Machine Translation System as Virtual Appliance: For Scalable Service Deployment on Cloud

by

Pawan Kumar, Rashid Ahmad, Chaudhary B. D., Rajeev Sangal

in

The 7th IEEE International Symposium on Service-Oriented System Engineering (IEEESOSE 2013)

Report No: IIIT/TR/2013/-1

Centre for Language Technologies Research Centre
International Institute of Information Technology
Hyderabad - 500 032, INDIA
March 2013
Abstract—Machine Translation (MT) system is a complex natural language processing (NLP) system composed of large number of heterogeneous modules. Deployment of such a complex system even on a stand alone system is a cumbersome, knowledge intensive and time consuming task, taking hours to load, configure and run the system. As an MT system goes through frequent and regular updates, mainly to improve its accuracy and performance, the cumbersome task of its deployment is required to be repeated on release of each new version. Further, when such a system is needed to be deployed on a cloud infrastructure, mainly to facilitate auto-scaling of computational resources for varying load conditions, the task of deployment gets even more complicated and more time consuming.

This paper proposes that every software version of a complex NLP application like MT system should be built and released as a virtual appliance that can be deployed with a very little setup time and with ease even by a common user. It discusses the experiments performed to build the MT system into a virtual appliance, for stand alone system deployment as well as for cloud deployment, and reports the deployment time measurements in both the scenario. Deployment of the virtual MT appliance took 130 seconds in stand alone system; its deployment on a large number of virtual machines in the cloud environment took 150 seconds on an average, in contrast to several hours taken for the deployment of MT applications earlier.

Keywords — Deployment; Machine Translation; NLP Application; Virtual Appliance; Cloud; Auto scaling.

I. INTRODUCTION

Most of the NLP applications like Machine Translations (MT) Systems in general are composed of large number of modules, that are heterogeneous in nature, and these heterogeneous modules in turn depend upon complex set of environmental dependencies to perform a given task. To resolve such complex environmental dependencies at the time of software deployment is a hard task; it is also technical intensive and time consuming; additionally it is undesirable too.

Software deployment is defined as the process between the acquisition, and execution of the software. This process is performed as the post-development activity that takes care of user-centric customization and configuration of the software systems. At times this process can be quite complex, and may need the involvement and expertise of the developers and the system administrators quite extensively. Apart from technical complexity, the deployment tasks may be time consuming (of the order of hours). It is found [2, 5] that in general, 19% of total cost of operation (TCO) of a software system goes in deployment cost. As an MT system is far more complex and technical intensive it is fair to expect that its TCO must be far higher.

Unlike generic applications, an NLP application like MT system goes through frequent and regular updates, mainly to improve its accuracy and performance, and also to increase the coverage of its domain. Every new release of MT system requires fresh deployment of the new version from scratch, aggravating the technical administration distress, and in turn, inflating the total cost of operation far higher.

Furthermore, response of an MT system becomes exponentially slow with growing load. Scaling up of computation resources with growing load, mainly to provide users with better response time requires additional financial commitment from the service provider which cannot be done on-the-fly. Cloud infrastructure offered by third party, where computation resources can be scaled-on-the-fly, seems to be the most appropriate platform for offering service of such types of applications. Deploying an application on cloud infrastructure, in compared to a stand alone system, is far more complex, technical intensive, and time consuming. And hence, the total cost of deployment for such applications becomes even more significant when it is to be deployed on the cloud.

For a complex and technical intensive application like an MT system which is to be distributed to lay users, and which may have frequent software updates, the need to minimize the deployment time and to diminish its technical complexity become imperative.
To satisfy these non-functional requirements this paper proposes that complex NLP applications like MT systems should be packaged and released as virtual appliance [2, 25]. An application as a virtual appliance can just be taken out of the box by a common user to deploy easily on his machine, and with a very little setup time the application system becomes ready to use from that instant onwards. Though packaging an application as a virtual appliance do take time and is also technically intricate, but once it is built, its deployment can be done even by a common user, and takes very little time as the application’s complexity and its technical intricacy have been made transparent in its virtual appliance incarnation. A virtual appliance packaged either for stand-alone system or for cloud is expected to give proportional deployment advantages.

This paper reports the experimental time measurement results for software deployment of MT virtual appliance in relation to MT application. The results are reported for standalone system as well as for cloud platform.

II. THE SAMPAK MACHINE TRANSLATION SYSTEM

Sampark MT [8] is a machine translation system for translation of written text from one Indian language to another Indian language (developed for 9 bidirectional language pairs). Each Sampark MT system is typically composed of 15-20 modules depending on the specificity of the language pair. These modules in turn depend upon complex set of environmental dependencies. These dependencies could be due to the large number of language libraries used by the MT modules at runtime, and operating system (OS) and associated other third party tools. All these dependencies are linked at runtime. Sampark MT systems are comparatively more complex due to their inherent heterogeneous nature. This heterogeneity [4, 10, 11] is due to variations (in computation model, programming language, module interfaces, etc.) of the various MT modules.

Apart from resolving the dependencies, the inherent heterogeneous nature of the NLP/MT modules further adds to the complexity of the MT system. Each MT system needs to be configured for specific language-pair and for a designated translation domain. A typical deployment process of Sampark MT system (composed of building, configuring and verification tasks) takes some where between two to three hours time, depending on the language pair. Many a times, on a release of a new MT version, this process can be quite complex and may need the involvement of the developer/computational linguists to resolve its deployment issues. It usually requires the expertise of the MT application as well as the underlying systems used by the MT application.

Sampark system has been built by a consortium [10] of eleven research institutions. Each group is continuously working on its module to improve the accuracy (and domain coverage) and performance of the system. So the updates to the MT systems are very frequent. To handle these frequent updates in a large number of deployed systems, which is a routine task, is highly technical intensive, and is usually very time consuming.

Sampark MT systems are also deployed on the web, and people are regularly using it. It was found that as the web traffic increased the performance of the system (response time) for each translation deteriorated sharply. Various options available for improving the response time were experimented, viz.,

- Refactoring MT modules to keep the file I/O minimum in production system[10],
- Incorporation of platform cache for compute intensive modules [9],
- Distributing the translation tasks among the cluster of machines running MT systems, work load partitioning using Hadoop framework as middleware [12], and
- Deploying MT system on the cloud using Eucalyptus

With the above experiments, it was possible to considerably improve the response time of the Sampark MT system. With provisioning of additional hardware resources, it was possible to keep the response time within optimum limits as the load increased. The Sampark MT system was able to scale with provisioning of additional resources [12].

To translate a book of seventy pages, the Sampark MT system on a stand alone system took 71 minutes and 25 seconds, while on Eucalyptus Cloud environment with twelve (12) virtual machines (VMs) it took only 5 minutes and 27 seconds.

The objective of our initial experiment of deploying Sampark MT system, on cloud environment with provisioning of larger computation resources, was to verify its scalability and see improvement in the response time. But it was realized that for effective cloud deployment, optimum resource utilization is possible only, if the application is able to scale-up and scale-down rapidly, i.e., to scale-up or scale-down in real time.

In our case, the MT system was unable to scale-up (or scale-down) rapidly as it was taking very long time to deploy new instances of MT on additional virtual machines acquired dynamically on the cloud. Once the additional resources were provisioned, the system remained underutilized if the load decreased rapidly. While the MT system in principal was scalable, it remained either in the state of over-provisioned (i.e., resources remained underutilized) or under-provisioned (i.e., starved for resources). The large deployment time of the MT system was the main reason for its inability to rapid scaling up/down, making it completely unsuitable for cloud deployment.

In the early stage of improving the deployment process, we built a deployment script that would setup and configure the MT system with little intervention from the user. This script improved the speed of deployment, it came down from several hours to minutes (some where between 20 to 30 minutes depending upon the expertise of the user), but it still required the user to provide complex details of the environment like, OS details, path of various language libraries, and other third party tools required to run the various modules of the MT system. The complexity of the deployment process remained unsolved. Moreover, in spite of the deployment script, deployment process largely remained manual.
Our paper mainly illustrates how to build MT as a virtual appliance that can be deployed on stand alone VM. The same MT virtual appliance can be deployed on cloud, and can be scaled-up or scaled-down in-time.

III. VIRTUAL APPLIANCE: RELATED WORK

A virtual appliance [2, 7, 19, 20, 21, 22] is a full application stack containing the Just enough Operating System (JeOS), the application software, their required dependencies, and the configuration and data files required to run the system. Everything is pre-integrated, pre-installed, and pre-configured to run on a virtual machine, with minimum or no manual intervention [4]. A virtual appliance comes in the form of a data file that can be easily deployed on the cloud as well. Virtualization not only increases the mobility of application software, it also reduces deployment time considerably [23].

For cloud environment, virtualization can greatly simplify dynamic server provisioning as each virtual machine is contained in a single file. As a result, virtual machines can be easily cloned (copied to create additional images) and rapidly deployed using virtualization in-time.

In the past many tools were designed to ease systems/service deployment. Dearle [3] studied six cases of software deployment technologies and gave some of the future directions about the impact of virtualizations. The benefits of virtualization and virtual machines are discussed in [19, 20, 21]. Sapuntzakis et al. [1] proposed Collective, a compute utility using virtual appliances to manage systems. In the past, software vendors have assembled the software applications, operating systems, and the middleware (if any) as virtual appliance (or as software appliance) and have distributed as ready-to-run application stack [18] that boots in a setup wizard. Most of the time, virtual appliances have been built to ease the distribution of software.

In the past people have used virtual appliance to improving software manageability and automate provisioning [14]. Deploying applications in the cloud allows scaling on demand, and provides benefits of elasticity and transference of risks, especially the risks of over-provisioning and under-provisioning of resources [13]. The key benefit of enabling the application for cloud deployment as virtual appliance is to add or remove computational resources with fine granularity and with a lead time in the granularity of few seconds/minutes rather than hours.

In contrast we have experimented to build MT system as virtual appliance to facilitate deployment on stand alone system as well as for cloud environment. Additionally, this virtual appliance also enables the MT system to scale-in time when deployed in the cloud.

IV. BUILDING SAM PARK MT SYSTEM AS A VIRTUAL APPLIANCE

A virtual appliance can run on a stand alone virtual machine or on virtual machines made available in the cloud. To build the Sampark MT system as a virtual appliance and deploy it on to a cloud we used following tools:

- CentOS 5.3, a variant of Linux, as the guest operating system,
- Xen [28] for virtualization of the hardware,
- CentOS 5.7, a variant of Linux, as the Host Operating System for virtualization,
- Hadoop [26] Version 0.20.2 as the middleware for work load partitioning, and
- Eucalyptus [27] Version 2.0.3 for setting up the cloud infrastructure.

Please refer Annexure I, for clear understanding of various terms used in this section, in building the Sampark MT system as a virtual appliance and in deploying the same, in the Eucalyptus cloud.

In our experiment setup, we had

- Each PC - a Quad-core 2.5 GHz Intel processor, 4 GB RAM, and 2MB of L2 cache,
- Each Virtual Machine in the cloud had: 2 CPU with 1GB RAM each,
- In the Eucalyptus Cloud – a total of 12 Xen Virtual Machines.

The Eucalyptus cloud is setup using Xen as virtualization platform. An instance of the default image of CentOS 5.3 is instantiated on the available VM. It takes approximately 60 seconds to boot the virtual image whose image size is 1GB on our hardware as mentioned above.

Once the VM is ready and the guest OS is running, Sampark MT system along with its dependencies is installed on the VM. Next, middleware Hadoop, is installed in the system. Then the Sampark MT along with the OS and Hadoop is rebased. This current image is bundled along with kernel and ramdisk by the command euca-bundle-vol. This bundled image is uploaded at the base machine by command euca-upload-bundle, and lastly image is registered by command euca-register to make sure it is available for launch. Now the rebased image is ready for deployment as virtual appliance on the available Xen VMs.

Deployment of this Sampark MT virtual appliance on a virtual machine takes approximately 150 seconds (on our hardware), which is well within the desirable limit. Image size of this Sampark MT virtual appliance is 4GB.

V. PROVISIONING POLICY FOR AUTOSCALING SAM PARK MT SYSTEM

Policy for provisioning of additional computational resources in the cloud for auto-scaling the Sampark MT system is based on

- CPU load averages over a defined duration (in our case it was 60 seconds), and
- Memory usage (average over a defined period).

Since we are using Linux as the virtual appliances OS, we need to configure the baseline-images for monitoring the
instances. Linux utility *sysstat* is bundled as a part of the virtual appliance. On each instance, a load monitoring script (using *sar* command of Linux) runs as a daemon. It monitors the CPU utilization of each of the instances and logs utilization data into a data file in binary form (which can be later read by *sar* command).

To implement *auto-scaling* of computational resources we built a provisioning script *mpp* (Monitor & Provision) to provision additional instances of VMs, if required. This script *mpp*, runs on the Cluster Controller (CC) and lists the IP addresses of the running instances by the command *euca-describe-instances*. For each of these running instances, *mpp* script collects the CPU utilization by looking to the data file generated by the *sar* utility, using remote *ssh*. If the average of all instances cross a preset threshold (*provisioning policy*), new instances of the MT systems are started/instantiated on the available VM using command *euca-run-instances*.

Conversely when the CPU utilization goes below a specific threshold same script *mpp* also terminates the instances by command *euca-terminate-instances*.

In addition to the above tasks, while scaling-up, this *mpp* script also informs the Hadoop master (*NameNode*) about the addition of new *DataNodes* in the Hadoop cluster, a new record is added to the *conf/slaves* file of the Hadoop master server. While scaling-down, *mpp* script drops a *DataNode* from the cluster, by adding a new record to the exclude file in the *NameNodes* local file system, and a command “*bin/hadoop dfsadmin -refreshNodes*” is sent to *NameNode* to decommission the *DataNode*.

The auto-scaling script is configurable, and in our case it took 3 minutes to scale up the MT application after provisioning of the new resources in the cloud.

**VI. Conclusion**

In this paper we have proposed and implemented how a complex NLP application like MT system which would be mostly used by non-technical users should be packaged into a virtual appliance. Virtual appliance simplifies the task of software deployment significantly and brings down the deployment time to minutes rather than hours. In our case deployment of Sampark MT virtual appliance took approximately 130 seconds on a commodity hardware.

The MT virtual appliance can be deployed on the virtual machines in the cloud as well, with significant reduction in the deployment time. Deployment time in cloud came down from several hours to few minutes, in our case it was on an average 150 seconds. With the addition of simple provisioning script MT application is able to autoscale in-time (3 minutes after provisioning of compute resources) for varying load conditions in the cloud.

In future we also intend to enhance this virtual appliance so that it is available as repositories and from there it can be deployed in the cloud that would enable MT system to handle the frequent updates as well.

**ACKNOWLEDGMENT**

We would like to thank Dr. Mukul K Sinha for his guidance to setup the experiments, and conclude the usefulness of the observations thereafter. We would like to thank IIIT-Hyderabad, for allowing us to carry out the experiments in their lab.

**Annexure-I: Terminology**

**Linux Commands:**
- *sysstat*: A Linux utility for collecting system status
- *sar*: System Activity Report for collecting CPU, Memory, and I/O load

**Eucalyptus Cloud:**
- *image*: An image is a snapshot of a system's root file system and it provides the basis for instances
- *baseline-image*: baseline-image usually includes the optimum operating system, any required service packs, a standard set of application (that is to be virtualized), other underlying tools required by the application, and the necessary patches if any, or loosely speaking snapshot of the root file system with the running application

**euca2ools**: Image management commands for Eucalyptus Cloud
- *euca-bundle-vol*: Bundle the local file system of a running instance along with kernel and ramdisk
- *euca-upload-bundle*: Upload the bundled image to the cloud
- *euca-register*: Register an image for use with the cloud
- *euca-describe-instances*: To check the status of the image instance in the cloud
- *euca-run-instance*: To start a new instance of the image in the cloud
- *euca-terminate-instance*: To terminate an instance of image from the cloud

**Hadoop**: MapReduce Terms
- *NameNode*: Hadoop master node, the machine where the computation task is actually submitted, *NameNode* partitions a work load among the available *DataNodes*, and schedules the computation task to each of these data nodes
- *DataNode*: Hadoop worker nodes, on which the tasks are actually computed
- *hadoop*: Hadoop command interpreter that actually executes all the commands for the Hadoop cluster

**References**


...