Cloud systems for environmental telemetry - A case study for ecological monitoring in agriculture

George Suciu[1], Octavian Fratu[1], Cristian Cernat[1], Traian Militaru[1], Gyorgy Todoran[1] and Vlad Poenaru[1]

Abstract – **Large telemetry systems have several hundreds of RTUs that are sending data to be processed by intelligence algorithms and stored in a database that is accessible via Web interface on the Internet. In this paper we present the way in which SlapOS, an open source provisioning and billing system for distributed cloud computing, is used to gather centrally environmental information from different sensors at remote observation points.**

Keywords – **Cloud, Telemetry, Sensors, Remote monitoring, RTU.**

I. Introduction

In this paper we develop a test platform for environmental telemetry and use it as a case study for monitoring ecological parameters in agriculture. We use different types of RTUs (Radio Transmission Units) and Sensors that monitor and transmit important information such as temperature, precipitation, wind speed and leaf wetness from selected locations.

The RTUs will transmit sensor data over GSM/GPRS to our cloud platform where we can conveniently process the site-specific weather and soil data in near real-time, display it in our web-based visualization application and get detailed recommendations when and where to spray and how much to irrigate - resulting in optimized yield, quality and income.

Our system can also help keeping track of pathogen development, optimize treatments to hit a disease dead on, warn of frost, and to produce crops as environmentally friendly as possible and to improve agricultural risk management. Falling producer prices and rising costs of production are increasingly forcing agricultural businesses to optimize production costs [1]. Therefore "precision farming", the selective use of inputs such as water, fertilizers or chemicals, is now indispensable in modern agriculture. The growing environmental awareness of consumers further accelerates this process and promotes the usage of remote automatic monitoring system for field information such as the one we developed [2].

We will introduce in this article SlapOS, the first open source operating system for Distributed Cloud Computing. SlapOS is based on a grid computing daemon called slapgrid which is capable of installing any software on a PC and instantiate any number of processes of potentially infinite duration of any installed software. Slapgrid daemon receives requests from a central scheduler the SlapOS Master which collects back accounting information from each process. SlapOS Master follows an Enterprise Resource Planning (ERP) model to handle at the same time process allocation optimization and billing. SLAP stands for “Simple Language for Accounting and Provisioning”.

This structure has been implemented for cloud-based automation of ERP and CRM software for small businesses and aspects are under development under the framework of the European research project “Cloud Consulting” [3]. We will use our platform hosted on several servers running Ubuntu Linux – Apache – MySQL template with current software release. On our cloud testing environment we provide the platform for processing information from hundreds different sensors, enabling the analysis of environmental data through a large sample of RTUs.

In previous approaches RTUs were implemented in most cases on a local server and no company could aggregate enough sensor data to consider automating the treatment process.

II. Cloud Architecture For Telemetry

A. Cloud Architecture

SlapOS is an open source Cloud Operating system which was inspired by recent research in Grid Computing and in particular by BonjourGrid [4]–[5] a meta Desktop Grid middleware for the coordination of multiple instances of Desktop Grid middleware. It is based on the motto that ”everything is a process”.

SlapOS is based on a Master and Slave design. In this chapter we are going to provide an overview of SlapOS architecture and are going in particular to explain the role of Master node and Slave nodes, as well as the software components which they rely on to operate a distributed cloud for telemetry applications.

Slave nodes request to Master nodes which software they should install, which software they show run and report to Master node how much resources each running software has been using for a certain period of time. Master nodes keep track of available slave node capacity and available software. Master node also acts as a Web portal and Web service so that end users and software bots can request software instances which are instantiated and run on Slave nodes. Master nodes are stateful. Slave nodes are stateless. More precisely, all information required to rebuild a Slave node is stored in the Master node. This may include the URL of a backup service which keeps an online copy of data so that in case of failure of

[1]The authors are with the Faculty of Electronics, Telecommunications and Information Technology at Politehnica University of Bucharest, Bd. Iuliu Maniu, nr. 1-3, Bucharest 060042, Romania, E-mails: george@beia.ro, ofratu@elcom.pub.ro, cernatcristi@gmail.com, gelmosro@yahoo.com, todoran.gyorgy@gmail.com, vlad.wing@gmail.com.

a Slave node, a replacement Slave node can be rebuilt with the same data.

It is thus very important to make sure that the state data present in Master node is well protected. This could be implemented by hosting Master node on a trusted IaaS infrastructure with redundant resource. Or - better - by hosting multiple Master nodes on many Slave nodes located in different regions of the world thanks to appropriate data redundancy heuristic. We are touching here the first reflexive nature of SlapOS. A SlapOS master is normally a running instance of SlapOS Master software instantiated on a collection of Slave nodes which, together, form a trusted hosting infrastructure. In other terms, SlapOS is self-hosted, as seen in Fig. 1.

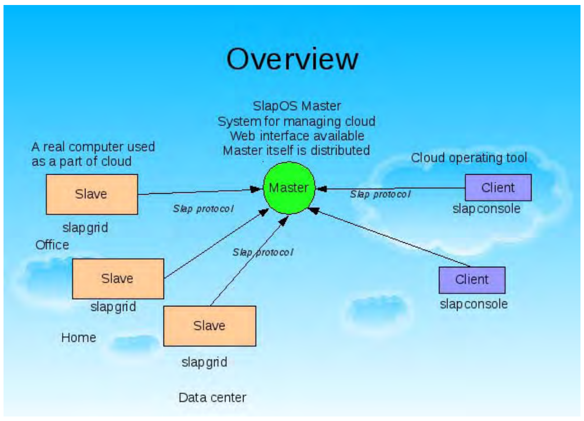

Fig. 1. SlapOS Master – Slave Architecture

B. SlapOS Master

SlapOS master nodes keep track of the identity of all parties which are involved in the process of requesting Cloud resources, accounting Cloud resources and billing Cloud resources. This includes end users (Person) and their company (Organisation). It includes suppliers of cloud resources as well as consumers of cloud resources. It also includes so-called computer partitions which may run a software robot to request Cloud resources without human intervention. It also includes Slave nodes which need to request to SlapOS master which resources should be allocated. SlapOS generated X509 certificates for each type of identity: X509 certificates for people like you and me who login, an X509 certificate for each server which contributes to the resources of SlapOS and an X509 for each running software instance which may need to request or notify SlapOS master. A SlapOS Master node with a single Slave node, a single user and 10 computer partitions will thus generate up to 12 X509 certificates: one for the slave, one for the user and 10 for computer partitions.

Any user, software or slave node with an X509 certificate may request resources to SlapOS Master node. SlapOS Master node plays here the same role as the backoffice of a marketplace. Each allocation request is recorded in SlapOS Master node as if it were a resource trading contract in which a resource consumer requests a given resource under certain conditions. The resource can be a NoSQL storage, a virtual machine, an ERP with web-portal interface for displaying sensor data and Google Maps integration for RTUs localization, a Wiki, etc. The conditions can include price, region (ex. China) or specific hardware (ex. 64 bit CPU). Conditions are somehow called Service Level Agreements (SLA) in other architectures but they are considered here rather as trading specifications than guarantees. It is even possible to specify a given computer rather than relying on the automated marketplace logic of SlapOS Master.

By default, SlapOS Master acts as an automatic marketplace. Requests are processed by trying to find a Slave node which meets all conditions which were specified. SlapOS thus needs to know which resources are available at a given time, at which price and under which characteristics. Last, SlapOS Master also needs to know which software can be installed on which Slave node and under which conditions.

C. SlapOS Slave

SlapOS Slave nodes are relatively simple compared to the Master node. Every slave node needs to run software requested by the Master node. It is thus on the Slave nodes that software is installed. To save disk space, Slave nodes only install the software which they really need.

Each slave node is divided into a certain number of so-called computer partitions. One may view a computer partition as a lightweight secure container, based on Unix users and directories rather than on virtualization. A typical barebone PC can easily provide 100 computer partitions and can thus run 100 RTU web portals or 100 sensors monitoring sites, each of which with its own independent database. A larger server can contain 200 to 500 computer partitions.

SlapOS approach of computer partitions was designed to reduce costs drastically compared to approaches based on a disk images and virtualization. As presented in Fig. 2, it does not prevent from running virtualization software inside a computer partition, which makes SlapOS at the same time cost efficient and compatible with legacy software.

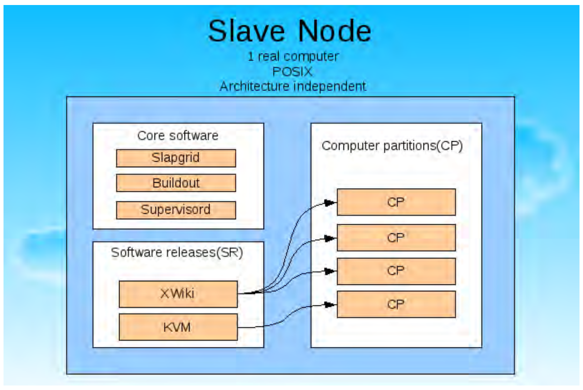

Fig. 2. SlapOS Slave Node

SlapOS Slave software consists of a POSIX operating system, SlapGRID, supervisord and buildout [3]. SlapOS is designed to run on any operating system which supports GNU's glibc and supervisord. Such operating systems include for example GNU/Linux, FreeBSD, MacOS/X, Solaris, AIX, etc

D. SlapOS kernel

SlapOS relies on mature software: buildout and supervisord. Both software are controlled by SLAPGrid, the only original software of SlapOS. SLAPGrid acts as a glue between SlapOS Master node (ERP5) and both buildout and supervisord, as shown in Fig. 3. SLAPGrid requests to SlapOS Master Node which software should be installed and executed. SLAPGrid uses buildout to install software and supervisord to start and stop software processes. SLAPGrid also collects accounting data produced by each running software and sends it back to SlapOS Master.

Supervisord is a process control daemon. It can be used to programmatically start and stop processes with different users, handle their output, their log files, their errors, etc. It is a kind of much improved init.d which can be remotely controlled. Supervisord is lightweight and old enough to be really mature (ie. no memory leaks).

Buildout is a Python-based build system for creating, assembling and deploying applications from multiple parts, some of which may be non-Python-based. Buildout can be used to build C, C++, ruby, java, perl, etc. software on Linux, MacOS, Windows, etc. Buildout can either build applications by downloading their source code from source repositories (subversion, git, mercurial, etc.) or by downloading binaries from package repositories (rpm, deb, eggs, gems, war, etc.). Buildout excels in particular at building applications in a way which is operating system agnostic and to automate application configuration process in a reproducible way.

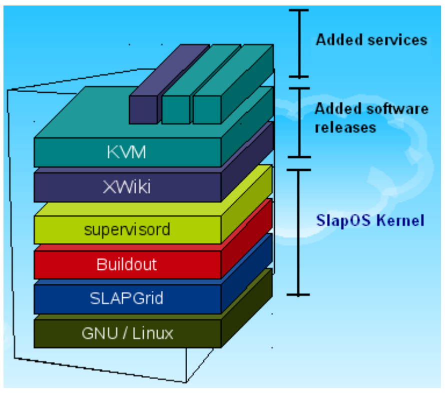

Fig. 3. SlapOS Kernel and User Software

Every computer partition consists of a dedicated IPv6 address, a dedicated local IPv4 address, a dedicated tap interface (slaptapN), a dedicated user (slapuserN) and a dedicated directory (/srv/slapgrid/slappartN). Optionally, a dedicated block device and routable IPv4 address can be defined.

SlapOS is usually configured to use IPv6 addresses. Although use of IPv6 is not a requirement (an IPv4 only SlapOS deployment is possible) it is a strong recommendation. IPv6 simplifies greatly the deployment of SlapOS either for public Cloud applications or for private Cloud applications. In the case of public Clouds, use of IPv6 helps interconnecting SlapOS Slave Nodes hosted at home without having to setup tunnels or complex port redirections. In the case of private Cloud, IPv6 replaces existing corporate tunnels with a more resilient protocol which provides also a wider and flat corporate addressing space. IPv6 addressing helps allocating hundreds of IPv6 addresses on a single server. Each running process can thus be attached to a different IPv6 address, without having to change its default port settings. Accounting network traffic per computer partition is simplified. All this would of course be possible with IPv4 or through VPNs but it would be much more difficult or less resilient. The exhaustion of IPv4 addresses prevents in practice allocation of some many public IPv4 addresses to a single computer. After one year of experimentation with IPv6 in Romania, using IPv6 native Internet access (more than 50% of worldwide IPv6 traffic), we found that IPv6 is simple to use and creates the condition for many innovations which would else be impossible.

III. Research Results And Future Design

In order to collect the information from the RTUs we developed the following test platform as shown in Fig. 4. The usage of GSM/GPRS data transmission can be extended in areas where there is no coverage by using a UHF bridge operating in the fixed frequency range 430 – 440 MHz connected to a gateway that has access to the Internet.

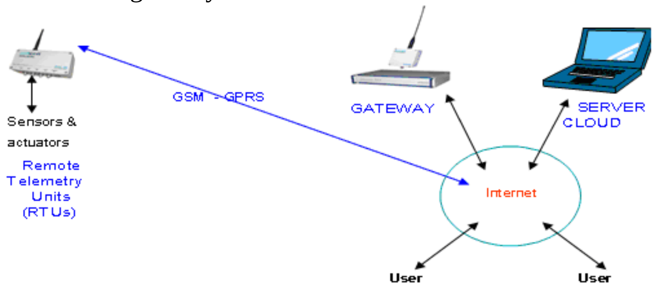

Fig. 4. General Architecture of Telemetry System

The case study was done on 2 grape yards in Romania (Bucharest and Blaj) with the following sensors, as seen in Fig. 5

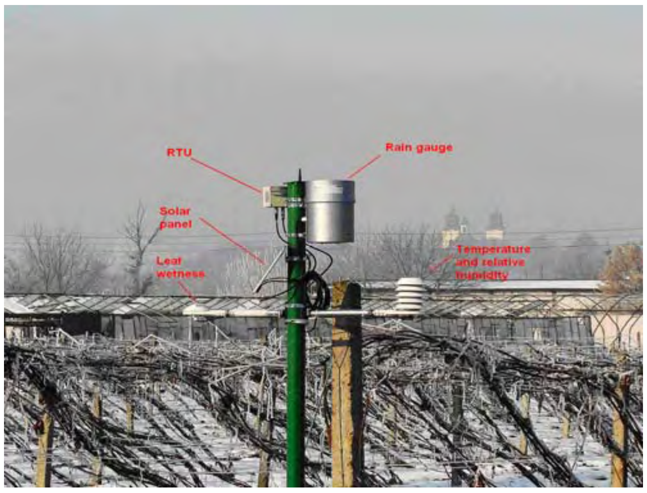

Fig. 5. General Structure of RTU and Sensors

The total quantity of rain reported by the system during the months of May – September 2011 was of 222 l/sqm with the following monthly distribution: May – 33 l/sqm, June – 116 l/sqm, July – 49 l/sqm, August - 5 l/sqm, September – 8 l/sqm. Other climatic parameters such as Precipitation, Leaf Wetness, Temperature and Relative Humidity can be seen on Fig. 6.

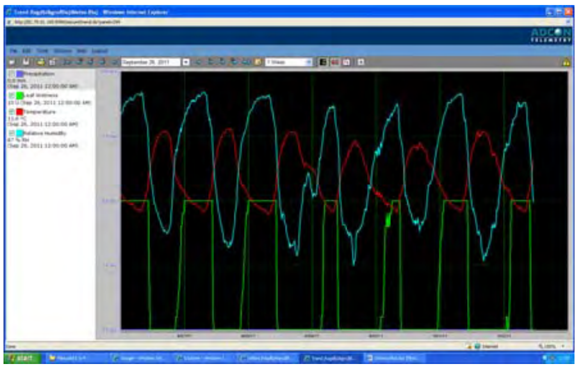

Fig. 6. Results climatic parameters during a week (26.09 – 02.10.2011)

Another important parameter we studied is the accumulation of thermal energy over time, known as degree-days or heat units. The growth and development of plants, insects, and many other invertebrate organisms is largely dependent on temperature. In other words, a constant amount of thermal energy is required for the growth and development of many organisms, but the time period over which that thermal energy is accumulated can vary. Many organisms slow or stop their growth and development when temperatures are above or below threshold levels. Degree-days and other heat unit measurements have been used for determination of planting dates, prediction of harvest dates, and selection of appropriate crop varieties.

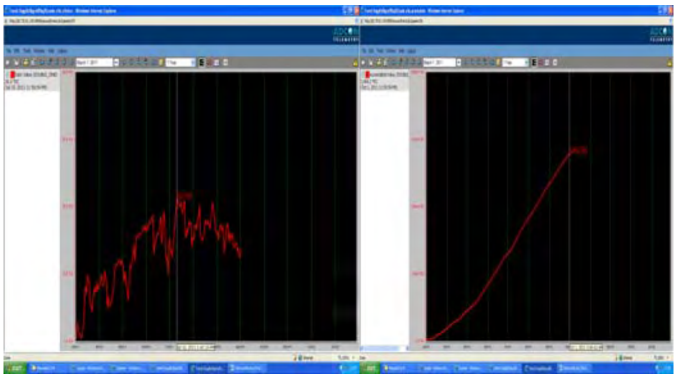

Fig. 7. Heat units graphs (daily degree-days and total)

The calculation methods available for heat unit include: Averaging, Standard, GDD (Growing Degree-Days), Single Triangle, Double Triangle, Single Sine, Double Sine and Near Real-Time. As shown in Fig. 7 we used the Averaging Method and the maximum heat unit (26,3 degree days) was calculated on the date of 10.07.2011 and the total accumulated thermal energy by the crop on 01.10.2011 was 3.484,2 degree-days.

IV. Conclusion

Our system for environmental telemetry can be adapted also to other applications besides agriculture and meteorology. Knowing how the weather will be is important but knowing how the environmental parameters are right now is just as much important for power plants, airports, wind and solar parks, incinerators and landfills - they all need wind, temperature, radiation data, etc. reliably and up to date.

Even though IPv6 is used to interconnect processes globally on a SlapOS public or private Cloud, we found that some existing software on RTUs are incompatible with IPv6. Reasons varry. Sometimes, IP addresses are stored in a structure of 3 integers, which is incompatible with IPv6. Sometimes, IPv6 URLs are not recognized since only dot is recognized as a separator in IP addresses. For this reason, we decided to provide to each computer partition a dedicated, local, non routable IPv4 address.

We hope in the future that Microsoft Windows will also be supported as a host (Microsoft Windows is already supported as a guest) through glibc implementation on Windows and a port of supervisord to Windows.

Acknowledgement

This paper is presented as part of the project *"Valorificarea capitalului uman din cercetare prin burse doctorale (ValueDoc)" Project co-financed from the European Social Fund through POSDRU, financing contract POSDRU/107/1.5/S/76909* and part of the *project "Cloud Consulting" and "TELE GREEN".*

References

[1] Tokihiro Fukatsu, Tomonari Watanabe, Haoming Hu, Hideo Yoichi, Masayuki Hirafuji, "Field monitoring support system for the occurrence of Leptocorisa chinensis Dallas (Hemiptera: Alydidae) using synthetic attractants, Field Servers, and image analysis", Computers and Electronics in Agriculture, vol. 80, January 2012, pp. 8–16, 2012

[2] Jiang, J.A.; Tseng, C.L.; Lu, F.M.; Yang, E.C.; Wu, Z.S.; Chen, C.P.; Lin, S.H.; Lin, K.C.; Liao, C.S., "A GSM-based remote wireless automatic monitoring system for field information: A case study for ecological monitoring of the oriental fruit fly, Bactrocera dorsalis (Hendel)", Computers and Electronics in Agriculture vol. 62, Issue 2, July 2008, pp. 243–259, 2008

[3] George Suciu, Octavian Fratu, Simona Halunga, Cristian George Cernat, Vlad Andrei Poenaru, Victor Suciu, "Cloud Consulting: ERP and Communication Application Integration in Open Source Cloud Systems", 19th Telecommunications Forum - TELFOR 2011, IEEE Communications Society, pp. 578-581, 2011

[4] Heithem Abbes, Christophe C´erin, and Mohamed Jemni.Bonjourgrid as a decentralised job scheduler. In APSCC 08.Proceedings of the 2008 IEEE Asia-Pacific Services ComputingConference, pages 89–94,Washington, DC, USA,2008. IEEE Computer Society.

[5] Heithem Abbes, Christophe C´erin, Mohamed Jemni: A decentralized and fault-tolerant Desktop Grid system for distributed applications. Concurrency and Computation: Practice and Experience 22(3): 261-277 (2010)