Church et al. Journal of Cloud Computing: Advances, Systems and Applications (2017) 6:11
DOI 10.1186/s13677-017-0080-5

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

Migration of a SCADA system to IaaS clouds – a case study

Philip Church[1,3*], Harald Mueller[2], Caspar Ryan[1], Spyridon V. Gogouvitis[2], Andrzej Goscinski[1,3] and Zahir Tari[1]

Abstract

SCADA systems allow users to monitor and/or control physical devices, processes, and events remotely and in real-time. As these systems are critical to industrial processes, they are often run on highly reliable and dedicated hardware. Moving these SCADA systems to an Infrastructure as a Service (IaaS) cloud allows for: cheaper deployments, system redundancy support, and increased uptime. The goal of this work was to present the results of our experimental study of moving/migrating a selected SCADA system to a cloud environment and present major lessons learned. To this end, EclipseSCADA was deployed to the NeCTAR research cloud using the "lift and shift" approach. Performance metrics of a unique nature and large scale of experimentation were collected from the deployed EclipseSCADA system under different loads to examine the effects cloud resources and public networks have on SCADA behavior.

Keyword: SCADA, IaaS cloud, Migration, Case study

Introduction

Supervisory Control and Data Acquisition (SCADA) systems are instrumental to a wide range of mission-critical industrial systems, from infrastructure installations like gas pipelines or water control facilities to industrial plants. SCADA systems allow a user to monitor (using sensors) and control (using actuators) an industrial system remotely. As these systems are critical to industrial processes, they are often run on highly reliable and dedicated hardware [1]. This is in contrast to the current trend and state of computing, which is moving from running applications on internally hosted servers to flexible, cheaper, nternal or external cloud systems.

For users, the main benefit of moving applications such as SCADA to the cloud lies in the potential cost savings and reduced setup time. Cloud resources are purchased and accessed on-demand, at a total cost of ownership cheaper than buying and operating hardware; furthermore, as there is no need to install and/or maintain hardware and manage software, the need for technical staff is reduced. For industry, running SCADA on the cloud can lead to new business models, instead of a traditional one-off hardware cost and software licensing fee. Cloud-based SCADA solutions can provide users with flexible fees based on the amount of computing resources, technical support, and software they use.

One way to provide a cloud-based SCADA system would be the cloud native approach, which means to develop it from scratch, ideally using an appropriate architectural design to make use of cloud inherent features. Another option is to take an existing SCADA implementation and migrate it to a cloud environment. Often this latter approach is an initial step towards the cloud, given that solid and well proven implementations are available.

When moving an open-source SCADA system to cloud infrastructure, there is a need to ensure that the real-time monitoring and control demands of the industrial system can be achieved. Based on a comparison of commercial and open-source SCADA features, EclipseSCADA[1] was selected as an example of a representative SCADA system. Using the "lift and shift" method, EclipseSCADA was migrated to the IaaS cloud, and performance metrics of a unique nature and large scale of experimentation collected. Based on the collected metrics, we provide a number of recommendations, which can benefit users planning to move SCADA systems to cloud infrastructure.

Through the carried out work, the following contributions have been made:

* Correspondence: philip.church@rmit.edu.au
[1]School of Computer Science and Information Technology, GPO, RMIT University, Box 2476, Melbourne, Australia
[3]School of Information Technology, Deakin University, Geelong, Australia
Full list of author information is available at the end of the article

- The presentation of a general migration process and performance metrics of a unique nature and large scale of experimentation collected, which can be applied to move SCADA systems to cloud resources.
- A detailed performance study of EclipseSCADA was carried out to identify how a SCADA system behaves when running across multiple regions of a cloud.
- The presentation of a series of lessons, which can be used to select SCADA parts that should be migrated and location of their deployment, their relationship, influence on execution performance, and even improve the performance of SCADA systems running on the cloud.

The rest of the paper is as follows: Section 3 introduces "generalized" SCADA architecture. Section 3 presents related work in running and migrating SCADA system on the cloud. Section 4 compares open-source SCADA systems with common commercial SCADA products in order to find a representative SCADA system to be migrated. Section 5 shows a case study – it describes the process used to develop and deploy the selected SCADA system (EclipseSCADA) to IaaS cloud resources (NeCTAR [2]). To identify problems regarding the real-time monitoring and control mechanism of SCADA systems when running on the cloud, an analysis of the ported EclipseSCADA system was carried out in Section 6. This section also contains an analysis of collected performance results and migrating lessons. Section 7 concludes the paper.

"Generalized" SCADA architecture

SCADA describes applications, which aim to control and monitor remote equipment via a communication channel [3]. There have been a number of attempts to form a generalized SCADA framework and architecture. Boye [4] defines a simple SCADA architecture that consists of sensors, switches and/or actuators (field devices), connected and read by a device server (see Fig. 1). Data is transferred across a network to a control server, which handles events (informing a user if sensor data exceeds set boundaries). A user of a SCADA system accesses data via the master server using a workstation.

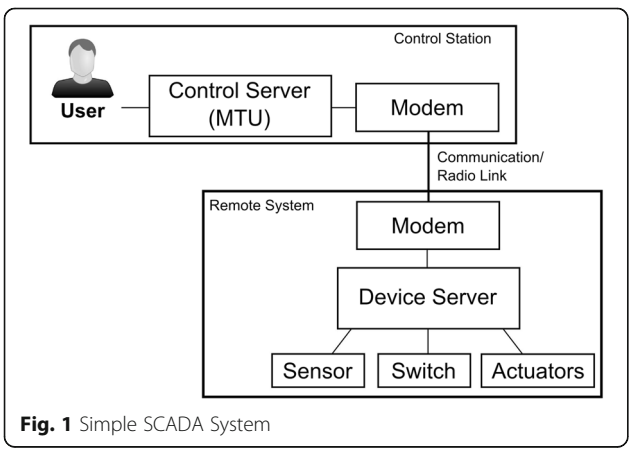

Fig. 1 Simple SCADA System

In contrast, IEEE [5] defines an in-depth standard describing the components that make up a SCADA framework. According to the IEEE standard, the system is divided into a remote site and master station (see Fig. 2). The remote site consists of field devices connected to a device server. Communication between the field device and device server makes use of a SCADA communication standard (used by the driver). Collected information is stored in a real-time and historical database on the master station. Communication between components of the master station uses an internal communication protocol. The Master Terminal Unit (MTU) contains a number of tools that interact with the data stored in the databases including:

- An event handler, which reacts to changes to the real time database;
- A device manager, which can modify the behaviour of field devices;
- An alarm manager, which allows users to setup monitoring rules and notify a user if rules have been broken;
- An archiver, which provides analytics of stored data; and
- A GUI or Human Machine Interface, which provides the user with a graphical representation of the remote site.

Every SCADA system uses a large number of field devices. A field device understands and collects data from sensors, switches and actuators. For this reason, field devices make use of a compute device called a Programmable Logic Controller (PLC), or Remote Terminal Unit (RTU) (see Fig. 3). Users can access the PLC or RTU through a network interface or a Human Machine Interface (HMI), allowing for configuration and access to connected sensors, switches or actuators. An inbuilt computer runs code which converts signals from connected sensors, switches or actuators to digital data, or vice versa. The code that runs on field devices falls under two categories: monitoring loops for sensors that may incorporate sampling/averaging, and state diagrams that control the state of output devices such as switches/actuators. Often this code has real time requirements, as it directly monitors and controls the connected field devices.

Field devices are designed to be reliable, often incorporating backup power and redundancy in the form of backup I/O cards. If a device is connected to an I/O card that fails, the device automatically gets connected to the redundant I/O. In the absence of this feature, if an input card fails, signals would be lost until the card is replaced.

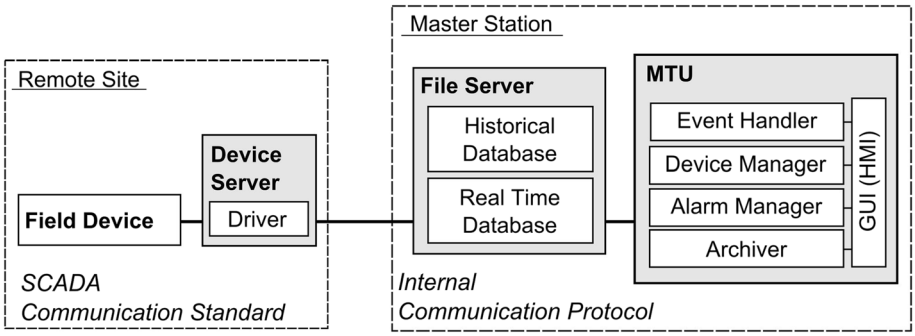

Fig. 2 General SCADA software architecture

Through the use of SCADA and Field Devices, it becomes possible to monitor and control large scale systems (such as gas pipelines, which cover very large distances) cheaply and efficiently.

To transfer data between a field device and SCADA system, communication protocols are used. Protocols are either Polling or Event Driven (see Fig. 4). Polling protocols transfer data on a timed loop, where the time between each transfer is called the polling interval. Event driven protocols transfer data on sensor change. Commonly used polling protocols include Modbus [6] and Profibus [7]. Commonly used event driven protocols include S7 [8] and iec104 [9].

Related work

There are a number of papers, which discuss how to build cloud-based SCADA architectures. They focus on implementing a solution from the ground up, as opposed to utilizing pre-existing SCADA solutions.

- Liu, et al. presents a generalized overview of clouds and SCADA and proposes the possibility of running SCADA in the cloud [10].
- Gligor and Turc recommend exposing each SCADA component as a service and deploying them through a Local Directory Service (LDS) [11]. The LDS stores a description of available SCADA resources, access methods, and description. The use of a broker allows some components can be replaced by cloud services; for example the database service can utilize Data Center as a Service (DaaS). This approach is very flexible; allowing users to extending the SCADA system by adding new functionalities to existing services or defines new ones in accordance with needs and formulated requirements.
  Based on these concepts, a web-based SCADA system is implemented on Rackspace cloud resources. Data was transferred using a simple protocol, consisting of a few numerical and process monitoring variables. Data transfer rates were measured from a local database to a cloud database, results measured were between 125 to 156 ms.
- Goose et al. present a secure SCADA cloud framework called SKYDA [12]. This SCADA system is designed to take advantage of the scalability and reliability offered by a cloud-based infrastructure. This paper focuses on providing a high level understanding of SCADA replication using clouds, moving all SCADA components (except the field devices) as a single service. Field devices are connected to the cloud

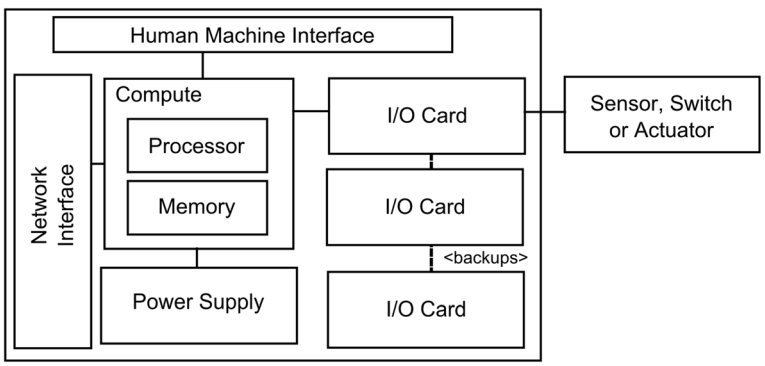

Fig. 3 Field Device Architecture

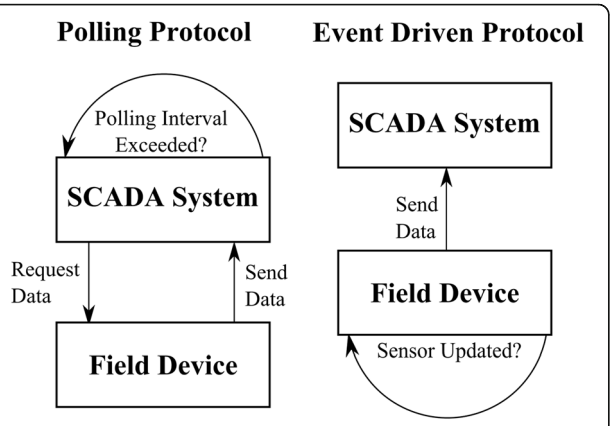

Fig. 4 Polling vs. Event Driven Protocol

based SCADA system directly or (for legacy devices) via a proxy. The framework utilizes multiple cloud providers, running multiple copies of the SCADA Master application in multiple clouds to provide fault tolerance.

There also are solutions provided by commercial cloud-based SCADA providers; the two major commercial solutions are Ignition and XiO's Cloud SCADA.

- Ignition SCADA is a SCADA solution that has been built from the ground up using Java to take advantage of cloud features [13]. Ignition interfaces with most Programmable Logic Controllers (PLC), allowing users to take advantage of existing sensors/actuators. Ignition users do not maintain hardware themselves; instead they access systems remotely via web interfaces. Users are charged based on the number of servers used instead of via software licensing fees. Ignition allows users to customize their architecture by choosing to deploy components individually.
- XiO Cloud SCADA [14] consists of two components, a local (customizable) hardware module called a Soft-I/O, which contains the sensors and actuators, and the cloud component, the SCADA application which runs on secure commercial servers. Users subscribe to a Cloud Service, for a monthly fee, giving them access to the SCADA system through web and mobile apps. Users can customize the priorities of their XiO SCADA system, for example to priorities energy efficiency.

In general, there is a trend to develop SCADA systems specifically for cloud infrastructure. The fact that existing SCADA solutions have not been used alludes to potential issues with migrating SCADA solutions. However, it is not clear if issues are performance or deployment related. The solution presented by Gligor and Turc addresses performance through the use of a simple data transfer protocol, while the local directory service addresses deployment issues. The SKYDA cloud framework only addresses deployment issues through the use of automated replication.

Common open-source SCADA solutions include EPICS, TANGO, EclipseSCADA and IndigoSCADA. EPICS (Experimental Physics and Industry Control System) [15] is a SCADA system designed to operate devices such as particle accelerators, large experiments, and major telescopes. TANGO [16] is an object orientated distributed control system supported by a consortium of European Synchrotrons in Germany, Spain, Italy, Poland, and France. EclipseSCADA [17] is a key eclipse foundation project used commercially, the details of which have not been made public. IndigoSCADA [18] is a light weight SCADA system for Linux and Windows. While some of these solutions have been run on cloud infrastructure, no major performance study has been carried out on how these solutions behave. This paper focuses on the process of migrating and understanding, based on the nature and scale of experimentation undertaken, the feasibility and outcomes of migrating existing open SCADA solutions to run on cloud infrastructure.

Open-source SCADA selection

It was the goal to ensure that the results of this migration study would be able to be applied to a wide range of open and commercial systems. Therefore there was a need to select an open-source SCADA package which would: provide similar architecture and features to commercial packages, have manageable code with minimal coupling between components, and be widely adopted and supported.

Most commercial solutions focus on detailing features of their product rather than the system architecture, which is making comparisons difficult. One commercial SCADA package which provides a detailed description of the system architecture and provided features is WinCC. As WinCC has an open architecture [19], it has been selected as the focus of architecture and feature comparison. By choosing an open-source SCADA solution with similar features and structure to WinCC, we make the claim that our outcomes are applicable to commercial SCADA systems.

SCADA architecture and feature comparison

The WinCC open architecture [20] is a theoretical model that consists of a number of software components called managers. There are three key managers that make up the core WinCC functionality: event, device, and database managers. Similar features are provided by the open-source SCADA solutions presented in section 3.

- Event managers handle notifications and alarms. The EPICS solution provides alarm handling through an

application, which notifies the user when information is stored in a distributed database. Users of TANGO build event notification through access to databases. EclipseSCADA and IndigoSCADA perform checks on data stored in the real-time database.
- The Device manager handles connection of field devices and incorporates drivers. Similar functions are provided by all open-source SCADA solutions; each solution provides a system node which listens to connected devices via drivers.
- Database managers handle both historical and real-time data using relational databases. This approach is most similar to IndigoSCADA, which also uses relational database storage for both historical and real-time data. EclipseSCADA, EPICS, and TANGO store historical data in a relational database, but real-time data is stored in memory in the form of blocks.

Device support is another key feature, which can be compared (see Table 1). WinCC supports drivers in four categories: Open Platform Communications (OPC), Field bus, telecontrol systems, and TCP/IP drivers. Most open-source solutions provide similar support for field devices; EPIC, TANGO and EclipseSCADA support all but telecontrol systems. IndigoSCADA provides support for all but field bus devices.

SCADA code manageability comparison

Related work (see section 3) alludes to the need to modify a SCADA system to take advantage of cloud. For this reason a study of code manageability was carried out. Manageability of code can be based on static code metrics measuring coupling (see Table 2). Solutions with fewer modules and fewer links between modules will be easier to analyze and port. CppDepend [21] and Eclipse Metrics [22] were used to identify the number of modules, links, and shared functions.

Of the studied solutions, results show that EPICS is the most complex, having on average 2.2 links per module. TANGO and EclipseSCADA have around 1 link per module; however, EclipseSCADA is more manageable, consisting of fewer modules. IndigoSCADA is the simplest solution having on average 0.7 links per module.

Measuring the variance of shared functions can also give an indication of code manageability. Solutions, in which code is weakly coupled across many modules, can be difficult to partition, while modules with few tightly coupled links are easier to partition. EPICS and EclipseSCADA are loosely coupled solutions with few strong links. EPICS having more modules and links, has code-reuse spread across more modules. TANGO and IndigoSCADA have modules, which are highly coupled, only a few modules which reuse code.

The last aspect of manageability is the type of SCADA package; TANGO and EclipseSCADA are toolkits and require development be carried out before migration. EPICS and IndigoSCADA are out-of-the-box systems and do not require further development, reducing the time needed to carry out migration.

SCADA adoption comparison

SCADA adoption and support differs depending on the solution.

- TANGO is supported by a consortium of European Synchrotrons in Germany, Spain, Italy, Poland, and France. Key projects that utilized TANGO include: the C3 Prototype of the European Mars Analog Station, the diagnostics of the Laser Mégajoule and the laser facility CILEX-APOLLON. TANGO has a large community of users, contributing a large number of device drivers, at time of writing over 558 different devices are supported, including sensors, motors, vacuums and lasers (beam lines).
- EPICS is a collaborative project between Argonne and Los Alamos national labs. As of 2015, EPICS is deployed in over one hundred research lab across the globe; key research institutes using EPICS include Fermilab, Australian Synchrotron, and the Lawrence Berkeley National Labs. Commercial companies also provide services for EPICS, in the form of consulting, by providing device drivers for their hardware or by selling instruments with an embedded IOC.
- EclipseSCADA is part of the Eclipse IoT Industry Working Group initiative. EclipseSCADA has been deployed in a number of productive installations around the world. Commercial support is available through a company called IBS SYSTEMS, which uses the EclipseSCADA toolkit to build SCADA solutions targeted to users' industrial requirements.

Table 1 Comparison of supported SCADA drivers

Drivers	WinCC	EPICS	TANGO	EclipseSCADA	IndigoSCADA
OPC	Yes	Yes	Yes	Yes	Yes
Field Bus	Yes	S7, Profibus	S7, Profibus	S7	No
Telecontrol	Yes	No	No	No	DNP3, iec104, iec61850
TCP/IP	Yes	Modbus, Ethernet	Modbus	Modbus	Modbus, RFC1006

Table 2 Manageability of open-source SCADA solutions

SCADA Solutions	# Modules	Avg. # of Links	Variance of Shared Functions	SCADA System Type
EPICS	9	2.2	134	Out-of-the-box
TANGO	9	1.1	5598	Toolkit
EclipseSCADA	6	1	113	Toolkit
IndigoSCADA	9	0.7	7163	Out-of-the-box

- IndigoSCADA is supported by a company called Enscada, little is known about deployment of this SCADA system in research and/or commercial environments.

Conclusion

A summary of open-source SCADA packages in terms of their features, manageability and adoption/support is as follows (see Table 3):

- Similarity of WinCC has been measured based on features and driver support. Of the selected SCADA systems, both IndigoSCADA and EclipseSCADA share features in terms of event, device and storage managers, while TANGO and EPICS implement event handling and data storage differently than WinCC. Driver support across all open-source solutions is similar, supporting 3 out of 4 categories. As a toolkit with features similar to WinCC, EclipseSCADA would allow for a SCADA configuration similar to WinCC to be developed.
- Manageability is measured in terms of coupling and required development. IndigoSCADA, TANGO and EclipseSCADA had significantly less coupling then EPICS. Out of the solutions with low coupling, only Indigo is an out-of-the-box solution which does not require further development. Out of the toolkits, EclipseSCADA has fewer modules compared to TANGO and is therefore more manageable.
- EPICS and TANGO are widely used in research areas, and have a large support base. EclipseSCADA is not as well-known as EPICS and TANGO, but used commercially, and has links to the eclipse foundation, which are points in its favor. IndigoSCADA has limited adoption in research or industry, and is ranked last.

Based on these criteria, the open source EclipseSCADA was chosen. As a toolkit, EclipseSCADA is flexible enough to build a SCADA solution that is similar to WinCC. The low coupling score suggests a modular solution, which could be modified to suit cloud infrastructure.

Migration of EclipseSCADA

The open source EclipseSCADA toolkit was chosen as a representative SCADA system. Using EclipseSCADA, a SCADA system was developed, which consisted of a remote sever, a master server, and a file server. Using the "lift and shift" method, EclipseSCADA was ported to the NeCTAR research cloud.

EclipseSCADA solution development

Development of a SCADA solution using the EclipseSCADA toolkit is carried out using the EclipseIDE. Through the toolkit, users are provided with a configuration template that they can use to define the structure of the SCADA system, by creating and configuring servers and devices (see Fig. 5).

The configuration template defines two type of servers, external or system nodes: the remote server is where real time storage of data is performed, and the system node is where device drivers are run. Users modify these templates by specifying the running IP address of each server. Users can link any number of devices to a system node. Device templates are available for a number of communication protocols, which can be modified by specifying port numbers and data types, and communication period.

When compiled, the SCADA system defined by the user takes the form of OSGi services [23], which are bundled Java code that can be installed, started, stopped, and updated through a simple API. A separate service is built for each IP address defined by the user, and is designed to be deployed on the specified machine. Users

Table 3 Examination of open-source SCADA solutions

SCADA Solutions	Criteria				
	Similarity to WinCC		Manageability		Widely Used
	Features	Driver Support	Coupling	Toolkit	
EPICS	Through Extensions	3/4 categories	Low	No	Yes
TANGO	Through Extensions	3/4 categories	High	Yes	Yes
EclipseSCADA	Through Customization	3/4 categories	Low	Yes	Yes
IndigoSCADA	Yes	3/4 categories	High	No	No

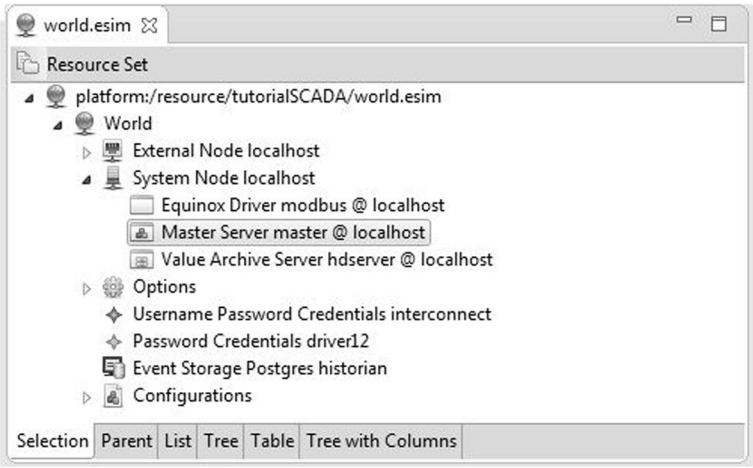

Fig. 5 EclipseSCADA configuration template

can access these compiled OSGi services through a GUI, which implements Human Machine Interface (HMI) features. This GUI can link to system node and file server, giving end-users access to real time and historical data, alarm and event handling.

Using the EclipseSCADA toolkit, the core components shown in Fig. 6 were developed. The core components consist of a remote server, a master server, and file server.

- The remote server provides real-time collection of sensor, switch, and actuator information. Data is stored in memory (data blocks) and converted to the EclipseSCADA internal protocol, Next Generation Protocol (NGP [24]).
- The master server provides event and alarm handling for NGP formatted data.
- The file server provides data archival. A relational database service is used to store and manage data.

Migration methodology

When migrating EclipseSCADA to the cloud, three different deployment paths can be considered: re-hosting, refactoring, and revising [25]. The quickest and simplest approach is achieved by simply re-hosting an application in the cloud ("lift & shift" method). Re-hosting is the process of installing an existing application in a cloud environment and mainly relying on Infrastructure as a Service (IaaS) offerings. This can be the first step in a gradual approach, enabling to perform initial analysis and to improve the application during multiple iterations.

To better benefit from the characteristics of cloud computing, e.g., making an application scalable or more reliable, re-engineering might be necessary. This can be a simple refactoring, i.e. a modification of one or a few features. An example is adding monitoring capabilities, which enable elastic behavior such as adding new resources when the application is heavily used or releasing resources when they are not needed. It might also require revising an application, i.e. making major modifications at its core; examples would be using a Platform as a Service (PaaS) database offering or changing an application into a multi-tenancy SaaS offering. To fully benefit from the cloud, it might also be necessary to rebuild an application and integrate, for example automation for scaling in and out. Driven by the increasing number of SaaS offerings, an option might be to replace an existing application with a cloud-based SaaS solution [26].

We used a combination of re-hosting and performance testing to deploy EclipseSCADA to the NeCTAR research cloud [2] (running OpenStack [27]). The steps used to carry out "lift and shift" are as follows:

1. Create and upload an Ubuntu VM image to the NeCTAR cloud.
2. Deploy the Ubuntu VM and carry out installation of EclipseSCADA.
   a. Install the EclipseSCADA dependencies (Java, Eclipse, etc.).
   b. Copy each EclipseSCADA component into the VM.
   c. Configure EclipseSCADA links by specifying the location of each sensor/switch and SCADA component.
3. Start each EclipseSCADA component.

Following these instructions each EclipseSCADA component (field device, remote server, master server, and file server) was deployed to the NeCTAR cloud as a standalone virtual machine.

Sensor and actuator simulation

EclipseSCADA was configured to use simulated Modbus/TCP devices. Modbus is one of the most commonly

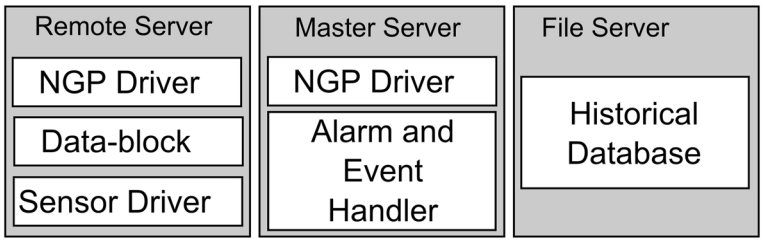

Fig. 6 EclispeSCADA Core Components

used polling protocols, with numerous simulators and libraries. While the original version of the Modbus protocol is designed for use over serial ports, the Modbus/TCP variant enables communication using the Internet Protocol Suite.

A Modbus device consists of registers that are used to hold data. Request messages are sent from the SCADA system to a Modbus device to carry out operations on registers. The Modbus device also sends response messages back to the SCADA system. A message consists of a number of fields;

- Transaction Identifier - Incrementing ID for synchronization of messages.
- Protocol Identifier - Set to 0, reserved for future extensions.
- Length Field - Remaining bytes in message.
- Unit Identifier - The register ID to carry out the operation.
- Function Code - The function to be carried out (read, write, etc.).
- Data Bytes - Data for response or commands (data which was read or to be written).

The operation of the Modbus/TCP Protocol and five sensors is illustrated in Fig. 7. The EclipseSCADA system requests data by sending a message to the connected field device. The field device retrieves the requested sensor data and sends sensor data back to the EclipseSCADA system. Sensors are polled sequentially until data from each connected sensor is retrieved by the SCADA system.

When measuring the time taken to transfer Modbus data, there are two terms that must be defined: Round Trip Time (RTT) and the Polling Interval. Round Trip Time (RTT) is the time between sending a request until receiving the respective response with the sensor data. The Polling Interval is the time between sending requests to the same sensor. The Polling Interval is configured by the user and is often implemented in the form of a timer.

Analysis of EclipseSCADA metrics

To study the real-time monitoring and control mechanism of SCADA systems when running on the cloud, an analysis of the migrated EclipseSCADA system was carried out. Metrics were collected which focus on understanding the influence of the size of a monitored environment and EclipseSCADA specific delays, the influence of the location

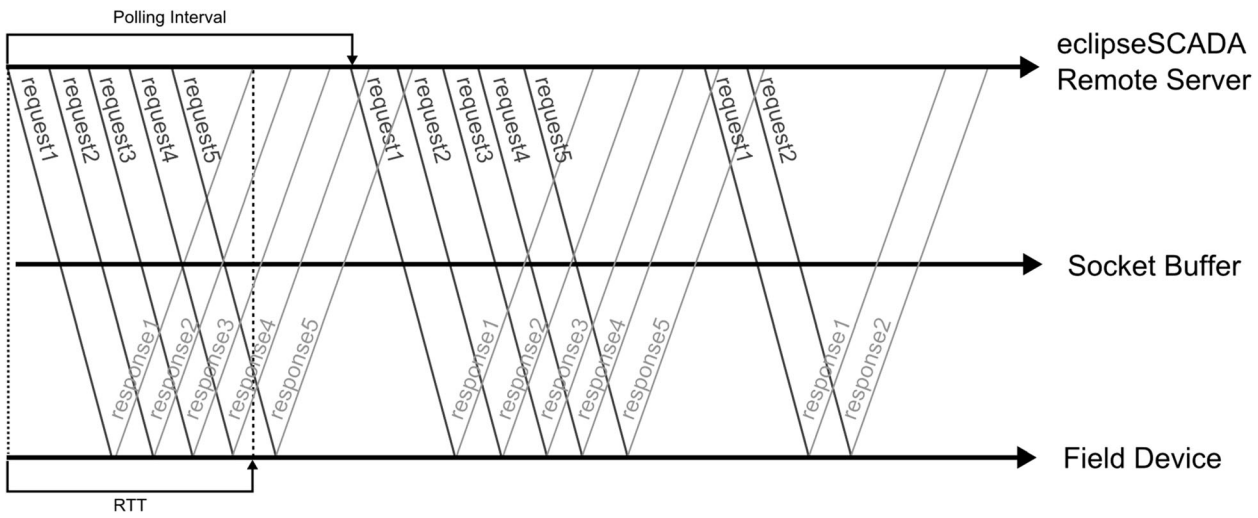

Fig. 7 Modbus Polling RTT

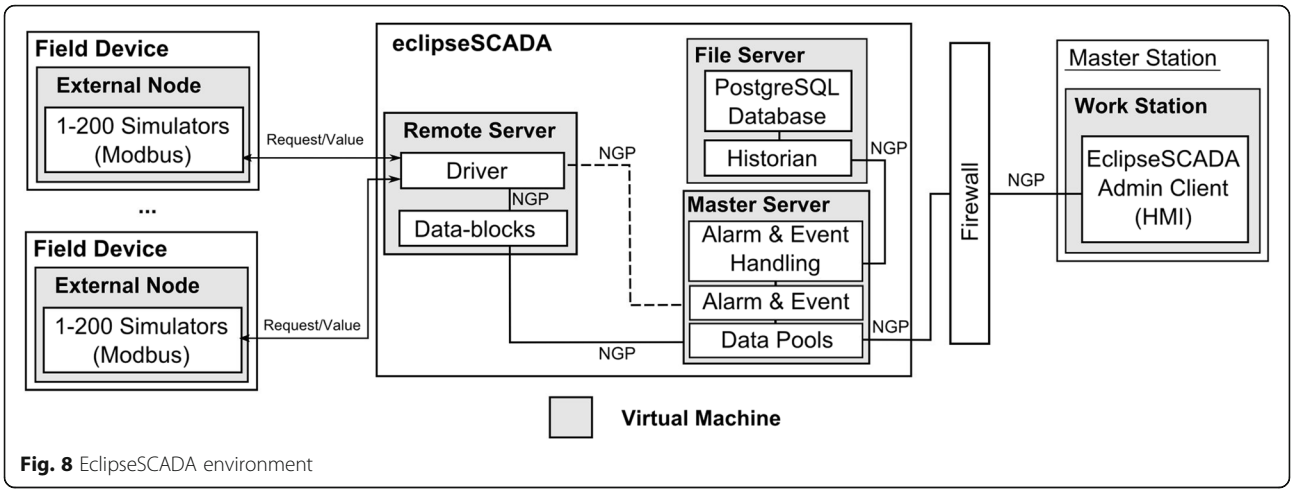

Fig. 8 EclipseSCADA environment

of EclipseSCADA components on EclipseSCADA performance, and the relationship between processing time and communication time.

Cloud and SCADA environment setup

Each EclipseSCADA component was deployed on an individual virtual machine hosted on the NeCTAR research cloud [2], which runs OpenStack (Kilo Release), configured with Neutron with Linux bridge networking. Each virtual machine (m1.medium) had 2 virtual CPUs, 8GB RAM and 60GB secondary disk. Each virtual CPU consisting of a single core running at 2.6 GHz clock speed.

Performance metrics were collected while EclipseSCADA was placed under different sensor loads (200, 400, 600 and 800). Modbus/TCP simulators were deployed on each field device to represent sensors. As each field device could only contain 200 simulated sensors (due to implementation restrictions of the used Modbus simulator), additional field devices (running on separate virtual machines) were added to the EclipseSCADA system for the 400, 600 and 800 sensor load experiments (see Fig. 8).

The EclipseSCADA system was deployed across Melbourne and Tasmania cloud regions, where simulated field devices were deployed in Tasmania, and SCADA components were deployed in Melbourne. There was an estimated distance of 429 km between these two sites. Table 4 shows the speed of the internal and external network, where the internal Melbourne network is over twice as fast as the public (Melbourne to Tasmania) network.

Table 4 Comparison of network speed (TCP)

Location	Download (Gbits/sec)	Upload (Gbits/sec)
Internal Melbourne Network	3.77	3.30
Melbourne to Tasmania	1.52	1.07

The study of the influence of the size of a monitored environment

In order to understand the influence the size of a monitored environment has on the performance of EclipseSCADA, the Modbus/TCP Round Trip Time (RTT) was measured using Wireshark [28] (see Table 5). As mentioned above, simulated field devices were deployed in Tasmania, while SCADA components were deployed in Melbourne.

The numbers of simulators were varied between 1 and 800, with a polling rate of 1 millisecond. Typically SCADA systems are setup with a slower polling interval of seconds, but with thousands of sensors. A polling rate of 1 millisecond was deliberately chosen to put more load on the system, in order to adjust for the number of sensors.

Results showed that the RTT changes depending on the number of simulated sensors attached to the eclipseSCADA system. In a rather small SCADA system consisting of a single simulator sensor, the time taken to transfer a single Modbus/TCP sensor of data is on average 11 milliseconds, where 4 milliseconds are spent sending a request and 7 milliseconds are spent sending sensor data. As the size of a monitored environment increases, i.e., more simulator sensors are added to the SCADA system, the average time to retrieve a single point of sensor data increases.

In conclusion, the majority of the RTT time is spent retrieving data from the simulator. In order to understand the cause of this behavior, experiments were

Table 5 Average Time to Transfer Modbus Data (ms)

# Simulated Sensors	Request	Response	Total RTT
1	4	7	11
200	4	3630	3634
400	4	5996	6000
600	4	9034	9038
800	4	13,961	13,965

carried out to investigate how the network and SCADA components interact.

The study of the influence of component location

In order to understand how the distribution of SCADA components affects network communication, "field devices" and "remote server" components were deployed across the Melbourne and Tasmania cloud regions. Modbus/TCP Round Trip Time (RTT) was measured: between Melbourne and Tasmania (Remote Region), within the Melbourne region (Single Region), and on a single machine (Single Machine) (see Fig. 9).

Results indicate that changing how EclipseSCADA is deployed geographically has minimal overall effect on Modbus/TCP data transfer. The single machine setup eliminates the network completely, resulting in an average RTT reduction of ~0.5 seconds (when compared with the remote region setup). Likewise, EclipseSCADA running in a single region performs only slightly better with an average RTT reduction of ~0.3 seconds.

In conclusion, network transfer and delays are responsible for a small percentage of the total Modbus/TCP RTT. To further reduce RTT, it is necessary to reduce the number of sensors managed by a single remote server, or use a more efficient communication protocol. By using event-driven protocols, it is possible to remove the need for request messages.

Processing time vs. communication

Results presented in Fig. 9, show that eliminating the network (single machine) has a minimal effect on Modbus/TCP RTT. From this, it is possible to conclude that most of the observed delay when requesting data (see Table 5) is due to processing rather than network delays. Furthermore, as the time to retrieve sensor data increases with the number of monitored sensors, it is likely that the delay is due to the implementation of the Modbus/TCP protocol sending all requests before receiving sensor data.

In conclusion, network communication can be neglected in comparison to the time spent processing. Results show that the migrated SCADA system spent more time generating and sending polling requests than storing data. For this reason, Modbus/TCP requests and responses should be managed by an event queue or on separate threads.

Distinctive features of experimentation

Sections 6.1–6.4 show a unique nature and large scale of experimentation. First, the experiments were carried out using two production clouds, the NeCTAR research clouds; each ran OpenStack. Second, the experiments were carried out across Melbourne and Tasmania cloud regions; the distance between them, 429 km, is impressive in terms of industry, business, and research applications. Third, the speed of the internal and external network, where the internal Melbourne network is over twice as fast as the public (Melbourne to Tasmania) network; such a difference creates some buffering problems. Fourth, since we wanted to carry out experiments in boundary computation and communication conditions, with a huge number of devices, it was necessary to use simulated field devices. That allowed us to run experiments with load of up to 800 sensors deployed in Tasmania (SCADA components were deployed in Melbourne). Fifth, the numbers of simulated devices were varied between 1 and 800, with a polling rate of 1 millisecond. Typically, SCADA systems are setup with a slower polling interval of seconds. A polling rate of 1 millisecond was deliberately chosen to put more load on the system. Sixth, all experiments were carried out during normal exploitation of clouds and connected them public networks; that formed normal business/industry conditions.

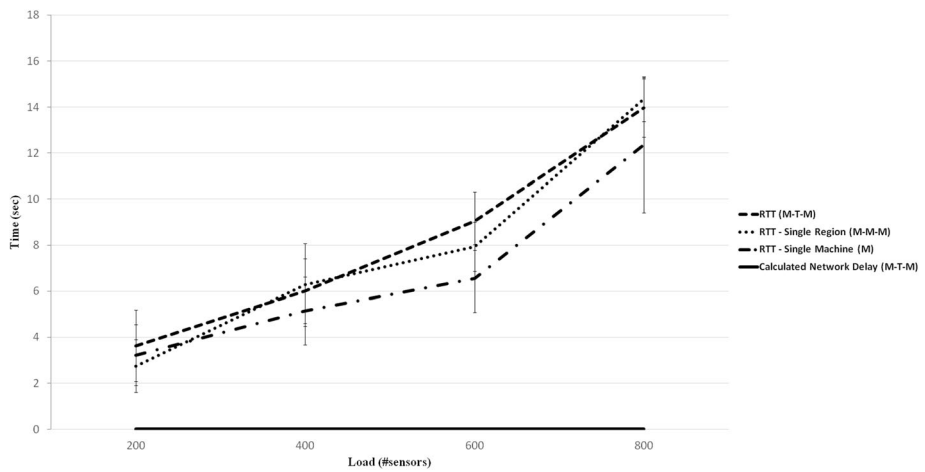

Fig. 9 Average RTT of Modbus Data

Recommendations

Based on these observations, we make the following recommendations when moving and configuring SCADA to make use of cloud resources:

- Use event-driven protocols (where a sensor informs the SCADA system on change) – Polling in general is not efficient; to retrieve a sensor value, a request message and response message must be sent; additionally as a timer is used there is a tendency for a polling based system to flood a network with repetitive sensor data. Event-driven protocols reduce the amount of data sent across a network.
- If a legacy SCADA system is used it needs to be ensured that processing of the messages does not incur excessive delays. Our results, for example, indicate that for larger system loads, network communication can be neglected in comparison to the time spent processing and storing data. When using device protocols that are built for serial devices (like Modbus), it is sometimes a design limitation of the device or protocol, that there cannot be requests in parallel. For the other cases, proper techniques, such as use of an event queue or multi-threading, need to be employed to minimize processing delays.
- When the above cannot be influenced and polling protocols with serial processing need to be used (like Modbus TCP in certain configurations), field devices should be spread across many remote servers – The amount of sensors per remote server depends on the processing and transfer time required by the SCADA system. This also means that the remote server should be able to scale horizontally. It can also be concluded that the network transmission time influences the performance, and that protocol conversion - at least for polling protocols - should be done close to the field devices.

Conclusion and future work

Monitoring and control of industrial complexes is of critical importance. Each individual complex is in the majority of cases distributed, e.g., a gas pipeline, industrial plant. Therefore, there is a need for a system that can provide services in real time and is distributed. SCADA systems are instrumental to a wide range of mission-critical industrial systems. These systems allow a user to monitor (using sensors) and control (using actuators) an industrial system remotely.

Currently, SCADA systems run on highly reliable and dedicated hardware. This is in contrast to the current state of computing, which is moving from running applications on internally hosted servers to flexible, cheaper, internal or external cloud systems. The problem is to identify factors that affect performance of cloud-based SCADA systems.

Experiments were carried out to examine the effects cloud resources and public networks have on SCADA systems. Using the "lift and shift" approach, EclipseSCADA was deployed to the NeCTAR research cloud in multiple locations. Performance metrics were collected from the deployed EclipseSCADA system under different loads.

When moving a SCADA system to cloud infrastructure, there is a need to ensure that the real-time monitoring and control demands of the industrial system can be achieved. By carrying out analysis of the EclipseSCADA system we made a series of recommendations that should be taken into consideration when migrating SCADA systems to the cloud. In general, latencies introduced by running SCADA system components in the cloud are not a limiting factor, given that response time requirements are usually in the order of several hundred milliseconds to seconds. While scalable compute power in a cloud-based system tempts to centralizing functionalities, this may lead to problems in given system designs. We have specifically discovered limitations with polling protocols processed in a serialized way. We propose in particular the adoption of event-driven communication protocols to reduce network transfer, doing protocol conversion close to the field devices, and replication of remote servers, to ensure polling intervals are met.

Adaption of event-driven protocols for cloud based SCADA systems would require the use of smart sensors that would be able to respond to requests made by the cloud. The complexity of these field devices means that simulators are not readily available. Polling protocols are based on timers transfer the same amount of data every polling interval. When working with event driven simulators, the amount of data being transferred would change depending on the registered events (which is depended on the size of the system and application). For this reason experiments with event driven protocols must include the simulation of realistic datasets, and ideally under different applications and scenarios. Future work would extend the experimental methodology to event driven protocols, which would require the collection of datasets from large scale SCADA installations and the development of simulators which would playback these datasets.

Endnote

[1]As of the 19th of November 2015, EclipseSCADA has been renamed to Eclipse NeoSCADA (https://www.eclipse.org/eclipsescada/news/2015/11/19/a_new_name__eclipse_neoscada.html).

Acknowledgments

The work presented in this paper was funded by Siemens AG, Corporate Technology, Munich.

A NeCTAR research grant, provided access to the IaaS cloud resources necessary to deploy EclipseSCADA in the cloud.
The authors are also grateful for the support of Deakin University, for providing a working environment, and wish to express their gratitude to the anonymous reviewers and Editor in Chief for their constructive comments.

Authors' contributions
PC carried out the experimental work and wrote the manuscript. AG, CR, ZR participated in study design and coordination, as well as the validation of experimental results. HM and SVG contributed to the selection of the open source SCADA system. All authors have read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Author details
[1]School of Computer Science and Information Technology, GPO, RMIT University, Box 2476, Melbourne, Australia. [2]Corporate Technology, CT RDA ITP, Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany. [3]School of Information Technology, Deakin University, Geelong, Australia.

Received: 28 March 2017 Accepted: 3 May 2017
Published online: 09 June 2017

References

1. A. Alharthi, Z. Tari, A. Goscinski, and I. Khalil, "PPFSCADA - Privacy Preserving Framework for SCADA Data Publishing," *Int. Journal on Future Generation Computer Systems (FGCS),* 2014.
2. J. Kirby. (2012). *NeCTAR - Australian Research Cloud.* Available: http://www.nectar.org.au/. Accessed 6 June 2017.
3. Ken Barnes BJ, Nickelson R (2004) Review Of Supervisory Control And Data Acquisition (SCADA) Systems. Idaho National Engineering and Environmental Laboratory, Idaho Falls, Idaho
4. Stuart A, Boye E (2010) SCADA: supervisory control and data acquisition, Research Triangle Park. International Society of Automation, NC
5. IEEE Power Engineering Society, "IEEE Standard for SCADA and Automation Systems," 2008.
6. Modicon, "Modicon Modbus Protocol Reference Guide," ed, 1996, p. 121.
7. Real Time Automation. (2016, 21/4). *PROFIBUS Protocol Overview - Real Time Automation.* Available: http://www.rtaautomation.com/technologies/profibus/. Accessed 6 June 2017.
8. Siemens. (2016, 21/4). *Siemens communications overview*. Available: http://snap7.sourceforge.net/siemens_comm.html. Accessed 6 June 2017.
9. IPCOMM GmbH. (2016, 21/4). *IEC 60870–5-104*. Available: http://www.ipcomm.de/protocol/IEC104/en/sheet.html. Accessed 6 June 2017.
10. Liu M, Guo C, Yuan M (2014) The Framework of SCADA System Based on Cloud Computing. In: Leung V, Chen M. (eds) Cloud Computing. CloudComp 2013. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 133. Springer, Cham
11. Gligor A, Turc T (2012) Development of a Service Oriented SCADA System. *Procedia Economics and Finance* 3:256–261
12. S. Goose, J. Kirsch, and D. Wei, "SKYDA: cloud-based, secure SCADA-as-a-service," *International Transactions on Electrical Energy Systems,* pp. n/a-n/a, 2014.
13. Inductive Automation. (2014). *Ignition by Inductive Automation*. Available: https://www.inductiveautomation.com/scada-software/. Accessed 6 June 2017.
14. XiO. (2015). *XiO Cloud SCADA® Control System.* Available: http://www.xioio.com/wp/. Accessed 6 June 2017.
15. A. Johnson. (2014). *EPICS - Experimental Physics and Industrial Control System*. Available: http://www.aps.anl.gov/epics/index.php. Accessed 6 June 2017.
16. TANGO Consortium, "The TANGO Controls website," 2014.
17. IBH Systems GmbH. (2014). *openSCADA | We are the good guys*. Available: http://openscada.org. Accessed 6 June 2017.
18. apaatsf. (2015). *IndigoSCADA*. Available: http://sourceforge.net/projects/indigoscada/?source=navbar. Accessed 6 June 2017.
19. Siemens. (2014). *SCADA System SIMATIC WinCC*. Available: http://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/pages/default.aspx. Accessed 6 June 2017.
20. ETM professional control GmbH (2015) SIMATIC WinCC Open Architecture [Brochure]. Austria, 2015, p 8
21. CoderGears. (2015). *CppDepend: : C/C++ Static Analysis and Code Quality tool*. Available: http://www.cppdepend.com/. Accessed 6 June 2017.
22. sauerf. (2015). *Eclipse Metrics plugin*. Available: http://metrics.sourceforge.net/. Accessed 6 June 2017.
23. OSGi™ Alliance. (2015). *The OSGi Architecture*. Available: https://www.osgi.org/developer/architecture/. Accessed 6 June 2017.
24. J. Reimann. (2013, 22 Jan). *openSCADA Protocol Description*. Available: http://download.openscada.org/documentation/l/1.2.0/html/protocol/. Accessed 6 June 2017.
25. Gartner. (2011). *Gartner Identifies Five Ways to Migrate Applications to the Cloud*. Available: http://www.gartner.com/newsroom/id/1684114. Accessed 6 June 2017.
26. Philip Church, Harald Mueller, Caspar Ryan, Spyridon V. Gogouvitis, Andrzej Goscinski, Houssam Haitof, et al., (2017) "SCADA systems in the Cloud," in Handbook of Big Data Technologies, ed: A.Y. Zomaya and S. Sakr, Springer, Cham, pp. 691–718
27. OpenStack Project. (2012). *OpenStack - Open source software for building private and public clouds*. Available: http://www.openstack.org/. Accessed 6 June 2017.
28. Wireshark Foundation. (2015). *Wireshark - Go Deep*. Available: https://www.wireshark.org/