the standardized CAD to sketch and build a system that ultimately reduces errors, reduces infrastructure costs and adapts to the current industrial needs. Although our implementation is focused on the context of hydraulic elevators, it is not limited by that. One may specify different parameters and rules to identify and eventually produce other products as well.

III. System Architecture

At this point a bird's eye view of the system's architecture is presented. The basic entities of the system include a) the Blocks, b) the Rules, c) the Templates, d) the Tickets, e) the VMs, f) the Orchestrator. The various interactions between them, as discussed later, are illustrated in Fig. 1. The Blocks are common sub-designs that are used as-is during the designing process (e.g. a motor). The rules include information and constraints about the placement of the blocks and the designing of non-block sub-designs (e.g. there must be no more than 3cm distance between the door and the shaft). The Templates are user-defined design skeletons (e.g. the positions of various blocks). The Tickets are in effect messages that may contain a command for a VM (e.g. get status update), and may encapsulate data (e.g. new blocks, new rules, or the final design). The VMs are the actual places where the designing process is performed. The Coordinator is responsible for the scheduling of the Tickets and for the monitoring and maintenance of the VMs. Each entity has a more complex structure as follows.

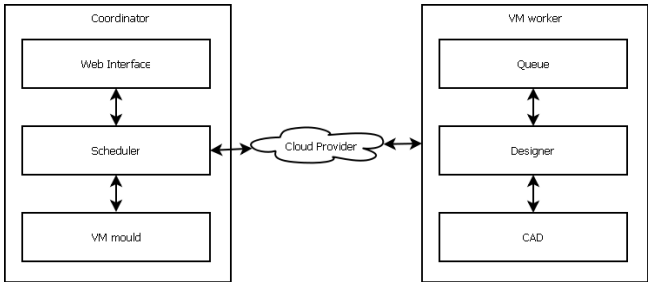

Fig. 1. System architecture.

A. Basic Entities

Each basic entity may have internal structure and serve various purposes. In particular: _ Blocks Since a large number of objects on the lift plans are reused frequently, it is efficient to retain a library of pre-drawn blocks of these objects and inject them when needed. These objects include, for instance, doors, brackets, pilots, mechanisms. A block may appear visually as in Fig. 2. The collection of the blocks can be updated (insert, delete, update);

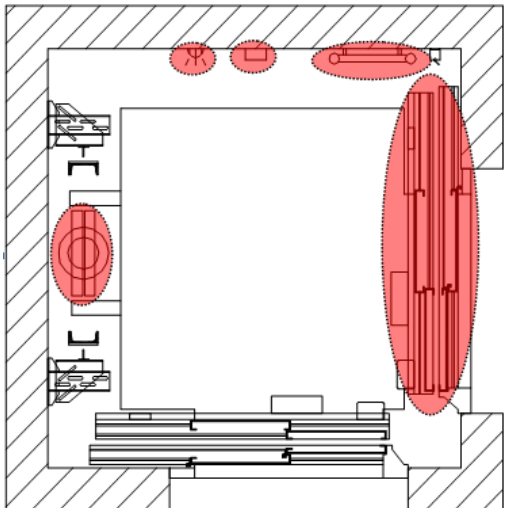

Fig. 2. Blocks injected into the design.

Rules. The rules define the relationships of the blocks and the non-block in during the design process. Especially, for each design, a tree is constructed where each node of the tree wraps the children-subtrees. The root of the tree is the design itself. The rules define constraints in terms of containment, siblings positions, and some general positional constraints.

Templates. A Template is a set of variables/parameters for the design of the elevator. They are defined by the users via a Web form. In practice a Template is an XML file that is being verified in a two-step process. Check if the schema is respected, and then check if any Rules are broken

Tickets. A ticket is in effect a SOAP envelope that wraps

any form of information exchanged in the system. It contains fields including but not limited to a unique ID, the priority, the timestamp when it was created, the IP address and the port of the sender, the type, and possibly a data section. The data section may contain a new rule/template/block or commands for deletion, query for status update, request for the VM to be terminated, or an actual design document that has just been designed.

VMs. In each VM the actual design operation is performed. In terms of software and design content each VM is identical with the others. Of course, each VM is responsible for designing lifts according to the tickets, different tickets for each VM. In particular, a VM consists of a CAD application (Autocad in our case) and a designer module that according to the tickets, the templates and the rules it designs the actual document using the API of the CAD application (in our case, DWG document and ObjectARX API). The designer module bridges the tickets stream from the coordinator to the CAD software. In effect the designer module exposes aWeb services API for the various ticket operations, queues the tickets, verifies them, and executes the respective commands. The VMs are located at the infrastructure of the Cloud provider and they have encrypted virtual hard drives by default to prevent security leaks.

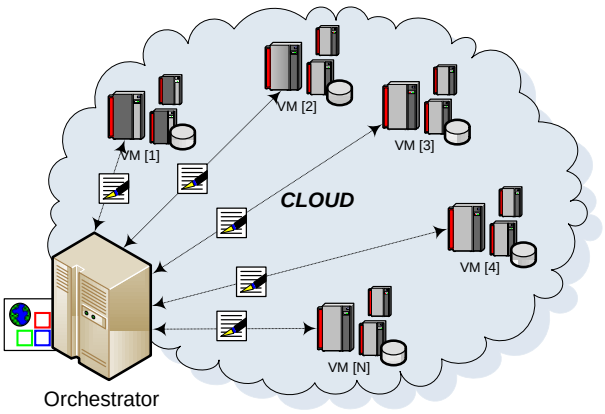

Fig. 3. Orchestrator managing design requests.

Orchestrator. The Orchestrator (Fig. 3) is essentially a server that offers Web interfaces for managing the whole infrastructure along with requests for design creation. It can be part of the Cloud or it can be located in a company's intranet. More specifically, a designer (human) is given a set of options to configure the design of a custom hydraulic elevator (e.g. its shaft dimensions, positions of the doors etc.). Then in a matter of a few minutes the Orchestrator forms a new ticket containing the design parameters, submits it to the Cloud, and receives the produced design available under a URI. Afterwards, the designer can further edit the design since is exported into an OpenDWG specification compliant file, allowing any application to access it. Another role of the Orchestrator is to define a set of rules that are applied during the design process. The rules include dimensional relations between various elevator components and set other business specific requirements. This enables design adaptation to the end client's desires and needs. The product can be built either as the baseline of its series, or can be customized with materials and accessories, each of which has its own ramifications on the final design of the product. Its placement allows for very granular changes in the purpose and structure of the rest of the system. Moreover, it helps to create reusable intelligence without the need to explicitly transform designers' intentions into code. Through the Orchestrator one is able to update a database of pre-drawn blocks of objects including doors, brackets, pilots, mechanisms etc. The most useful aspect of this approach is that it enables the designer to use the automatic drafting system to expand the variety of these components, by creating new blocks. The blocks participate inside rule definition and they are used during the design process.

B. APIs

All the APIs are implemented and exposed via Web services (WSDL document) and provide respective encryption and authentication schemes. Here, we present only the basic operations of the APIs. We divide the available APIs into two groups according to which module is responsible to serve each request:

Designer. API execute/cancelTicket() As mentioned earlier a ticket may contain any form of command or data like add new block or cancel rule. Both operations alter the queues content, meaning that the cancel ticket just removes it from the queue and the execute ticket places it at the correct position in queue according to the priority and time-stamp of the ticket. getStatus() Returns a ticket with information about the system's load and the queue's load. The load is defined as the moving average of the last 1, 5 and 15 minutes of wall clock time. getTicketStatus() retrieves the status of the ticket (done, queued, failed) and if done it encapsulates the result (DWG document). system() It orders the system to power-off, reboot, halt any pending tickets or resume the queue. The tickets' execution is performed based on the priority and then by FIFO.

Scheduler. API insert/delete/updateBlock() This will broadcast the related ticket including the related block information to all VMs. The VMs are obliged to execute it with high priority. insert/delete/updateRule() This will broadcast a ticket for altering a rule. This ticket has a high priority and it is executed as soon as possible. createDesign() Given a template the coordinator prepares the ticket and it submits it to the appropriate VM. restart/poweroff() Waits for a period of time to let the tickets finish and then broadcasts the ticket to all VMs.

C. Scheduling

The goal of the scheduler is two-fold; serve the users with new designs while keeping the Cloud as small as possible. This delicate balance is vital for efficient cost management and swift service. In practice all the system's operations originate and being controlled by the scheduler. Furthermore, the scheduler is fault tolerant in case of a failed tickets or event failed VMs, reports the failed tickets and provides details to the users. All in all the system should be up and running, it is a mission critical runtime system. More specifically, the algorithm of the scheduler includes several phases described as follows:

Startup. During the start-up of the system two VMs are spawned. The first VM is instantiated given a mould VM and it will be an actual worker. The second VM becomes the

current mould from which future VMs are spawned. Every ticket that involves the alteration of rules or blocks it is also applied to the mould VM to keep it up-to-date. Shrink Periodically the scheduler checks the load of the VMs and frees the one that is close to idle. The minimum number of worker VMs must be at least one. If a decision is made and a VM is about to be terminated then the scheduler prevents the assignment of any tickets to it and also waits for the termination of the remaining tickets.

Design. Given a template from the user, the scheduler identifies the most loaded server that satisfies the min QoS and sends the ticket. That is, the scheduler keeps statistics about the average time a ticket requires to be created (not waiting in the queue) and evaluates the waiting time for all the VMs. Then it selects the one that has the maximum possible queue length but satisfies a maximum waiting time limit (QoS). This load unbalancing policy enables the appearance of idle VMs so that they can be terminated during the shrinking process saving money. Then, at the expected time of completion the scheduler periodically probes the worker VM for the ticket's result. In the case where no VM satisfies the QoS limitations a new VM is spawned from the current mould.

Shut-down. During the shut-down process, any remaining ticket is given a time out to be completed, the mould replaces the old one and the VMs are eventually terminated (gracefully and then forcefully).

The scheduler logic although straight forward it involves proper coordination of the tickets and in order to prevent deadlocks applies the tickets asynchronously allowing different designs to be designed perhaps with different rules. This is a rather rare case, however, and do not pose any actual problems in practice as the rules and blocks rarely change and the designs are always verified by humans. Every interesting event is logged (like failures) and statistics are gathered to help future improvement.

IV. Discussion

In the above we have presented in a technology agnostic way how to expose a CAD application to do more with less resources. The advantages of this approach come straight from the Cloud computing domain. Those include payment according to usage which is far better than buying/maintaining/upgrading strong desktop machines to run the CAD application for each employee. Furthermore, adding an automated rule-based mechanism to build the designs tends to reduce the common human error and improve productivity of the designers. The designers revise the final document which expresses their intentions rather than spending hours doing repetitive designs. In terms of energy consumption and environmental pollution by using powered resources, adapting to the load and the day-to-day needs is the way to go.

Despite the improvements of scalability and cost there are several considerations to be taken into account including the following:

Security. This is an issue that is usually forgotten and in the case of a Cloud environment is essential. Because of the nature of the Cloud you have to trust the safety of your data to the discretion of the provider. In order to fully protect the system all the communication channels must be encrypted, and most importantly the virtual hard drives of the VMs must also be encrypted. The coordinator itself should be located in-house where the maintainers can have physical access.

Failure. A possible shortcoming of the proposed architecture is the fact that although the VMs can be replaced easily the coordinator cannot. The coordinator consists a single point of failure. A way to approach this problem is to have many coordinator replicating the activity of the primary coordinator and resume if a failure occurs. Another approach is to create groups of VMs with different coordinator in an attempt to reduce the failure in to a small group of VMs.

CAD. The CAD software itself is not designed or either licensed as a server application. This imposes several challenges. The majority of the application's time is spent in waiting while nonconcurrent processes are running. In order to maximize the availability of the component, we have encapsulated the CAD software in a single user interface, which is then wrapped by a queued Web Service. Another issue that was encountered is the need in part of the CAD software for administrative privileges on the host system, which would in turn require the same privileges for the web server that supports the WS. This is generally ill advised since it would allow for shatter attacks. That means that security exploits that are found on the web server are always critical since they can perform anything an administrator can do. An advantage of the architecture we propose is the gradual limiting of the privileges on the services, as the attack surface and exposure to the outside increases. In terms of licensing the problem is to define who the user is and how many are there. So, still, each user must have obtained a license.

V. Conclusion

The suggested architecture is a proof of concept that heavy design procedures can be carried out effectively in a Cloud. We broaden the so far short sighted approach of design which included a single real workstation with a designer. The potential within this architecture would boost small-scale companies by having access to virtually unlimited resources in an affordable way. For large scale companies, it offers a convenient way to reduce infrastructure costs but also to cope with many requests simultaneously. Future directions include the presentation of statistics from long term operation of the system and a detailed study of concurrency issues related to fault tolerance and sane sequence of the tickets execution.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010.

[2] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *Proc. Fifth International Joint Conference on INC, IMS and IDC, NCM '09*, Washington, DC, USA. IEEE Computer Society, 2009, pp. 44-51.

[3] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud landscape," in *Proc. the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, Washington, DC, USA IEEE Computer Society, 2009, pp. 23-31.

[4] D. Sharma, S. K. Garg, and C. Sharma, "A cloud computing-based framework for internationalisation of SMEs," *International Journal of Cloud Computing*, vol. 2, no. 4, pp. 364-377, 2013.
[5] X. Xu, "From cloud computing to cloud manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 1, pp. 75-86, 2012.

Georgios Andreadis was born in 1959 in Serres, Greece. He is a graduate of the Mechanical Engineer Department of the Aristotle University of Thessaloniki, Greece. He works as a lecturer in the Mechanical Engineer Department since 2002, specialised in web based CAD/CAM, mechanical design. He is also the instructor of educative seminars and the author of several papers published in various conferences and magazines.

Konstantinos Stamos was born in 1983 in Thessaloniki, Greece. In 2005 he graduated from Computer Science Department at Aristotle University of Thessaloniki, Greece. He received his Msc in 2007 at the same faculty and his PhD in 2010. His main research interests are focused on web data management, content distribution and delivery over the web, web data clustering, replication and web data caching.