

# Research on the Architecture and its Implementation for Instrumentation and Measurement Cloud

Hengjing He, Wei Zhao, Songling Huang, Geoffrey C. Fox, and Qing Wang

**Abstract**—Cloud computing has brought a new method of resource utilization and management. Nowadays some researchers are working on cloud-based instrumentation and measurement systems designated as Instrumentation and Measurement Clouds (IMCs). However, until now, no standard definition or detailed architecture with an implemented system for IMC has been presented. This paper adopts the philosophy of cloud computing and brings forward a relatively standard definition and a novel architecture for IMC. The architecture inherits many key features of cloud computing, such as service provision on demand, scalability and so on, for remote Instrumentation and Measurement (IM) resource utilization and management. In the architecture, instruments and sensors are virtualized into abstracted resources, and commonly used IM functions are wrapped into services. Users can use these resources and services on demand remotely. Platforms implemented under such architecture can reduce the investment for building IM systems greatly, enable remote sharing of IM resources, increase utilization efficiency of various resources, and facilitate processing and analyzing of Big Data from instruments and sensors. Practical systems with a typical application are implemented upon the architecture. Results demonstrate that the novel IMC architecture can provide a new effective and efficient framework for establishing IM systems.

**Index Terms**—Architecture, cloud computing, distributed computing, instrumentation and measurement cloud, parallel processing, power system state estimation, cloud service

## 1 INTRODUCTION

SINCE instrumentation and measurement (IM) technology is closely combined with information technology, development in information technology (IT) can lead to the advance of IM technology. In the early stages, computers were used to control instruments and sensors for local data acquisition and analysis. Later on, virtual instrumentation technology was developed and many of the functions that were implemented by hardware in instruments can now be achieved by computer software. With the development of networks and the internet, remote instrumentation and measurement (RIM) emerged as a new technology in IM field [1]. Such RIM technology has brought many benefits to related areas, especially to those areas that involve large distributed systems [2]. It can greatly facilitate instrument control and data acquisition and processing. Additionally, new computing paradigms, such as grid computing, can be integrated into IM technology to further improve the ability for data processing, and resource sharing and management of distributed IM systems [3]. Grid-enabled instrumentation

and measurement system (GEIS) is a typical type of those systems. GEIS brings many advantages to data intensive IM applications and heterogeneous IM resource management. However, due to some limitations of grid computing, IM systems that integrate a grid are eventually not widely adopted in practical use.

Currently, most IM systems are built upon local architecture or traditional client/server (C/S) architecture as explained in [4]. In a local IM architecture, such as the main architecture of the NI LabVIEW platform, instruments and sensors are connected directly to the computer and it is difficult to build large IM systems. As for C/S architecture, instruments and sensors simply provide remote access interfaces through gateway servers to clients. However, both architectures require the user to build the entire IT system and to maintain all resources. Thus, there has to be a great investment in building the whole IM system and, besides, system stability, scalability and fault tolerance can be serious problems for both architectures. Moreover, to satisfy resource requirement for peak and valley load, the IM system should be built according to the peak load at the very beginning and, even if the load drops, the system cannot scale down accordingly. Therefore, the utilization rate of resource in the IM system can be quite low, especially for those systems with great load dynamics. In addition to the problems mentioned above, large amounts of data collected from various devices need much more powerful computing resources for processing and analyzing, and traditional computing paradigms may be incapable of dealing with such scenarios.

- Hengjing He is with the State Key Lab. of Power System, Department of Electrical Engineering, Tsinghua University, Beijing, 100084, China. E-mail: hehj11@mails.tsinghua.edu.cn.
- Wei Zhao is with the State Key Lab. of Power System, Department of Electrical Engineering, Tsinghua University, Beijing, 100084, China. E-mail: zhaowei@mail.tsinghua.edu.cn.
- Songling Huang is with the State Key Lab. of Power System, Department of Electrical Engineering, Tsinghua University, Beijing, 100084, China. E-mail: huangsling@tsinghua.edu.cn.
- Geoffrey C. Fox is with the School of Informatics and Computing and CGL, Indiana University, Bloomington, USA. E-mail: gcf@indiana.edu.
- Qing Wang is with the School of Engineering and Computing Sciences, Durham University, Durham, UK. E-mail: qing.wang@durham.ac.uk

In recent years, the emergence of cloud computing has brought many new approaches for resource utilization and management [5]. Cloud manages all resources as a resource pool and provides those resources as online services to end users according to their demand. Such modes can greatly increase the utilization efficiency of resources and, at the same time, save the investment of users on both hardware and software resources [6]. Moreover, big data processing and analysis technologies developed along with cloud computing make data analysis much easier and faster in the IM field [7]. Motivated by these benefits, many researchers are exploring novel cloud based IM technologies to solve the problems above [8]. Until now, most of the work carried out in the interdisciplinary area of IM and cloud computing mainly focuses on the application of cloud computing in IM systems, which can only deal with a few aspects of related problems. Little research has been carried out to build novel IM modes and architectures that can inherit the essence of cloud computing for IM systems.

This paper introduces a novel IMC architecture with detailed system implementations. The architecture abstracts instruments and sensors into resources, and encapsulates frequently used modules and functions into services. Services are deployed in the cloud and users can consume these services on demand. IM applications are also deployed and run in the IMC platform. All IT resources are allocated and managed by the IAAS (Infrastructure as A Service) cloud platform, which will reduce investments for users and also increase resource utilization efficiency. By integrating cloud computing and big data processing technologies, IMC can benefit a lot from advantages such as system scalability, fault tolerance, distributed and parallel computing, and so on. An actual system based on this architecture is implemented using various cloud computing and big data processing frameworks. Applications and experiments are designed to test the system. Results show that the IMC architecture designed in this paper can properly integrate cloud computing with IM technologies and greatly facilitate building, managing and using IM systems.

The remainder of this paper is organized as follows: section 2 presents related work; section 3 introduces key concepts of IMC; section 4 describes the detailed IMC architecture designed by this paper; section 5 illustrates the implementation of the architecture; section 6 provides some applications and tests over the IMC system; section 7 discusses challenges and limitations of IMC; and finally, section 8 concludes the whole paper.

## 2 RELATED WORK

Most of the work related to IMC mainly focuses on the following areas: Grid-enabled instrumentation systems (GEIS) [9], sensor clouds [10] and instrumentation clouds [11].

GEIS mainly focuses on converting instruments into grid services [12], so that heterogeneous instrumentation resources can be accessed and managed through a grid, which can facilitate data intensive applications to use var-

ious grid resources and provide distributed instrumentation and experiment collaborations over the grid [13]. In GEIS, instruments are abstracted into unified services through middleware technologies and standard models. Typical GEISs include Instrument Element [14] architecture from the GridCC project, common instrument middleware architecture [15], e-infrastructure for remote instrumentation from the DORII project [3] and virtual laboratory architecture [16].

Although GEIS provides a good way for distributed instrumentation and collaborative experiments over the grid, limitations of grid computing, such as complexity, constrained accessibility and so on, prevented it from prevailing among scientific and industrial areas. However, some of the research work in GEIS, regarding common aspects, can guide the study of IMC. Data retrieval and transmission approach in distributed IM systems is an important one of those aspects. Through comprehensive research, [12] and [17] demonstrated that publish/subscribe mechanisms are more efficient for real-time data collecting and transmitting in a distributed IM environment. Thus, in the work of this paper, a message-based publish/subscribe mechanism is adopted as the information and data dissemination method.

Just like IMC, sensor cloud is also a very new concept. In [18] a sensor cloud infrastructure is developed and physical sensors are abstracted into virtual sensor objects. Such virtual sensor objects combined with related sensor definition templates can provide measurement services to end users. Besides, service and accounting models are designed, which makes the sensor cloud infrastructure conform to the philosophy of cloud computing. Another project working on cloud-based sensing systems is S4T (Stack4Things) [19]. This project is the base for the #SmartME [20] project, which mainly focuses on morphing a city into a smart city. The S4T project is trying to build up a cloud platform for managing a large number of sensors and actuators deployed in a distributed environment. One of the main ideas of the S4T project is virtualizing sensors and actuators into abstract resources similar to those resources in cloud computing and providing sensing and actuating services through the S4T platform. Thus, the S4T platform is also viable for building a sensor cloud but it is more focused on building a cloud on the edge of a sensor network. Some other researchers also use the terminology 'sensor cloud', but most of them only concentrate on the application of cloud computing technology in sensor control and management [21],[22],[23],[24]. Sensors are much simpler than instruments, however they can also be treated as instruments, only with less functions.

Current studies of instrumentation clouds only bring forward conceptual models and architectures with few details or implementations provided [11],[25],[26]. Some other research work is mainly about applications of cloud computing in massive data storage and processing [27],[28],[29], which, as explained before, only provide solutions for a few of the problems faced by current IM systems.

Compared with the above research, the work of this

paper has the following advantages: (1) the IMC platform developed in this paper is more open and easier to access than GEIS; (2) the IMC platform can manage both sensors and instruments, and provide commonly used IM functions as on-demand, scalable services to end users; (3) the IMC system implemented according to the IMC architecture in this paper is a real cloud platform for the IM field, rather than just an application of cloud computing technologies in the IM field. All these advantages are achieved by adopting a cloud-based resource management and usage mode, and using state-of-the-art technologies from cloud computing and big data fields. Details will be presented in the following sections.

### 3 INSTRUMENTATION AND MEASUREMENT CLOUD

Currently, no standard definition for IMC is presented. This section brings up a relatively standard definition for IMC and related elements by adopting key ideas of cloud computing in the IM field.

#### 3.1 Definition of IMC

The key ideas of cloud computing are: resource abstraction and virtualization, services delivered on demand, and scalability [30]. Owing to these ideas, cloud computing can provide scalable, on-demand hardware and software resources remotely. As introduced in Section 1, traditional IM systems and modes do not own such advantages. To inherit these merits, it is necessary to integrate cloud computing and related big data technologies into IM field, and establish a novel Instrumentation and Measurement Cloud (IMC). Based on our previous work [31], a definition for IMC is brought forward below:

**Definition 1.** *Instrumentation and Measurement Cloud(IMC) is a model, based upon cloud computing and big data frameworks, for enabling on-demand, convenient, ubiquitous network access to a shared pool of Instrumentation and Measurement(IM) resources(e.g. instruments, sensors, actuators) and services that can be rapidly provided and released with minimal interaction with resource provider and end user.*

From definition 1, it can be seen that IMC contains two direct entities, resource and service, and a third hidden entity which is the IM application designed by users. To distinguish these entities from similar concepts in the traditional IM field, IMC resource, IMC service and IMC application are used to feature these three entities of IMC. In this paper, IMC resource, IMC service and IMC application are also called IMC elements in general.

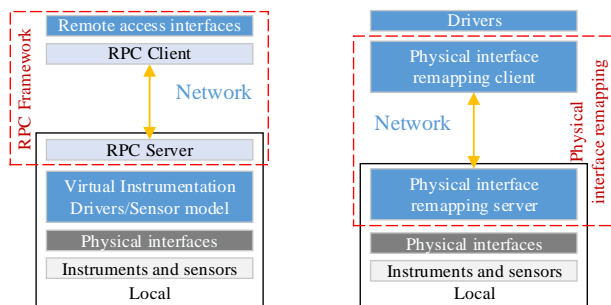
#### 3.2 Definition of IMC resource

**Definition 2.** *Instrumentation and Measurement Cloud resource stands for virtualized instrument, sensor or actuator resources that are connected to IMC through the network for online sharing and management.*

As can be seen from the above definition, computing resources are not included in IMC resources. The reason why we define IMC resource this way is that computing

resources, such as the CPU, storage and the network, are managed and provided by cloud computing frameworks of IMC, and, to IMC, they play the role of service.

Unlike computing, storage and networking resources, instruments and sensors have heterogeneous software and hardware architectures, which means it is very difficult to virtualize them into a unified resource pool and at the same time keep their full features. However, thanks to virtual instrumentation (VI) technology and standard sensor models, many of the modern instruments and sensors can provide unified access interfaces. However, such interfaces are designed just for local drivers. To virtualize those IM devices into IMC resources, interface remapping through the network is required. Generally, there are two ways to remap the interfaces, as shown in Fig. 1.



(a) Software interface remapping (b) Physical interface remapping  
Fig. 1. Remote instrument or sensor interface remapping

Fig. 1(a) shows a software interface remapping scheme using the remote procedure call (RPC) approach. Such a scheme is more data-oriented, since it mainly focuses on manipulating data exchanged between physical devices and up-level applications. This method is easier but less flexible. For different VI frameworks or sensor models, corresponding RPC modules should be developed.

The second method, illustrated in Fig. 1(b), remotely maps physical interfaces, such as USB [32], RS-232, GPIB and many others, to the cloud side, thus IM device connections to those interfaces will be forwarded to the cloud. Physical interface remapping is a device-oriented design which is more concerned about the device itself rather than the data generated from the device and it supports full features of the device. However, implementation of this method is much more difficult, especially for high-speed interfaces such as PCIE, than that of the software interface remapping approach. Furthermore, in this method, each IMC resource should be attached to a VM in the cloud. An IM system based on physical interface remapping is more like a traditional IM system. Unless there are special requirements, it is better to use software interface remapping for IM device virtualization.

#### 3.3 Definition of IMC service

**Definition 3.** *Instrumentation and Measurement Cloud service stands for online, scalable, shared IM functions, cloud computing resource services and big data processing services that can be consumed through IMC by IMC applications on demand.*

The definition of IMC service is close to the concept of information.

SAAS (Software as a Service) in cloud computing [33]. In IMC, commonly used IM functions are encapsulated into online services. Users can consume those services on demand and all those services are running in IMC. When consuming IMC services, users just need to send data to input interfaces of IMC services and retrieve results from output interfaces.

### 3.4 Definition of IMC application

**Definition 3.** *Instrumentation and Measurement Cloud application is the application program that consumes IMC resources and IMC services, and carries out custom logics to fulfill user defined IM tasks.*

Although the IMC platform can provide remote access to IMC services and IMC resources, it still requires the user to organize those services and manipulate related resources to carry out specific IM tasks. Since most of the computation-intensive data processing procedures can be implemented into IMC services deployed in IMC platforms, IMC applications are normally light-weighted. By developing web-based online IDE (Integrated Development Environment), IMC can provide PAAS (Platform as a Service) services to users, so that end users can develop, deploy and run IMC applications online through a web browser. In this case, building a distributed IM system will be much easier, since all IM resources and commonly used IM functions can be obtained online through IMC.

### 3.5 Instrumentation and Measurement mode in the IMC

The instrumentation and measurement mode in the IMC can be depicted as in Fig. 2.

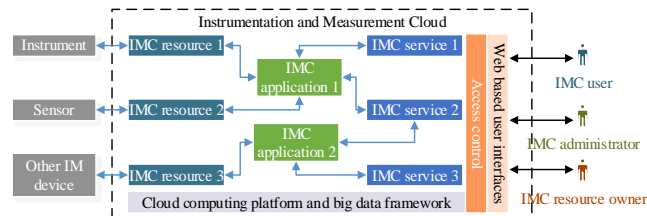


Fig. 2 Instrumentation and measurement mode in the IMC

In Fig. 2, there is an IMC user, an IMC administrator and an IMC resource owner. The IMC user normally consumes IMC resources and IMC services, and develops IMC applications to carry out IM tasks. The IMC administrator is responsible for building and maintaining the IMC platform. All IMC resources are provided and maintained by IMC resource owners. While IMC services can be developed by any of them, it is the IMC administrator's responsibility to check the quality of services and decide whether to deploy and publish them or not. As shown in Fig. 2, with a web-based user interface and access control, IMC users can request IMC resources and IMC services, and develop IMC applications online. By deploying IMC applications in the IMC platform, IMC users no longer need to invest in, establish and maintain the whole ICT (Information and Communication Technology) system for their IM tasks. Instead, they can use

resources and services provided by the IMC platform according to their need.

The above definitions have clarified the basic function requirements and characteristics of IMC. In the following section, a novel IMC architecture, which owns the above important characteristics, will be presented.

## 4 NOVEL ARCHITECTURE FOR INSTRUMENTATION AND MEASUREMENT CLOUD

The overall IMC architecture designed in this paper is shown in Fig. 3. This architecture mainly consists of six parts, which are coordination system, message broker, IMC resource agent, IMC service pool, IMC application executor and IMC manager. Details of each part will be presented in the following sections.

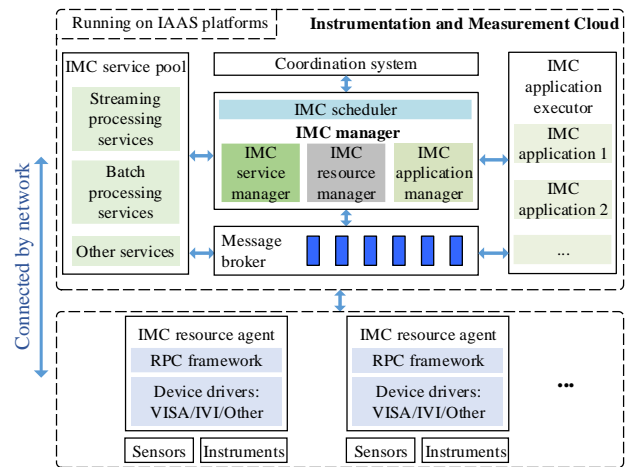


Fig. 3 Novel architecture for IMC

### 4.1 Coordination system and Message broker

The Coordination system records all configuration data and management data of living IMC services, IMC resources and IMC applications. As the IMC architecture in Fig. 3 is distributed, the coordination system should have a distributed synchronization mechanism for data manipulation. The coordination system should also support event notification, so that events from the three IMC elements can be discovered across the architecture and corresponding actions can be taken.

The message broker is the core component for data transmission. Data exchanges between IMC applications, IMC services and IMC resources are mainly achieved by the message broker. As illustrated in section 2, the effective and efficient way to transmit data in a distributed environment is via a publish/subscribe based messaging mechanism, thus a publish/subscribe based message broker is a good choice for data transmission in IMC. To transmit data over the message broker, a related client of the message broker should be integrated into IMC elements.

### 4.2 IMC resource agent

The IMC resource agent is responsible for instrument and sensor virtualization and IMC resource registration.

As illustrated in section 3.2, instruments and sensors can be virtualized through VI frameworks and standard sensor models. By virtualization, instruments and sensors are encapsulated into IMC resources with standard access interfaces. Assisted by RPC frameworks, these interfaces can be called remotely from IMC applications in the cloud, which will bring much convenience for building distributed IM systems.

Once virtualized, these IM resources can be registered into IMC by the IMC resource agent, so that users can use them over networks. To use IMC resources, users should first make a reservation for each resource through the IMC manager. After reservation, an access id with start and end time stamps will be allocated to IMC applications and the IMC resource agent, and also, the entry of the resource, such as the URL of the IMC resource agent, will be sent to IMC applications. When IMC applications want to access the reserved resources, firstly, they will have to send the access id to the IMC resource agent through that entry, and the IMC resource agent will then check if it also has the same access id and whether the current time is between the reserved time span. If all requirements are satisfied, the IMC resource agent will allocate a resource handler for the IMC application and allow the IMC application to use the IMC resource through RPC interfaces for instrumentation and measurement tasks.

The complete procedure for registering and using IMC resources in the IMC is depicted through the UML activity diagram shown in Fig. 4.

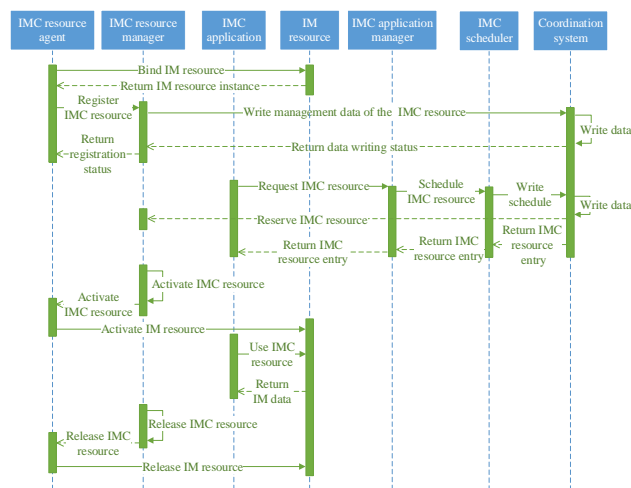


Fig. 4 Steps required for registering and consuming IMC resources in the IMC

### 4.3 IMC service pool

In IMC, services represent modularized function blocks implemented in big data analyzing and cloud computing frameworks. Such IMC services include stream data processing modules, batch processing modules and other function modules. IMC services are normally developed and deployed in parallel with distributed big data processing platforms. Each type of service can serve multiple IM applications and the platform or framework running these services will provide scaling, parallel processing and fault tolerance abilities. Services and applications in

IMC are connected by a publish/subscribe based message broker.

The message broker is used to transmit data between IMC services, IMC resources and IMC applications. Currently, many message brokers support clustering, which means brokers can support fault tolerance. However, overheads from message headings and message routing can degrade data transmission performance. To deal with this problem, some message brokers support light message headings and a simple message routing scheme. This can increase message processing and transmission speed, but at the same time reduce flexibility and functionality of the broker. Other brokers can provide more flexible control over message transmission and fault tolerance by adding extra information to messages, but this will bring more overheads. IMC service providers should choose the broker according to their needs.

All services in IMC should register themselves through the IMC manager. When registering, the IMC manager will create data entries for each IMC service in the coordination system and write management data of services into those entries. Normally, each IMC service has several input and output interfaces. For input interfaces of a service, the IMC manager will write message broker topics to which they are listening to their data entries and, if IMC applications are to consume this service, they can get those topics through the IMC manager and then publish messages to those topics. To get processed data from the IMC service, IMC applications need to write topics to which their sink modules are listening to data entries of the service's output interfaces.

As for stream data processing services, each of them has several input and output interfaces. Input interfaces will listen to dedicated message topics and, as for output interfaces, they will watch on a cache that stores destination message topics for outputting data.

For batch processing services, file systems and databases constitute the data sources. In most cases, batch processing is an off-line post-processing approach, but IM systems normally deal with real-time stream data; thus, batch-processing services will not be studied in this paper. However, batch-processing services are still indispensable to the whole IMC architecture.

To enable the parallel and distributed computing paradigm and enhance the ability of fault tolerance for IMC services, three key roles should be followed when developing IMC services:

1. Reduce coupling and dependency between data. Only in this way can the service be implemented in a parallel computing paradigm.
2. Make data self-descriptive. This is very important for multiple IMC applications to share the same service instance.
3. Try to avoid state caching for IMC applications in services and use dedicated memory cache systems. Most distributed and parallel cloud computing frameworks support fault tolerance. It means that when some processing nodes go down, the system can still run in a normal state by shifting tasks to other nodes. However, if those nodes cached states of IMC applications that they

serve, all these states will be lost and restoring the service process can be difficult, especially for streaming processing applications. Moreover, avoiding state caching in service instances can facilitate online load transfer, which is vital to load balancing.

#### 4.4 IMC application executor

The IMC application executor is responsible for running IMC applications that are deployed in the IMC. It often consists of a script interpreter or runtime engine. By deploying IMC applications into IMC, users can save the trouble of maintaining the client side. With proper user interfaces, users can access their IMC applications even through mobile terminals.

#### 4.5 IMC manager

The kernel of the whole architecture is the IMC manager. The IMC manager contains four main components: the resource manager, the service manager, the application manager and the scheduler. All IMC resources, IMC services and IMC applications are registered and managed by the corresponding component of the IMC manager. Fig. 5 shows how the IMC manager works.

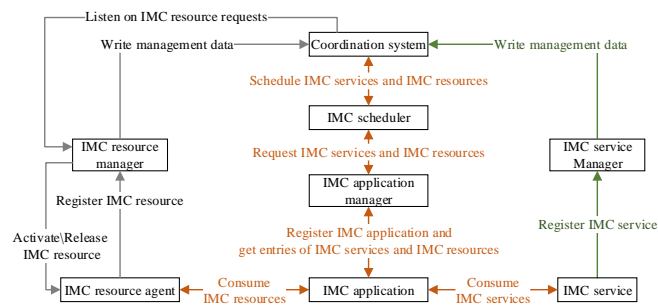


Fig. 5 Details of IMC manager

As shown in Fig. 5, when registering IMC resources, the resource manager will create a data entry for each IMC resource and store their management data. Under a RPC framework, management data often contains the URL of the resource side RPC server. Another data entry that caches all reservation information of the IMC resource is also created upon registration. Such reservation information will also be sent to the IMC resource agent for authorization purposes. A similar process happens when the service manager is registering services. However, the service manager mainly maintains management data about interfaces of each service. Service and resource requests from IMC applications are processed by the application manager. When IMC applications request services or resources, the application manager will query the coordination system, get all related management data and send these data back to applications. At the same time, some of the message broker's context of sink modules in IMC applications will be written into data entries to which output interfaces of services are listening.

The tasks of the scheduler are IMC resource and service scheduling, load balancing and scaling of IMC services. When an IMC application requests IMC resources, the

scheduler will call related algorithms to calculate a valid time span for each IMC resource and make reservations. Load balancing and service scaling is done by online load transfer and scaling the cluster that runs the service. However, online load transfer needs the support from both the framework that runs the IMC service and the IMC service itself.

Whenever an IMC application is registered in the IMC, the IMC application manager will record all consumed services and resources. If the state of services or resources changes, an event will be sent to the application manager to trigger the state transition process of the IMC application. In addition, a change of the IMC application state can trigger state transition of related IMC resources. State transition models for IMC service, IMC resource and IMC application in IMC are shown in Fig. 6.

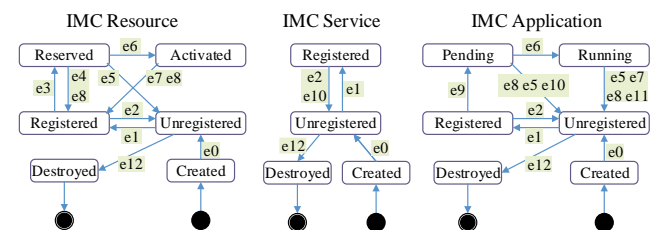


Fig. 6 State transition models of IMC service, IMC resource and IMC application in IMC. In the figure events are: e0-add element, e1-register, e2-unregister, e3-reserve, e4-cancel reservation, e5-resource invalid, e6-resouce available, e7-resource expired, e8-application invalid, e9-application start, e10-service invalid, e11-application terminated, e12-destroy element.

Fig. 6 shows only very basic state transition models for the IMC architecture; however, implementation of the IMC architecture will need detailed models to handle situations that are more complicated.

To demonstrate the feasibility and advantages of the IMC architecture brought forward by this paper, a practical system implemented upon this architecture is presented.

## 5 IMPLEMENTATION OF THE IMC ARCHITECTURE

### 5.1 Coordination system and Message broker

The coordination system is used to record important configuration data and management data about IMC resources, IMC services and IMC applications in the IMC. Here, Zookeeper, which is a distributed coordination system with fault tolerance ability, is utilized to store various data. Zookeeper uses a data structure called Zookeeper path that is similar to the directory of a file system and where each node of a path can store data.

Various message brokers can be used in the IMC and, in this paper, Rabbitmq is adopted just for demonstration.

### 5.2 VISA based IMC resource agent

First, to verify that the architecture is viable for instrument and sensor virtualization, a VISA (Virtual Instrument Software Architecture) based IMC resource agent is implemented. The agent uses Apache Thrift RPC framework and all VISA driver interfaces are remotely mapped. Since Thrift supports cross language service development,

it is also used as the service framework between all servers and clients in the IMC. Fig. 7 shows the details of the implemented IMC resource agent.

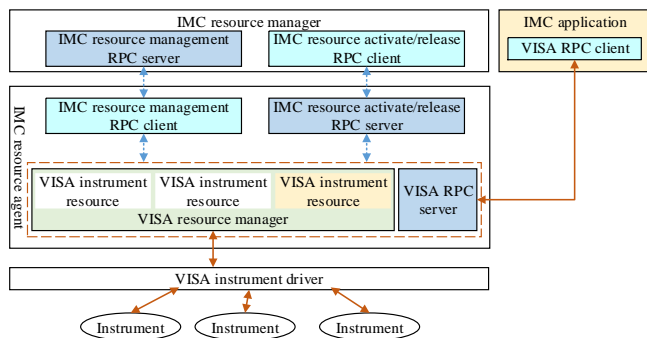


Fig. 7 Implementation of the IMC resource agent

As shown in Fig. 7, instrument resources are registered through IMC resource management RPC services, which are implemented in the Thrift framework. To access the instrument, each IMC resource agent needs to run a VISA RPC server and it wraps all VISA driver interfaces. However, those interfaces are extended to include a third parameter, which is the access ID. Such access ID contains both the name of the IMC resource and the reserved time span. The IMC resource agent will also store a set of  $\langle$ access ID, VISA instrument resource $\rangle$  maps and these maps are built up when IMC applications request IMC resources managed by this IMC resource agent.

Once the IMC application in the IMC needs to control an IMC resource, it will make a remote call with the access ID. On the IMC resource agent side, the agent will get the corresponding VISA instrument resource through this ID and call a local VISA instrument driver to control the instrument and then return the result to the IMC application. To make the agent as independent with OS platforms as possible, pyvisa, which is a python implemented frontend of VISA driver, is used as the VISA resource manager. Although instruments and sensors are diverse in their hardware architectures and software interfaces, the similar implementation method can be applied to virtualize them into an IMC resource as long as their interfaces are available.

As for IMC resources, the Zookeeper data paths used to record their information are shown in Fig. 8. The `/instcloud/resources` path is the root path for IMC resource management. When registering an IMC resource, the resource manager of the IMC manager will create a data path according to the domain, site and the resource name. Here domain and site are used to constitute a two-level naming convention so that resource management can be more convenient and flexible. Whenever a reservation is made for an IMC resource, a duration node with a start

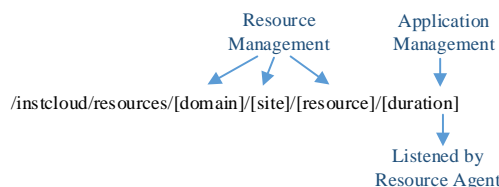


Fig. 8 Management data for the IMC resource

and an end timestamp will be added as a child to the resource node as long as the reservation is valid. The IMC resource manager will listen to such child-adding event and call the PRC client to activate the IMC resource.

### 5.3 Storm based IMC service for stream data processing

The next stage is to implement IM services. As explained before, IM tasks normally need to process real-time stream data, thus a stream data processing engine is required and related IM function modules need to be developed. In the work of this paper, Apache Storm [34] is used as the real-time stream data processing engine. Storm is a distribute parallel stream data processing engine, whose maximum processing speed can reach around 1 million messages per second, with fault tolerance ability. Storm contains two types of components, which are Spout and Bolt. Spout is the data source and it can be connected to a message broker to subscribe message topics. Bolt is the process component. Spouts and Bolts compose a Topology, which carries out the stream data processing logic. Each Spout or Bolt can run multiple instances so that they can process data in parallel. In the IMC, topologies are wrapped into IMC services, with certain Spouts functioning as input interfaces and some Bolts as output interfaces. Fig. 9 presents a simple example for computing electrical power to show how services in the IMC interact with IMC applications and the IMC manager in the cloud.

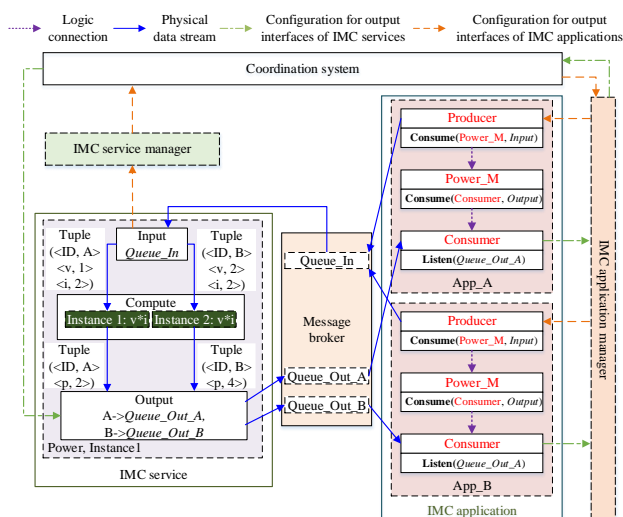


Fig. 9 An IMC service implemented through Storm to compute electrical power

In Fig. 9, input interfaces of an IMC service are implemented by instances of IMCSpout class, which is an extended Spout class with a service management RPC client. When an IMC service is submitted, each IMCSpout instance will register the context of the message topic that this IMCSpout is listening to through the RPC client. Since Rabbitmq is used as the message broker, the detailed context of a message topic is wrapped into the RMQContext class. To consume an IMC service, an IMC application just needs to get the RMQContext instance of

each input interface, create a Rabbitmq client corresponding to that context, and send data to the IMC service input interface through that Rabbitmq client.

Output interfaces are instances of an extended Bolt class named IMCBolt class with a service management client, and they will also create management data entries in Zookeeper and listen to those data paths. However, it is the IMC application manager that is responsible for writing message topic contexts to those paths. Whenever data on those paths are updated, output interfaces of an IMC service will update destination contexts. Each destination context in an output interface is combined with a session ID and a module ID, which will be introduced in the following part, and each message passed to an output interface will also contain these IDs. With these IDs, output interfaces will know which message topic the data should be sent to. Data paths for IMC services are shown in Fig. 10.

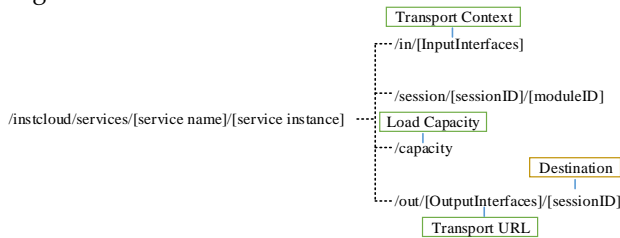


Fig. 10 Management data for IMC services

In Fig. 10, `/instcloud/services` is the root data path for IMC service management. Under the root path are service name and instance name data path. Children under `/in` node are input interfaces of the IMC service and, similarly, children under `/out` node are output interfaces. Each child node under `/in` node will store a Transport Context which records the context of the message topic that the interface is listening to. Each child node under `/out` will store a Transport URL that tells the output interface which message broker is used for data transmission. Under each node of output interface, there are child nodes representing IMC application sessions that consume output data from the output interface and each of the child nodes will store a Destination Object that records contexts of message topics for output data transmission.

The `/session` node and its children under each topology or service instance node are used for IMC service state management. For example, when an IMC application session releases a service, the corresponding session ID and module ID will be deleted and such deleting events will be listened to by the IMC service manager and related IMC service instances. Once such event happens, both the IMC service manager and related IMC service instances will clear cached data for that session.

The `/capacity` node stores load capacity of this service and the IMC manager will use this data for load balance and scaling.

#### 5.4 IMC application developing mode

IMC applications are implemented through text programming language and currently only Java APIs (Application Programming Interfaces) have been developed. Four clas-

ses are defined according to entities in the IMC, which are ICResource, ICService, StreamProducer and StreamConsumer. ICResource and ICService are wrappers of the IMC resource and the IMC service respectively.

When creating an ICResource object, the domain, site, resource name and time span should be specified, but for the ICService object, only service name is required. The ICResource class is normally wrapped with a RPC client that is used to control IMC resources.

StreamProducer is used to publish data to the input interfaces of an IMC service, while StreamConsumer is responsible for receiving data from an IMC service. However, all related message broker contexts are automatically set when submitting the IMC application. A complete IM process is wrapped into a session task and all related IMC resources, IMC services and other modules are managed through a Session object. All the four types of components in an IMC application session have their module IDs and each session has a universally unique ID. However, each module ID is only unique to a session. Fig. 11 shows a diagram of a simple IMC application, which is also referred to in Fig. 9, just for illustration.

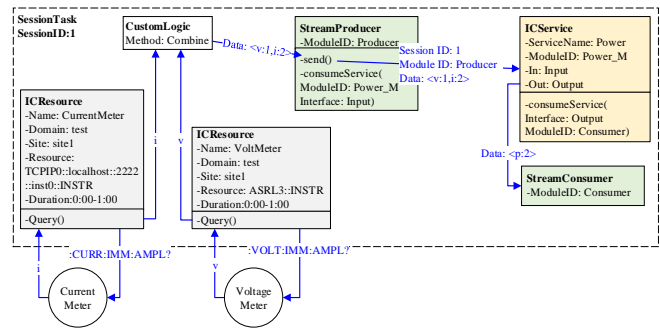


Fig. 11 An IMC application example

In Fig. 11, there is a `consumeService` method that is used to decide which module and interface the data of the current module should be sent to. For the StreamProducer object, the name of the next module and the interface should also be provided, while for the `consumeService` method of the ICService object, the name of the service output interface and the next module and its input interface, if any, should be defined.

#### 5.5 Implementation of the IMC Manager

Implementation of the IMC manager is shown in Fig. 12. In Fig. 12, the IMC manager needs to interact with Zookeeper, IMC resource agents, IMC applications and IMC services. In the implementation of this paper, Curator Framework is used to interact with Zookeeper. Most of the other remote

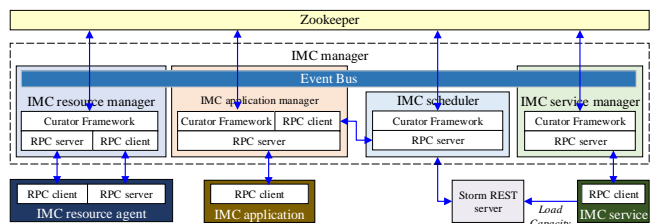


Fig. 12 Implementation of the IMC manager



management operations are carried out through the Apache Thrift RPC framework.

To obtain load capacities of Storm-based IMC services and schedule IT resources for those IMC services, the IMC scheduler will call the Storm REST (Representational State Transfer) API. Each module of the IMC manager in Fig. 12 can run a thread pool for its RPC server, so that it can serve multiple RPC clients.

Currently, the IMC manager is implemented in a centralized way just to verify the feasibility and functionality of the IMC architecture in this paper. As shown in Fig. 12, the IMC resource manager, the IMC service manager, the IMC application manager and the IMC scheduler are loosely coupled through an event bus. In this way, the IMC manager can manage the state of each IMC element according to Fig. 6.

However, each module of the IMC manager is relatively independent, thus it will be very easy to implement the IMC manager in a distributed way, e.g. substitute event bus with message broker.

## 6 APPLICATION AND EXPERIMENT

To test the designed IMC architecture, an application for power system state estimation (SE) [35] is developed based on the implemented system.

### 6.1 Distributed parallel power system state estimation

In a power system, not all states of the system are monitored and some measured states may not be correct. However, all real states of a power system should always obey the law of electric circuits. Thus, using these electric circuit laws and measured data, a group of measurement equations can be built up. With these equations, a mathematic model can be developed to estimate the true states of the whole power system. The most fundamental state estimation method is the WLS (Weighted least square) method [36], also referred to as the NE (Normal equation) method. The mathematical model for the WLS method can be formulated through Eqs. (1) - (4) below:

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{v} \quad (1)$$

where  $\mathbf{z}$  is the  $m$ -dimensional measurement vector,  $\mathbf{x}$  is the  $2n-2$  dimensional state variable vector,  $\mathbf{v}$  is the  $m$ -dimensional measurement error vector and  $\mathbf{h}(\mathbf{x})$  is the relation function of measurements and the state variable  $\mathbf{x}$ .

$$\begin{cases} J(\mathbf{x}) = [\mathbf{z} - \mathbf{h}(\mathbf{x})]^T \mathbf{W} [\mathbf{z} - \mathbf{h}(\mathbf{x})] \\ \mathbf{W} = \text{diag}[\frac{1}{\sigma_1^2}, \dots, \frac{1}{\sigma_i^2}, \dots, \frac{1}{\sigma_m^2}] \end{cases} \quad (2)$$

where  $J(\mathbf{x})$  is the objective function,  $\mathbf{W}$  is the  $m \times m$  diagonal weight matrix, and  $\sigma_i^2$  is the covariance of the  $i$ th measurement error.

$$\Delta \mathbf{z}(\mathbf{x}) = \mathbf{z} - \mathbf{h}(\mathbf{x}) \quad (3)$$

where  $\Delta \mathbf{z}(\mathbf{x})$  is the measurement residual vector.

$$\begin{cases} \mathbf{H}^T(\mathbf{x}_k) \mathbf{W} \mathbf{H}(\mathbf{x}_k) \Delta \mathbf{x}_{k+1} = \mathbf{H}^T(\mathbf{x}_k) \mathbf{W} \Delta \mathbf{z}(\mathbf{x}_k) \\ \mathbf{H}(\mathbf{x}_k) = \frac{\partial \mathbf{h}(\mathbf{x}_k)}{\partial \mathbf{x}} \end{cases} \quad (4)$$

where  $\mathbf{H}(\mathbf{x}_k)$  is the value of the Jacobian matrix of  $\mathbf{h}(\mathbf{x})$  in which  $\mathbf{x} = \mathbf{x}_k$ ,  $\Delta \mathbf{x}_{k+1}$  is the correction vector for the estimated state  $\mathbf{x}_k$  and  $k$  is the iteration index.

The WLS state estimation method is carried out by iterat-

ing (4).

As there are bad data among measurements, a bad data recognition process is required to find and eliminate bad measurements. A commonly used method is the normalized residual recognition method [37]. This method uses Eqs. (5)-(6) below to detect and identify bad measurement data.

$$\mathbf{r}_N = \left( \mathbf{W} - \mathbf{H}(\mathbf{x})(\mathbf{H}^T(\mathbf{x})\mathbf{W}^{-1}\mathbf{H}(\mathbf{x}))^{-1}\mathbf{H}^T(\mathbf{x}) \right)^{-\frac{1}{2}} \Delta \mathbf{z}(\mathbf{x}) \quad (5)$$

where  $\mathbf{r}_N$  is the normalized residual vector for measurement vector  $\mathbf{z}$ .

$$\gamma = (\sum_{i=1}^m \sigma_i^2)^{\frac{1}{2}} \quad (6)$$

where  $\gamma$  is the threshold for bad data recognition.

If the  $i$ th element  $r_{Ni}$  of  $\mathbf{r}_N$  is greater than  $\gamma$ , then the corresponding measurement  $z_i$  is taken as the bad data suspect. The measurement, which has the largest normalized residual larger than  $\gamma$ , is taken as the prime bad data suspect. Once bad measurement data are detected, they should be discarded and the state estimation process should restart. Normally, bad measurement data will not be discarded at one time, instead, only the bad data that has the prime suspect is discarded. The procedure of state estimation with bad data recognition for power systems is shown in Fig. 13.



Fig. 13 Power system state estimation procedure

In Fig. 13, the state estimation step is actually carrying out the iteration process formulated in (4), while the bad data recognition step is calculating  $\mathbf{r}_N$  and comparing its elements to  $\gamma$ .

Nevertheless, the estimation process is quite computation-intensive and consumes lots of memory, especially when the system is large. Still, the state estimation process is required to be as fast as possible. To achieve this goal, the parallel state estimation method [38] is introduced.

The parallel state estimation method is based on system partitioning. To implement a parallel state estimation algorithm, a large system should, firstly, be partitioned into several subsystems where all subsystems can estimate their own state in parallel. Each subsystem can use the aforementioned WLS method to estimate state except that the iteration process is different. In a parallel state estimation algorithm, after each iteration, all subsystems should share the states of boundary buses before the next iteration. The basic procedure of parallel state estimation algorithm is depicted in Fig. 14.

In Fig. 14, Step 2 carries out the iteration process formulated in (4) while Step 6 calculates  $\mathbf{r}_N$  and compares its elements to  $\gamma$ . Iteration and bad data recognition tasks from multiple subsystems can be distributed in a cluster to accelerate the overall estimation speed.

### 6.2 IMC-based power system state estimation

Since the IMC platform can greatly facilitate building distributed IM systems and provide efficient big data processing frameworks, this section tries to establish an

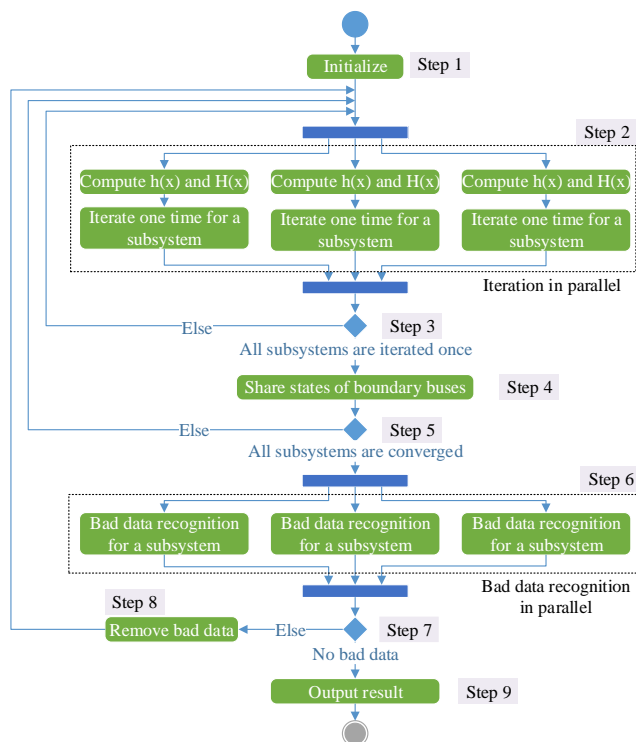


Fig. 14 Parallel state estimation for a large power system

IMC-based power system state estimation system. Such a system will not only make state estimation easier and more efficient, but can also test the functionalities of the IMC architecture brought up in this paper.

According to the IMC architecture elaborated on before, three aspects should be considered when developing an IMC-based state estimation system: (1) Virtualizing the measurement system into an IMC resource; (2) Implementing the parallel state estimation algorithm into an IMC service; (3) Developing the state estimation IMC application.

**(1) Virtualizing the measurement system into an IMC resource**

There are eight types of electric variables that need to be measured for state estimation. Since the test cannot be carried out in a real power system for safety reasons, all measurements are simulated by adding Gaussian noise to system power flow data.

As the implemented IMC resource agent is based on VISA, here, a meter that conforms to VISA standard is implemented for each power system through pyvisa-sim, which is an instrument simulation software for pyvisa, and custom defined instrument commands, such as ?MEA, are used to retrieve measurement data from the simulated meter. In this way, the measurement system of each power system can be virtualized into an IMC resource. However, in practical applications, a corresponding IMC resource agent should be developed according to the physical measurement system.

Once the measurement system of a power grid is virtualized into an IMC resource, a state estimation IMC application can retrieve all measured data by sending instrumentation commands through the RPC framework.

**(2) Implementing the parallel state estimation algorithm into an IMC service**

The parallel state estimation algorithm is implemented in-

to an IMC service, which can be consumed by different state estimation IMC applications to estimate states of multiple power systems simultaneously. According to Fig. 14, a Storm-based state estimation IMC service can be developed as shown in Fig. 15.

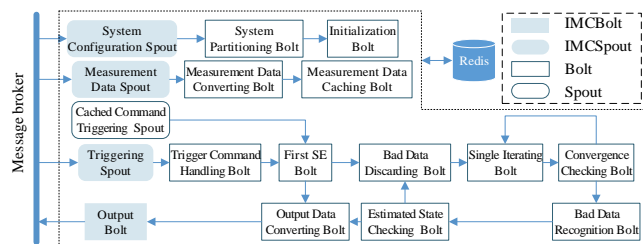


Fig. 15 Storm-based IMC service for power system state estimation

The IMC service mainly contains three subsystems. The first one is the system partitioning subsystem. This subsystem receives the configuration data of a power system from the SE IMC application through the System Configuring Spout. It is responsible for partitioning a large power system into small subsystems and initializing various parameters for SE. In this test, KaHIP [39], which is an effective and fast graph partitioning tool, is used to partition the power system. Also the topology is implemented through JAVA; however, JAVA is not quite suitable for matrix computation, thus all matrix-related computations are carried out by Matlab. Partitioned data and initialized parameters are all stored in Redis [40], which is an in-memory database, so that computation processes can be separated from data, which is one of the rules for developing an IMC service. Redis uses the key-value data structure, and it is fast and very suitable for big data processing. As an IMC service is shared by multiple IMC applications, when caching data, a key should be attached to each data to identify its origin and that is why Redis is chosen as the caching database. Here, most of the intermediate data are cached in Redis.

The second subsystem is the measurement subsystem. It converts raw measurement data into formatted data, so that they can be used for state estimation. Raw measurement data are acquired from the IMC resource corresponding to the virtualized measurement system of a power system by the SE IMC application, and then sent to the Measurement Data Spout through the message broker. The Measurement Data Converting Bolt will convert the data and then send the converted data to the Measurement Data Caching Bolt to store the data into Redis.

The third subsystem is the SE subsystem. This subsystem implements the parallel SE algorithm as shown in Fig. 14. Whenever a power system needs to estimate state, it can just send a trigger command with the ID of the power system to the SE subsystem through the Triggering Spout. After receiving the trigger command, the Trigger Command Handling Bolt of the SE subsystem will check if the power system that requested SE is currently being estimated. If so, the Trigger Command Handling Bolt will cache the command for the latter SE triggering, otherwise it forwards the command to the First SE Bolt. The Cached Command Triggering Spout will check all cached commands similar to the Com-

mand Handling Bolt periodically and send a valid command to the First SE Bolt. The First SE Bolt will then compute  $J(\mathbf{x})$  and compare it to the corresponding threshold to see if the initial state is the estimated state. If  $J(\mathbf{x})$  is below the threshold, the initial state is the estimated state and it will be forwarded to the Output Data Converting Bolt to convert the data for output, otherwise the SE trigger command will be sent to the Bad Data Discarding Bolt to start a new state estimation process.

The Single Iterating Bolt, the Convergence Checking Bolt, the Bad Data Recognition Bolt, the Estimated State Checking Bolt and the Bad Data Discarding Bolt implement step 2, steps 3-5, step 6, step 7 and step 8 in Fig. 14, respectively. However, there are three main differences.

Firstly, the Bad Data Discarding Bolt has two inputs, which are from the First SE Bolt and the Estimated State Checking Bolt. If the input is from the First SE Bolt, the Bad Data Discarding Bolt will do nothing but send the input command to the Single Iterating Bolt, otherwise it will discard bad measurement data and send a command to the Single Iterating Bolt for new SE.

Secondly, as estimated states from each iteration are cached in Redis, all states will be updated immediately, which means states of boundary buses are shared in time by the Single Iterating Bolt and that is why no Bolt for step 4 in Fig. 14 is implemented.

Thirdly, each of the Bolts in the SE subsystem can serve different power systems and do not bind to one specific power system. Such a loosely coupled processing characteristic is achieved by caching data in Redis and retrieving corresponding data of each power system through keys such as the ID of a power system.

Parallel iteration and bad data estimation are implemented by running multiple instances of corresponding Bolts. Once the state of a power system is estimated, it will be sent back to the corresponding SE IMC application through the Output Bolt.

Once the topology of the SE IMC service is developed, it can be deployed onto the IMC platform, and the SE IMC service can be started.

### (3) Developing the state estimation IMC application

With the measurement system being virtualized and the SE IMC service running, to carry out state estimation, end users just need to develop a very simple SE IMC application to acquire measurement data of a power system from the corresponding IMC resource and send those data to the SE IMC service for SE online. Detailed activities of the SE IMC application are shown in Fig. 16.

In Fig. 16, IMC resources and IMC services are defined through instances of ICRsource class and ICSservice class respectively. Data are sent by instances of StreamProducer class and are received by instances of StreamConsumer class.

## 6.3 System deployment and test scenarios

The whole system runs on an Openstack IaaS cloud platform. Each VM that runs a supervisor node of Storm is configured with E3-12XX/2.3GHz CPU and 8GB RAM.

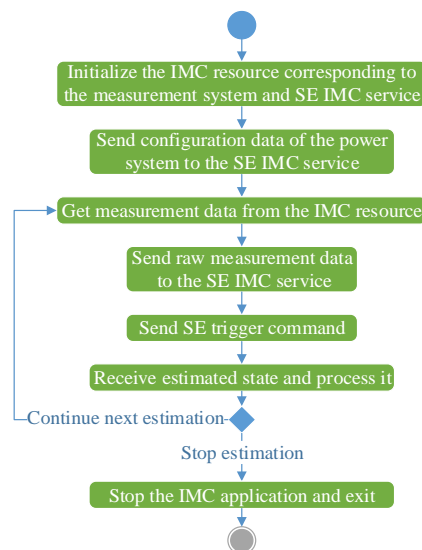


Fig. 16 Activity diagram of the SE IMC application

VMs for other nodes in the IMC platform are configured with the same type of CPU as Storm supervisor node but with only 4GB RAM. Detailed physical and logical configurations of the system are shown in Fig. 17.

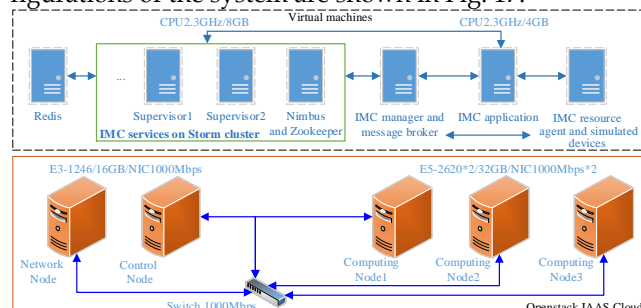


Fig. 17 Detailed configurations of the IMC system

In this test, case data from Matpower [41] are used and two test scenarios are set. In the first test scenario, state estimation for case3120sp is tested and the SE IMC service is exclusive to case3120sp. The number in the case name represents the number of buses in the system. This scenario is used to test the functionality of the implemented IMC platform to test the IMC architecture brought up in this paper. In the second test scenario, state estimations for case2869pegase, case3012wp and case3120sp are tested simultaneously and the SE IMC service is shared by multiple cases. This scenario is used to test the service sharing ability of the IMC architecture. Systems of these cases are very large in the electrical engineering field and, when running the parallel SE algorithm, each system is split into several subsystems. The number of buses in each subsystem is limited to 300 when splitting the system.

The most computation intensive Bolts are the Single Iterating Bolt and the Bad Data Recognition Bolt, and, while these two Bolts are under full load, all load capacities of other Bolts in Fig. 15 are less than 5%. Thus, multiple workers are set for the instances of these two Bolts. To find out which Bolt plays a more important role in determining the total estimation time  $T_{se}$ , different numbers of workers are set for each of the Bolts.

### 6.4 Test results

The result for the first test scenario is shown in Fig. 18.

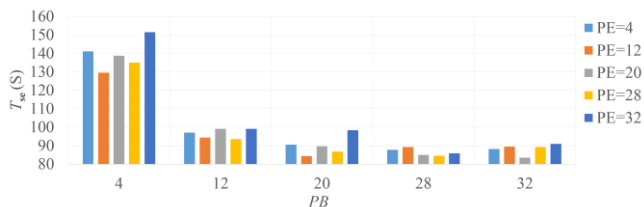


Fig. 18 State estimation time for case3120sp using the IMC platform

In Fig. 18,  $PB$  is the number of workers for the Bad Data Recognition Bolt and  $PE$  is the number of workers for the Single Iterating Bolt. Fig. 18 shows that, when increasing  $PB$ , estimation time will reduce dramatically. Fig. 18 also shows that, when  $PB$  is fixed, changing  $PE$  does not effectively influence the estimation time.

However, when  $PB$  increases to 12 or larger, the estimation time does not decrease any more. This phenomenon is caused by the overheads from distributed computing, such as data transmission delay between computing nodes.

Fig. 18 demonstrates that the bad measurement data recognition process is much more time-consuming than the estimation process. Fig. 18 also shows that sometimes increasing  $PB$  or  $PE$  may cause performance degradation and that is because instances of Bolt may distribute across different computing nodes. When more directly connected Bolt instances are running on the same node, communication delay can be reduced. However, if more directly connected Bolt instances distribute across different nodes, overheads from communication will degrade overall performance. Currently, the distribution of Bolt instances is carried out automatically by Storm and that is the reason for performance fluctuation in Fig. 18. A similar phenomenon will also happen in the second test scenario.

The result of the first test shows that the IMC architecture brought up in this paper is feasible and satisfies the basic functional requirements defined in section 3.

In the second test scenario, the results for each case alone are similar to Fig. 18; thus, those results are not presented. However, comparing the estimation time of case3120sp in the two different test scenarios is important, since it can reveal some characteristics of shared IMC services. Fig. 19 shows  $T_{se}$  of case3120sp in exclusive service mode and in shared service mode.

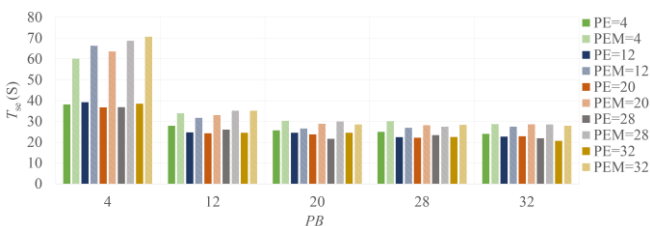


Fig. 19 Comparing  $T_{se}$  of case3120sp in exclusive service mode and in shared service mode

In Fig. 19,  $PEM$  designates the number of workers for the Single Iterating Bolt in the shared service mode. Fig.

19 shows that when  $PB$  is small, which means computing resource for the SE IMC service is inadequate,  $T_{se}$  is much higher in shared service mode than in exclusive service mode. However, when  $PB$  increases,  $T_{se}$  in shared service mode is only 10%-20% higher than in exclusive service mode. This demonstrates that, when computing resource is not strained, performance of the SE IMC service in shared service mode is very close to that in exclusive service mode, which means resource utilization efficiency can be greatly increased through service sharing in the IMC platform.

Such a resource utilization efficiency improvement is normally attributed to the overheads from synchronization of the parallel algorithm. For example, in the parallel SE algorithm shown in Fig. 14, step 2 or step 6 has to wait until all subsystems are processed to continue, and during this period, some of the computing resource will be idle if the service implemented upon this parallel algorithm is exclusive. However, if the service is implemented in shared mode, the idle resource can be used for SE of other power systems, thus resource utilization efficiency can be improved.

Comparing the results from the above two test scenarios demonstrates that the IMC architecture in this paper is viable for service sharing that can improve resource utilization efficiency.

## 7 DISCUSSION

Although the IMC brought up in this paper can greatly facilitate utilization and management of IM resources, limitations and challenges still exist.

Firstly, the IMC is not suitable for those application scenarios that are highly time-critical and require extremely high reliability. Such limitation is caused by the latency and fluctuation of networks, and overheads from message brokers, RPCs and distributed parallel computing frameworks. Secondly, high-speed and high-precision IM devices normally produce large amounts of data in a very short time, and directly transferring those large amounts of raw data in real time from IM devices to the IMC may be impossible due to the bandwidth limitation of the network. Thirdly, frequent remote procedure calls can bring a lot of overheads, especially for remote IM device control with short intervals.

Currently, a promising solution for the first and second challenges above is adopting fog computing [42] paradigms as a complementary framework between the IMC layer and the physical IM device layer. Fog computing is more focused on proximity to client objectives and end users, which leads to less latency and higher reliability. By pre-processing data locally or through fog computing, the amount of data that need to be transferred over the network can be greatly reduced, which will lower the requirement for the bandwidth of the network. Also, time-critical tasks can be carried out in a fog computing framework and other computation intensive tasks can be shifted to the IMC. In this way, the overall performance and QoS (Quality of Service) can be greatly improved.

To solve the third problem, frequent remote procedure

calls can be substituted by direct interaction between local devices and the IMC services. RPCs can just be used to manipulate the interaction process rather than relay data between IM devices and the IMC.

## 8 CONCLUSION AND FUTURE WORK

The instrumentation and measurement cloud can greatly facilitate management of instruments and sensors and, at the same time, allows users to utilize those resources and related IM services on demand remotely.

The IMC architecture brought forward in this paper provides efficient guidance for developing a practical IMC platform. With IM device virtualization and service wrapping, building a remote IM system just requires only very simple coding work. Most of the investment in IT facilities and system development work can be saved. Furthermore, with the ability to scale and to dynamically transfer load, the IMC can increase the utilization efficiency of various resources to a much higher level. Distributed parallel computing paradigms of the IMC will accelerate the processing speed, which brings lots of benefits for large-scale remote IM systems and, also, for analysis of big data coming from huge numbers of instruments and sensors. The application developed upon the implemented system has demonstrated the advantages of the work in this paper.

However, more research work is still required to deal with some challenges. Such challenges include latency and stability of networks, geographic characteristics of the physical object that is being measured, and so on. These challenges are not proprietary to the IMC but common in the remote instrumentation and measurement field. But with more effort, problems caused by these challenges can eventually be solved.

## ACKNOWLEDGMENT

This work was funded by the State Key Laboratory of Power System award SKLD15M02, Department of Electrical Engineering, Tsinghua University, Beijing, China. Wei Zhao is the corresponding author.

## REFERENCES

- [1] M. Bertocco, "Architectures For Remote Measurement," Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements, F. Davoli, S. Palazzo, and S. Zappatoreed, eds: Springer US, pp. 349-362, 2006.
- [2] A. Roversi, A. Conti, D. Dardari, and O. Andrisano, "A WEB-Based Architecture Enabling Cooperative Telemeasurements," Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements, F. Davoli, S. Palazzo, and S. Zappatoreed, eds: Springer US, pp. 395-407, 2006.
- [3] A. Cheptsov, B. Koller, D. Adami, F. Davoli, S. Mueller, N. Meyer, P. Lazzari, S. Salon, J. Watzl, M. Schiffers, and D. Kranzmueller, "E-Infrastructure for Remote Instrumentation," Computer Standards & Interfaces, vol. 34, no. 2012, pp. 476-484, 2012.
- [4] S. Qin, X. Liu, and L. Bo, "Study on the Networked Virtual Instrument and Its Application," Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2005. IDAACS 2005.

- IEEE, pp. 337-339, 2005.
- [5] Y. Jadeja and K. Modi, "Cloud computing - concepts, architecture and challenges," Computing, Electronics and Electrical Technologies (IC-CEET), 2012 International Conference on, pp. 877-880, 2012.
- [6] I. Foster, Z. Yong, I. Raicu, and L. Shiyong, "Cloud Computing and Grid Computing 360-Degree Compared," Grid Computing Environments Workshop, pp. 1-10, 2008.
- [7] M. Fazio, M. Paone, A. Puliafito, and M. Villari, "Huge amount of heterogeneous sensed data needs the cloud," Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on, pp. 1-6, 2012.
- [8] W. Rekik, M. Mhiri, and M. Khemakhem, "A smart cloud repository for online instrument," 2012 International Conference on Education and e-Learning Innovations (ICEELI), pp. 1-4, 2012.
- [9] A. S. McGough and D. J. Colling, "The GRIDCC Project," First International Conference on Communication System Software and Middleware, pp. 1-4, 2006.
- [10] M. Yuriyama and T. Kushida, "Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing," 13th International Conference on Network-Based Information Systems (NBIS), pp. 1-8, 2010.
- [11] R. Di Lauro, F. Lucarelli, and R. Montella, "SaaS - Sensing Instrument as a Service Using Cloud Computing to Turn Physical Instrument into Ubiquitous Service," IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 861-862, 2012.
- [12] F. Lelli, "Bringing Instruments to a Service-Oriented Interactive Grid," Ph.D Thesis, Dipartimento di Informatica, Università Ca' Foscari di Venezia, Venice, Italy, 2007.
- [13] G. C. Fox, R. Guha, D. F. McMullen, A. F. Mustacoglu, M. E. Pierce, A. E. Topcu, and D. J. Wild, "Web 2.0 for Grids and e-Science," Grid Enabled Remote Instrumentation, F. Davoli, N. Meyer, R. Pugliese, and S. Zappatoreed, eds: Springer US, pp. 409-431, 2009.
- [14] E. Frizziero, M. Gulmini, F. Lelli, G. Maron, A. Oh, S. Orlando, A. Petrucci, S. Squizzato, and S. Traldi, "Instrument Element: a new grid component that enables the control of remote instrumentation," Sixth IEEE International Symposium on Cluster Computing and the Grid, pp. 44-52, 2006.
- [15] I. M. Atkinson, D. d. Boulay, C. Chee, K. Chiu, P. Coddington, A. Gerson, T. King, D. F. McMullen, R. Quilici, P. Turner, A. Wendelborn, M. Wyatt, and D. Zhang, "Developing CIMA-based cyberinfrastructure for remote access to scientific instruments and collaborative e-Research," Conferences in Research and Practice in Information Technology Series, Ballarat, Australia, pp. 3-10, 2007.
- [16] M. Okon, D. Kaliszan, M. Lawenda, D. Stokosa, T. Rajtar, N. Meyer, and M. Stroinski, "Virtual laboratory as a remote and interactive access to the scientific instrumentation embedded in Grid environment," 2nd IEEE International Conference on e-Science and Grid Computing, Amsterdam, Netherlands, pp. 1-5, 2006.
- [17] L. Berruti, F. Davoli, and S. Zappatore, "Performance evaluation of measurement data acquisition mechanisms in a distributed computing environment integrating remote laboratory instrumentation," Future Generation Computer Systems, vol. 29, no. 2, pp. 460-471, 2013.
- [18] M. Yuriyama, T. Kushida, and M. Itakura, "A New Model of Accelerating Service Innovation with Sensor-Cloud Infrastructure," Annual SRII Global Conference (SRII), pp. 308-314, 2011.
- [19] G. Merlino, D. Bruneo, S. Distefano, F. Longo, and A. Puliafito, "Stack4Things: integrating IoT with OpenStack in a Smart City context," Smart Computing Workshops (SMARTCOMP Workshops), 2014 International Conference on, pp. 21-28, 2014.
- [20] G. Merlino, D. Bruneo, S. Distefano, F. Longo, A. Puliafito, and A. Al-Anbuky, "A smart city lighting case study on an openstack-powered in-

frastructure," *Sensors*, vol. 15, no. 7, pp. 16314-16335, 2015.

[21] K. Ahmed and M. Gregory, "Integrating Wireless Sensor Networks with Cloud Computing," 2011 Seventh International Conference on Mobile Ad-hoc and Sensor Networks, pp. 364-366, 2011.

[22] M. S. Aslam, S. Rea, and D. Pesch, "Service Provisioning for the WSN Cloud," *Cloud Computing (CLOUD)*, 2012 IEEE 5th International Conference on, pp. 962-969, 2012.

[23] Y. Byunggu, A. Cuzzocrea, J. Dong, and S. Maydeburu, "On Managing Very Large Sensor-Network Data Using Bigtable," *Cluster, Cloud and Grid Computing (CCGrid)*, 2012 12th IEEE/ACM International Symposium on, pp. 918-922, 2012.

[24] Y. Chang Ho, H. Hyuck, J. Hae-Sun, Y. Heon Young, and L. Yong-Woo, "Intelligent Management of Remote Facilities through a Ubiquitous Cloud Middleware," *IEEE International Conference on Cloud Computing*, pp. 65-71, 2009.

[25] S. Distefano, G. Merlino, and A. Puliafito, "Sensing and Actuation as a Service: A New Development for Clouds," 11th IEEE International Symposium on Network Computing and Applications (NCA), pp. 272-275, 2012.

[26] Y. Kang, Z. Wei, and H. Song-ling, "Discussion on a new concept of measuring instruments-Instrument Cloud," *China measurement and test*, vol. 38, no. 2, pp. 1-5, 2012.

[27] V. Rajesh, O. Pandithurai, and S. Mageshkumar, "Wireless sensor node data on cloud," *IEEE International Conference on Communication Control and Computing Technologies (ICCCCT)*, pp. 476-481, 2010.

[28] Y. Takabe, K. Matsumoto, M. Yamagiwa, and M. Uehara, "Proposed Sensor Network for Living Environments Using Cloud Computing," 15th International Conference on Network-Based Information Systems (NBIS), pp. 838-843, 2012.

[29] G. Zhongwen, L. Chao, F. Yuan, and H. Feng, "CCSA: A Cloud Computing Service Architecture for Sensor Networks," *International Conference on Cloud and Service Computing (CSC)*, pp. 25-31, 2012.

[30] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010.

[31] H. He, Z. Wei, and S. Huang, "Future trend of integrating instrumentation into the cloud," *Proceedings of International Workshop on Cloud Computing & Information Security*, vol. 52, no. 2, pp. 359-365, 2013.

[32] T. Hirofuchi, E. Kawai, K. Fujikawa, and H. Sunahara, "USB/IP: A Transparent Device Sharing Technology over IP Network," *IPSJ Digital Courier*, vol. 1, pp. 394-406, 2005.

[33] P. Kalagiakos and P. Karampelas, "Cloud Computing learning," *Application of Information and Communication Technologies (AICT)*, 2011 5th International Conference on, pp. 1-4, 2011.

[34] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, Utah, USA*, pp. 147-156, 2014.

[35] H. Karimipour and V. Dinavahi, "Parallel Domain Decomposition Based Distributed State Estimation for Large-scale Power Systems," *IEEE Transactions on Industry Applications*, no. 99, pp. 1-6, 2015.

[36] M. Shahidehpour and Y. Wang, "Communication and control in electric power systems: applications of parallel and distributed processing," 1 ed: John Wiley & Sons, pp. 235-249, 2004.

[37] Y. Erkeng, "Power system state estimation," Beijing: Hydroelectric Power publishing company, 1985.

[38] H. Karimipour and V. Dinavahi, "Parallel domain decomposition based distributed state estimation for large-scale power systems," *IEEE/IAS*

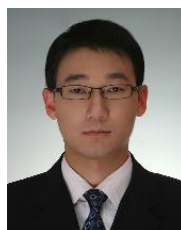
51st Industrial & Commercial Power Systems Technical Conference (I&CPS), 2015.

[39] P. Sanders and C. Schulz, "Think locally, act globally: Highly balanced graph partitioning," *Experimental Algorithms*, 1 ed: Springer, pp. 164-175, 2013.

[40] J. Zawodny, "Redis: Lightweight key/value store that goes the extra mile," *Linux Magazine*, vol. 79, 2009.

[41] R. D. Zimmerman, S. Murillo, x, C. E. nchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Transactions on Power Systems*, vol. 26, no. 1, pp. 12-19, 2011.

[42] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13-16, 2012.



**Hengjing He** received his master's degree from Shanghai Jiao Tong University in China in 2011. He is now a PhD student in Tsinghua University. His research interests include Computerized Measurement and Instrumentation, Cloud based Instrumentation and real-time big on data processing in Measurement systems.



**Wei Zhao** received his Ph.D. from Moscow Energy Institute, Russia, in 1991. He is now a professor of Tsinghua University. His research areas include electromagnetic measurement, virtual instrumentation, networked instrumentation system, cloud based instrumentation.



**Songling Huang** received his Ph.D. from Tsinghua University in 2001. Currently, he is a professor in the Department of Electrical Engineering at Tsinghua University. His current research interests are electromagnetic measurement and nondestructive evaluation.



**Geoffrey C. Fox** received a Ph.D. in Theoretical Physics from Cambridge University in 1967 and is now the Associate Dean for Research and Graduate Studies at the School of Informatics and Computing at Indiana University, Bloomington, and professor of Computer Science, Informatics, and Physics at Indiana University where he is director of the Community Grids Laboratory. His research interests include data science, parallel and distributed computing.



**Qing Wang** received her Ph.D. from the De Montfort University, UK in 2001. She is now a lecturer with the School of Engineering and Computing Sciences at Durham University. Her research interests include electronic instruments and measurement, computer simulation and advanced manufacturing technology. Dr Wang is a Chartered Engineer (CEng), a senior member of IEEE (SMIEEE), a member of IMechE (MIMechE), a member of ASME (MASME) and a Fellow of the Higher Education Academy (FHEA).