

## Data collection in data centers

### FIELD OF THE INVENTION

Embodiments of the present invention relate to the management of data centers. In particular, embodiments of the present invention relate to the collection of running status information for devices in data centers.

### BACKGROUND OF THE INVENTION

Data centers are crucial to support information technology services such as web searching, instant messaging, social networking, online gaming, and online office applications. To support these services, data centers are reported to have emitted as much carbon dioxide as all of Argentina in 2008 and consumed about 1.3 % of world total electricity in 2010. Given the huge capital expenditures and operating expenditures, many technologies have been developed to build a more efficient, cleaner data centers for economic, social, and environmental benefits.

Among the emerging data center technologies, cloud computing and software defined networking (SDN) stand out. Cloud computing provisions computing and storage resources on demand offering a new business model for information technology companies. Conventional efforts often focus on virtualization technologies, including server, storage, and network virtualization, and data center management frameworks, such as Open-Stack and CloudStack. With these virtualization technologies and management tools, virtual machines (VMs) can be created with proper computing, storage, networking configuration. In the paradigm of SDN, network control plane is decoupled from data plane and is directly programmable. Using remote management protocols, for example OpenFlow, network devices can be programmed by controllers. On the controller, various applications can be built for specific purposes, for example, green data center networks, network load balancing, and runtime network configuration for hadoop.

Current approaches in data center operating, however, have two problems. First, while technologies such as cloud computing and SDN provide methods for dynamic management, data center runtime status data, on which dynamic management is based, usually receives little attention. Second, a technology only focuses on a single domain, for example VM management in cloud computing and network management in SDN, may not be able to offer an optimal solution.

Conventional data collection in data centers involves embedded firmware of networking devices such as switches and routers, out-of-band management of servers, environmental sensor management systems, and power management of rack power distribution units (PDUs). These solutions focus on a specific domain of data center, e.g., network, servers, environment, and power. Intergraded monitoring systems offer web-page based visualization of the running status data, but do not offer convenient interface to access the data.

## SUMMARY OF THE INVENTION

According to a first aspect of the present invention there is provided a method of managing a data center. The data center comprises a first data center device; and a second data center device. The method comprises: extracting running status information for the first device in a first data format; converting the running status information for the first device from the first format to a data transfer format; sending the running status information for the first device in the data transfer format to a management server for the data center over a messaging bus; storing the running status information for the first device in a management database on the management server; extracting running status information for the second device in a second data format; converting the running status information for the second device from the second format to a data transfer format; sending the running status information for the second device in the data transfer format to a management server for the data center over the messaging bus; and storing the running status information for the second device in a management database on the management server.

In embodiments of the invention, software sensors are developed for each different category of devices, e.g., servers, switches, power meters, and environmental sensors. Based on the software drivers provided, software sensors read running status data from corresponding devices and export the data in a unified format. A messaging bus and database are used for data transmission and storage respectively. As a result, unified data collection in data centers for subsequent data analytics and system tuning is achieved.

In an embodiment extracting running status information for the first device comprises executing an adapter configured for the first device.

The adapter configured for the first device may have an associated configuration file. Then executing the adapter configured for the first device comprises executing the adapter configured for the first device according to the associated configuration file.

In an embodiment the running status information for the first device and the running status information for the second device are sent at regular time intervals. The timing of the sending of the running status information for the first device and the sending of the running status information for the second device may be set by a signal distribution calculated by the management server.

In an embodiment the data transfer format is a text based data format. The data format may comprise a series of indications of metrics and a series of corresponding values for the metrics.

In an embodiment sending running status information for the first device comprises sending a first set of data at a first time and a second set of data at a later time, and wherein the first set of data comprises indications of metrics and corresponding values, and the second set of data comprises a set of values with the indications of metrics omitted. Thus, key-value splitting to decrease the data size in data transmission may be achieved. The key-value splitting decreases the data size by exploiting the natural structure of the data.

In an embodiment the running status information for the first device comprises dynamic data which changes over time and stable data which indicates the configuration of the first device, and wherein the dynamic data is sent at regular time intervals and the stable data is sent when a change in the configuration of the first device is detected.

Embodiments may further comprise analyzing the running status information for the first device and / or the second device at the management server and sending control signals for the first device and / or the second device over the messaging bus.

According to a second aspect of the present invention there is provided a management system for a data center, the data center comprising a first data center device; and a second data center device, the management system comprising: a first monitoring server; a second monitoring server; a management server and a messaging bus coupling the first monitoring server, the second monitoring server and the management server. The first monitoring server is configured to extract running status information for the first device in a first data format; convert the running status information for the first device from the first format to a data transfer format; and send the running status information for the first device in the data transfer format to the management server for the data center over the messaging bus. The monitoring server device is configured to extract running status information for the second device in a second data format; convert the running status information for the second device from the second format to a data transfer format; and send the running status information for the second device in the data transfer format to a management server for the data center over the messaging bus. The management server is configured to store the running status information for the first device in a management database on the management server; storing the running status information for the second device in the management database on the management server.

According to a third aspect of the present invention there is provided a method on a monitoring server for monitoring a device of a data center. The method comprises converting the running status information for the device from the first format to a data transfer format; and sending the running status information for the device in the data transfer format to a management server for the data center over a messaging bus.

According to a fourth aspect of the present invention there is provided a monitoring server for monitoring a device of a data center. The monitoring server is configured to extract running status information for the device in a first data format; convert the running status information for the device from the first format to a data transfer format; and send the running status information for the device in the data transfer format to a management server for the data center over a messaging bus.

Further aspects of the present invention provide a computer readable carrier medium comprising processor executable instructions which when executed on a processor cause the processor to carry out a method in accordance with the above aspects of the present invention.

## BRIEF DESCRIPTION OF THE DRAWINGS

In the following, embodiments of the present invention will be described as non-limiting examples with reference to the accompanying drawings in which:

Figure 1 schematically shows a data center according to an embodiment of the present invention;

Figure 2 is a flow chart showing a method of data collection in a data center according to an embodiment of the present invention;

Figure 3 shows the processing carried out in an embodiment of the present invention;

Figure 4 shows the processing in carried out in an embodiment of the present invention;

Figure 5 illustrates key-value splitting which may be implemented in embodiments of the present invention;

Figure 6 is a flow chart illustrating signal de-synchronization in an embodiment of the present invention;

Figures 7a and 7b show benchmark results of Ejabberd message transmission performance on an embodiment of the present invention;

Figures 8a and 8b show benchmark results of MongoDB save operation performance on an embodiment of the present invention;

Figure 9a illustrates the application of an embodiment of the present invention to a network topology with two paths;

Figure 9b illustrates the change in traffic size between the two paths under management using an embodiment of the present invention;

Figure 10 shows the migration of a virtual machine from a source to a destination using an embodiment of the present invention; and

Figure 11 shows an implementation of server power management for device safety using an embodiment of the present invention.

## DETAILED DESCRIPTION

A data center is a system consisting of devices with various functions, including PDUs for power supply, air conditioners for cooling, servers and networking devices for ICT service provisioning. Challenges arise to manage and optimize these devices

as a whole system, since these devices provide inherently different methods for monitoring and control.

Figure 1 schematically shows a data center according to an embodiment of the present invention. The data center 100 comprises a cooling system 110; a power supply system 120; and an ICT infrastructure system 130.

The cooling system 110 transfers heat generated by all the data center devices to ensure proper functioning and safety of the data center. As shown in Figure 1, the cooling system 110 comprises a first device 112 and a second device 114. The first device 112 and the second device 114 may be any type of cooling device. Air cooling using devices such as air conditioners and exhaust fans are the most commonly used solutions. As shown in Figure 1, a first monitoring server 116 runs a first software sensor 118 which receives running status information of the first device 112 and the second device 114. This running status information may be for example, environment monitoring information, including temperature and humidity.

The power supply system 120 provides electric power for all the data center devices. As shown in Figure 1, the power supply system 120 comprises a third device 122 and a fourth device 124. The third device 122 and the fourth device 124 may be, for example, power distribution units (PDUs). In this embodiment, we specially focus on power supply in data centers, where PDUs are used as end points for power supply. PDUs are fitted with multiple outlets to accommodate high device density in data centers. Additional features of a PDU often include remote power consumption and environment monitoring via interfaces such as SSH, SNMP, Modbus, and web pages. As shown in Figure 1, a second monitoring server 126 runs a second software sensor 128 which receives running status information of the third device device 122 and the fourth device 124.

The ICT infrastructure system 130 comprises devices such as servers and networking devices (e.g., switches, routers, firewalls, load balancers) and is the part that provides ICT services the data center 100. Servers provide computing and storage capabilities to host various applications, for example, web pages, databases,

and online games. In the paradigm of cloud computing, computing and storage resources of physical servers are virtualized as VM instances and provisioned on demand with elasticity. Networking devices interconnect devices in a data center and provide connectivity outside the data center. Emerging technologies in the networking domain such as software-defined networking (SDN) and network function virtualization (NFV) may be implemented in the ICT infrastructure system 130.

As shown in Figure 1, the ICT infrastructure system 130 comprises a fifth device 132 and a sixth device 134. The fifth device 132 and the sixth device 134 may be servers or network devices as described above. A third software sensor 138 runs on a third monitoring server 136. The third software sensor 138 receives running status information of the fifth device and the sixth device 134.

The first monitoring server 118, the second monitoring server 128 and the third monitoring server 138 may be implemented as computers having at least one processor and storage for a computer program which implements the software sensors when run on the processor.

The software sensors are connected to a messaging bus 140. A management server 150 is connected to the messaging bus. The management server 150 stores a database 152 which stores the running status information collected from the devices of the data center 100.

As discussed above, the devices making up the data center 100 provide inherently different methods for monitoring and control. Table 1 below shows examples of interfaces of data center software and hardware facility.

Table 1: Interfaces of data center software and hardware facility

| System | Subsystem | Tools                    | Interfaces |
|--------|-----------|--------------------------|------------|
| Power  | PDU       | DELL managed<br>Rack PDU | WUI, SNMP  |

|         |                        |                                                                                                               |                                                                                 |
|---------|------------------------|---------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| Cooling | Environment monitoring | SynapSense                                                                                                    | WUI, Modbus                                                                     |
| ICT     | Server                 | Ganglia (Ganglia: <a href="http://ganglia.info/">http://ganglia.info/</a> )<br>iLO from HP<br>iDRAC from DELL | WUI<br><br>WUI<br>WUI, Shell command                                            |
|         | VM                     | OpenStack, CloudStack<br>virsh<br>virt-manager<br>libvirt                                                     | WUI, REST API<br>Shell command<br>Graphic user interface<br>Programming library |
|         | Switch                 | HP Switch firmware<br>OpenFlow controller (Floodlight)<br>ovs-dpctl                                           | WUI, CLI, SNMP<br>WUI, REST API<br><br>Shell command                            |

As shown in Table 1, most devices provide command line interfaces (CLIs) and web user interfaces (WUIs) via computer network. However, application programming interfaces (APIs) are often absent for further development.

This leads to two major issues: Firstly, automated data center management is required for elastic service provisioning with agility. However, WUIs and CLIs are designed for human operations. As a result, human configuration and management are required, which can be cumbersome and error prone. Secondly, the functions and interfaces imposed by each device only focus on its own domain.

Embodiments of the present invention aim to address these issues. We argue that information should be shared and joint operations be applied, to achieve optimized data center operation. Since a unified API may not be present for all the data center devices. We must accept the fact that the interfaces imposed by current devices vary. As a solution, we propose to build adapters for the data center devices to shield differences among the devices and provide abstraction of the devices' functions. We abstract a device as a box with interfaces for data input and output: (i) device

monitoring: the data output interface provides device running status information; and  
(ii) device control: the data input interface takes in control commands for device runtime status control.

In the present disclosure, we focus on monitoring data collection as a proof-of-concept prototype of the unified data center management framework. A data center is naturally large scale and highly dynamic. It brings challenges for data center data collection. We summarize the key features that are demanded for data center data collection as follows:

(i) Real-time. Data center running status data should be collected at a real-time manner to accommodate timely data center management. Due to the high sampling frequency, the overhead of each sampling should be kept low. The data sample interval should be less than or equal to 15 seconds, as often used by existing monitoring systems, for example, ganglia.

(ii) Extensibility. The data collection system should offer a friendly interface for users to add data metrics for newly installed devices and define novel metrics for special purposes.

(iii) Scalability. The scale of data centers is currently large and expected to grow rapidly. Key components of the data collection system should be able to scale-up and provide methods, for example clustering.

To collect data from all the three main subsystems in a data center, i.e., a cooling subsystem, a power subsystem, and a ICT infrastructure subsystems, we present a three-step system, consisting of a data sourcing step, a data transfer step, and a data storage step, an example of the method is described in more detail with reference to figure 2 below. In the data sourcing step, runtime status data are collected from all the data center devices and encoded in an identical format on the monitoring servers. In the data transfer step, the sourced data are transferred for further data usage, for example, data storage and data analysis. In the data storage step, the received data are saved in a persistent data storage base.

To collect monitoring results from and send device tuning commands to all the data center devices via the computer network, one of the main challenges is to accommodate all the different interfaces exposed by the devices and existing device monitoring and management software, as shown in Table 1. To deal with the complexity brought by diversity of interfaces, we design a message bus centric architecture. Our system can be divided into three components, adapters that deal with the difference of individual devices and output formatted data, a message bus that transfers the formatted data, and a database to store the collected data.

This system is designed for data center status information collection, with consideration on scalability, efficiency, extensibility and future integration with data center management. The collected data will be mined to help make system tuning decisions. The channel that is used to transfer collected data will be reused to transfer system tuning commands to individual devices.

To collect data from all the data center devices, one of the main challenges is to accommodate all the different interfaces exposed by existing software and hardware facilities, as shown in Table 1. We propose a new technical framework, embedded software as sensors (ESaS), for real-time monitoring and management in data center operations. In the framework of ESaS, software adapters are embedded in data center facilitates. These adapters work as virtual sensors to monitor ICT activities (e.g., CPU usage, I /O statistics, memory footprint, heap statistics, thread information, etc.), power usage, and environment conditions in real-time. This information is then encoded in a common format for data export, thus offering a unified monitoring data collection interface across all the data center devices.

Besides data automatically collected from individual data center facilitates, the system requires data from user input for full functioning, for example, server hierarchy (i.e., which servers are grouped as a “clusters”); physical locations of the devices (i.e., server room, rack slot); physical cabling (i.e., which PDU outlet is used for a server).

Data collected from multiple sources should be transferred for further usages, for example, data storage in our data collection system or data analysis in stream processing systems. The computer network in a data center can be reused as the data transfer infrastructure. Various network protocols can be used, including low-level transport protocols such as TCP and UDP, and high-level messaging protocols such as XMPP and AMQP. Generally, low-level protocols provide higher performance but bring higher programming complexity than that of high-level protocols.

Data storage stores the data collected from individual data center devices for data persistence, and provides data retrieval interfaces for further data consumption. Both traditional relational database management system (RDBMS), for example MySQL, and emerging NoSQL databases, for example MongoDB, can be considered.

The philosophy behind this design is to separate data storage and data transmission from data sourcing. Data sourcing will deal with the complexity imposed by device diversity. A unified data transfer channel will be used for data transfer and a central data storage will be used for data persistence. As a result, in the data collection workflow for individual devices, only data sourcing varies for different data center devices, data transfer and data storage can be reused. Furthermore, this messaging bus design makes it convenient to implement data center management. The messaging channel could be reused to transfer device tuning commands, and the adapters could implement features to deal with the diversity of device management interfaces.

Figure 2 is a flow chart showing a method of data collection in a data center according to an embodiment of the present invention. The method may be carried out on the data center 100 shown in Figure 1.

In step 202, a software sensor running on one of the monitoring servers extracts running status information for that device. As is described in more detail below with reference to Figure 3 below, the software sensor loads an adapter which extracts the running status information for the device in the data format of the device.

In step 204, the software sensor running on the monitoring server converts the running status information into a data transfer format. As described in more detail below, the data transfer format may be text based format such as XML or JSON.

In step 206, the software sensor running on the monitoring server sends the running status information of the device to the management server 150 over the messaging bus 140.

In step 208, the management server 150 stores the running status information for the device in the database 152.

The method shown in Figure 2 may be repeated for each device and / or monitoring server of the data center 100. It is noted that once the running status information has been converted into the data transfer format in step 204, the later steps will all involve processing data in the data transfer format.

Figure 3 shows the processing carried out in an embodiment of the present invention. As shown in Figure 3, the software sensor 302 dynamically loads adapters 304. The adapters 304 monitor the running of the data center on which they are executed and generate running status information for the device. The adapters 304 are implemented using the data exporting interface offered by individual type of devices, e.g., IMPI for servers, LIBVIRT for virtual machines, SNMP for switches and environmental sensor, and NVML for GPUs. The software sensor 302 converts the running status information to the data transfer format and periodically transmits the running status information over the message bus 308. After conversion, the running status information is formatted in a unified form (i.e., JSON) with a {time: value} pair and multiple {key: value} pairs, which represent multiple metrics export at a time, as described in more detail with reference to Figure 5 below. A standardised message bus 308 (i.e., XMPP) is used for transmission of exported data and can be reused to transfer system tuning commands.

The management server 150 acts as a data sink 310 which can receive data from multiple software sensors and save the data 312 in a centralized database (i.e., MongoDB), which provides convenient interfaces to read the data for visualization or analytics.

An example implementation of an embodiment of the present invention will now be described. In this embodiment, as a prototype system, data center data collection for servers, VMs, switches, and PDUs is implemented in a test bed. The test bed comprises of servers from DELL (Intel Xeon E5 2620 CPU, 64G DDR3 memory, 4 Gigabit Ethernet interfaces, 1 TB SCSI disk, iDRAC 7), switches from HP (24 or 48 Gigabit Ethernet ports) and PDUs from DELL. Linux is increasingly popular as the operation system (OS) for servers in data center, and is our primary supported platform. Particularly, we have tested on both Ubuntu and CentOS. For performance considerations, C/C++ is chosen as the programming language.

As discussed above, various suitable options exist for data transfer and data storage. In this embodiment, we choose XMPP for data transfer and MongoDB for data storage. The reasons for this choice are as follows:

(i) XMPP and MongoDB provide simpler and flexible implementations compared with their alternatives, for example, TCP/UDP and MySQL. With XMPP, you do not need to manage the socket connections and parse the received binary data as in the case of TCP and UDP; MongoDB stores data in a schema-less manner, allowing dynamic data structure changes, for example, adding or removing a data field.

(ii) Both XMPP and MongoDB servers can scale up for higher performance with advanced configurations, for example, clustering. Clustering of XMPP and MongoDB servers also helps avoid single points of failure, improving reliability and fault tolerance.

In addition to the features discussed above, the following techniques may also be implemented in embodiments to further improve the performance, flexibility, and

scalability of the system, including configuration file support, dynamical module loading, and data compression.

Adapters collect data from individual devices and format the data for data transfer. The following libraries may be used for data sourcing from servers, VMs, switches, and PDUs:

(i) Net-SNMP. Simple Network Management Protocol (SNMP) is a widely used protocol for monitoring the status of network devices, servers and even devices like PDUs and sensors. Net-SNMP is a suite of useful applications and programming libraries. In our prototype, SNMP can be used to collect running status data from switches, PDUs, and servers. Data collected via SNMP are encoded as {object identifier (OID): value} pairs, and the meanings of the OIDs can be found in the management information base (MIB) of the corresponding device.

(ii) Libvirt. Libvirt is a toolkit to interact with all the popular virtualization technologies, including KVM/QEMU, Xen, LXC, OpenVZ, VirtualBox, VMware ESX and GSX, Microsoft Hyper-V. Only the APIs to get host and VM status information are currently used in our prototype, further we can implement functions such as VM creation, clone and migration via libvirt. VM status information can be retrieved without intrusion into VMs with libvirt. This is useful in a virtualized environment where VMs are rented by various tenants who may not allow data collection program running on their VMs.

(iii) Libgtop. Libgtop is a library to get system-specific data such as CPU, memory, disk, and network usage. Linux has a virtual filesystem, /proc, that contains runtime system information. There is a need to parse the files in /proc to get the value of a specific metric and libgtop can save this trouble.

(iv) Iptables. Iptables is a firewall, it can be used to set-up, maintain, and inspect the tables of packet filter rules in the Linux kernel. Since packet count and traffic size information are also recorded in the packet filter rules, iptables can also be used to

get traffic size of per-socket-pair. OpenFlow provides similar information; however, switches or routers usually provide only traffic size information per-port.

(v) NVML. NVML is a C-based API for monitoring and managing various states of the NVIDIAGPU devices. It provides a direct access to GPU status data, including clock frequency, fan speed, memory usage, power, and temperature.

Though data collected using the adapters discussed above may have different meanings and data types, we abstract the collected data for every metric as a tuple:

$$\{time, metric, value\}.$$

In the above tuple, *time* indicates the time when the data are collected; *metric* represents the meaning of the data, whether it is CPU usage, environment temperature, power consumption, or other; *value* records the value for metric at time, it can be a number or a string. A record for data collected at a time usually consists of a *time* field and multiple *{metric: value}* pairs.

The data format to encode a record can be roughly classified into octet-based formats and text-based formats. Octet-based formats require definition of which octets represent the value for which metric and the data type for this value. On the other hand, text-based formats, for example XML<sup>11</sup> and JSON<sup>12</sup> encode all the metrics and values in as a text.

For the data transfer format, we choose a text-based format, especially JSON, as the data format for data transfer (implemented using boost property tree), since it allows more flexible data structure and simpler implementation than octet-based formats. A JSON encoded record can also be inserted in MongoDB directly. The following is a sample JSON-encoded message:

```
"computeNodeStats":{
  "time": "1374582648",
  "hostName": "cloudagent",
  "hostCPUStats":{
    "kernel": "1700202850000000",
```

```

        "user": "8764609800000000",
        "idle": "1934279140000000",
        "iowait": "16153300000000"
    }
}.

```

As shown in the sample message above, nesting of metric fields is allowed. This can help represent the hierarchy of the metrics.

Table 2 below shows examples of metrics which may be supported in embodiments of the present invention.

| Device         | Library applied | Metrics                                                                                                                                                                           |
|----------------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PDU            | Net-SNMP        | Power and energy usage of each outlet                                                                                                                                             |
| Server (iDRAC) | Net-SNMP        | Fan speed, power consumption<br>Temperature (CPU, inlet, and exhaust)                                                                                                             |
| Switch         | Net-SNMP        | Input and output bytes of each port                                                                                                                                               |
| Server         | Libgtop         | CPU (load, system time, user time, etc.)<br>Memory (total size, free size)<br>Network (input and output bytes, etc.)<br>Disk (total and free sizes, Read and write bytes, etc.)   |
| VM             | Libvirt         | CPU (system time, user time, etc.)<br>Memory (balloon and resident set size)<br>Network (input and output bytes, etc.)<br>Disk (total and free sizes, Read and write bytes, etc.) |
| Server         | iptables        | Input and output size for each rule                                                                                                                                               |
| GPU            | NVML            | Clock frequency, fan speed, memory usage, Power, temperature, etc.                                                                                                                |

A message bus can be used to exchange information among independent components in a system. Particularly in our data center data collection system, the message bus is mainly used to transfer data collected for data storage. We choose XMPP as the protocol to implement the message bus. The XMPP-based message bus consists of two main components:

(i) a messaging server, ejabberd (version: 2.1.10–2ubuntu1.1) is chosen as the XMPP server in our prototype, and alternative options include openfire. Ejabberd is a XMPP instant messaging server, written in Erlang/OTP. Ejabberd supports clustering, nodes in an ejabberd cluster can be added or removed on the fly.

(ii) messaging clients, that send and receive messages. Various C/C++ XMPP libraries exist, for example, gloox and swiften. We have used swiften in our prototype implementation.

We choose MongoDB(version: mongodb-10gen 2.4.6) for data persistence. MongoDB is an open-source document-based NoSQL database, which is written in C++. It stores data in JSON-style documents with dynamic schemas for simplicity and flexibility. MongoDB organizes data in the following structure,

*{database, collection, document}.*

*document* is the basic unit of data in MongoDB. The MongoDB *document* format is known as BSON, consisting of multiple *{field : value}* pairs, which is very similar to JSON. A *collection* is a group of MongoDB *documents*, and a *database* is a physical container for *collections*. We designed a simple data structure to store collected data in MongoDB, as follows:

(i) A database for each data category, for example, server status and switch status.

(ii) A collection for each device identified by the host name of a device. The host name is chosen and ensured unique manually.

(iii) A document for data collected each time. It is a record encoded in BSON, consisting a *{time : value}* pair and multiple *{metric : value}* pairs.

Embodiments may be designed with extensibility to satisfy complex and changing data collection requirements, and performance consideration to accommodate real-time data collection in a large-scale data center.

A critical demand on data center data collection is to add support for new data metrics. New devices installed will demand monitoring of these devices. Specific metrics, which have not been paid attention to, may attract interests later. For example, a particular system may require monitoring the status of each CPU core; while previously, only overall status data of a CPU are collected.

Figure 4 shows the processing in an embodiment of the present invention. The embodiment shown in Figure 4 allows dynamical loading of adapters which can also be configured via a configuration file.

As shown in Figure 4, a software sensor implemented as an agent 410 runs four adapters 420, 430, 440 & 450. Each adapter corresponds to a device category. Each adapter has an associated configuration file 422, 432, 442 & 452. A timer 460 triggers data collection at a certain time interval. When the timer 460 triggers the data collection, each adapter collects monitoring data according to its configuration file. The monitoring data indicates the running status information for the device monitored by the corresponding adapter. At each time interval, the monitoring data from each of the adapters is converted into the data transmission format and sent out by a send message module 470 of the agent 410. The data is received by the data sink 480 which is implemented in the management server, which receives and stores the data 482.

As described above, in this embodiment, each adapter has a configuration file to define the behavior of the adapter. Typically, for the SNMP adapter, we can specify the OIDs of data to be collected in its configuration file.

In some embodiments, dynamical module loading is possible. This is achieved by the data sourcing program of the software sensor also having a configuration file. This configuration file specifies which adapters to be loaded and some parameters, for example the data collection interval. An advantage of dynamical module loading is that a new adapter can be loaded on the fly, without recompiling the whole data sourcing program.

Figure 5 illustrates key-value splitting which may be implemented in embodiments of the present invention. Key-value splitting may be implemented to decrease the data size in data transmission. As shown in Figure 5, the original data 510 at each time interval comprises information 512, 514 which indicates the metric. For the first field this metric is time 512 and for the second and subsequent fields, this metric is key 514. For each metric, the original data also includes a value field 516, 518 which indicates the value for the metric.

When key-value splitting is implemented, the first set of data transmitted 520 resembles the original data 510. That is to say that the metric and the value is transmitted for each field. However, for subsequently transmitted data 530, the metric field is omitted and only the value data is transmitted.

As described above, key-value splitting is used to decrease the data size, exploiting the natural structure of the data, rather than conventional data compression algorithms (e.g., GZIP and DEFLATE). Since the data with the same structure will be transmitted periodically, the original data can be reconstructed using the entire data transmitted first and subsequent {value} fields.

In some embodiments a conventional data compression algorithm is used either as an alternative to key-values splitting or in addition to key value splitting.

In some embodiments, signal de-synchronization is carried out when multiple software sensors are sending data to a single data sink.

Figure 6 is a flow chart illustrating signal de-synchronization in an embodiment of the present invention. In step 602, system time is synchronized across the software sensors running on the devices in the data center. In step 604, each software sensor notifies the data sampling period of the software sensor to the management server. In step 606, the management server calculates the signal distribution in timeline to maximize the minimal interval of two adjacent data received by the data sink. The distribution is recalculated when a software sensor joins or quits.

In step 608, the management server sends a message to the software sensor to shift the data transmission time along the timeline based on the calculated signal distribution.

Sending data at a time will stress the data sink and cause spikes in background traffic. The method described above with reference to Figure 6 avoids such spikes.

In the following, the performance of implementations of embodiments of the present is described. Data transfer performance of embodiments of the present invention rely heavily on the capacity of the ejabberd XMPP server. A simple benchmark of ejabberd shows that it is suitable for a data center with thousands of servers. Also, data compression may be implemented data to further improve the performance of data transfer.

In the ejabberd performance benchmarking, a sender will continuously collect data, format the data as JSON messages, and then send messages to a sink. With a typical configuration of the data sourcing program and the adapters, the message size is 2314 bytes. If the message sending speed exceeds the capacity of the ejabberd XMPP server, messages will be delayed. As a result, latency received message can be used to indicate whether the message speed has reached the capacity of the ejabberd XMPP server.

Figures 7a and 7b show benchmark results of Ejabberd message transmission performance.

Figure 7a shows latency change and message sequence number for different numbers of messages sent per second (MSPS). As shown in Figure 7a, when there is no sleep time among two sequential messages sent out, the latency can be huge and will increase (the line with circles in Fig. 7a). In the benchmark, we vary the sleep time (i.e., 0, 100, 300, 400, 500, 1,000  $\mu$ s) among two sequential messages, to vary the messages sent per second (MSPS), and record the latency of each message. As shown in Figure 7a, when message sending rate is about 12,865.0 MSPS (no sleep time), the latency can be as large as thousands of milliseconds and may increase overtime. When the message sending rate decreases, the latency decreases and grows slower, and finally be stable as several milliseconds, as in the case for 1,432.8 MSPS and 890.0 MSPS (500 and 1,000  $\mu$ s sleep time).

Figure 7b shows the average latency for each message sending rate. Figure 7b shows the average latency for all the messages sent at a rate and the turning point is (1,695.9, 2.6). As a result, we estimate the capacity of the ejabberd XMPP server as about 1,700 messages per seconds. Assuming for each device monitoring data are collected once every second, the ejabberd XMPP server can handle over 1,000 devices, regardless of messages sending distribution over time and online user numbers for the ejabberd XMPP server.

In this benchmarking, we just deploy ejabberd in a single-node mode. The traffic shaper module is disabled by setting bandwidth limits of traffic shapers to extremely large numbers. With advanced technologies, for example clustering, the performance of the ejabberd XMPP server can be further improved.

Data compression is a common method to decrease data size and thus lower the burden on data transfer. Two data compression strategies are considered in our implementation, conventional compression algorithms and a special data compression algorithm designed for JSON based on key value splitting as discussed above.

We use Gzip (implemented using crypto++ library) as a representative of commonly used conventional lossless data compression algorithms, alternative options include

Zlib. The compressed data of Gzip use binary coding. To be transferred via XMPP, it needs to be encoded as text using a binary-to-text encoding scheme, for example, Base64.

As described above, messages are formatted to JSON strings, which consist of multiple {metric : value} pairs. Since all the messages for data collected from a device at different times have similar structure with almost the same metric fields, and may just vary in some value fields. This feature can be utilized in JSON message compression. Common practices of JSON compression consist of two main methods. These are as follows:

- (i) separate *value* fields from *metric* fields, and just the first message will contain the *metric* fields; and
- (ii) build a map for repeated *value* fields, so that a long *value* string can be represented by an index of the map.

In an embodiment, collected data is classified into two categories, relatively stable data such as CPU model and number of CPU cores, and dynamical data such as CPU usage and CPU temperature. Stable data will only be transferred once if no changes are detected. For the dynamic data only separation of values and keys is used, since dynamic data occupy most of the transferred data size and do not have many repeated value fields.

As a result, only the first message will contain metric fields, subsequent messages with the same metric fields will only contain value fields. The sample JSON message is encoded as a “frame” consisting of the metric fields shown as follows:

```
"computeNodeStats":{
  "time": " ",
  "hostName": " ",
  "hostCPUStats":{
    "kernel": " ",
    "user": " ",
    "idle": " ",
    "iowait": " "
  }
}
```

}.

and “fillers” consisting of value fields shown as follows:

```
"1374582648", "cloudagent", "1700202850000000",  
"8764609800000000", "1934279140000000", "16153300000000".
```

Comparison of data compression algorithms is shown in Table 3 below.

|            | JSON   | XML    | Split | Gzip  | Gzip-Base64 |
|------------|--------|--------|-------|-------|-------------|
| SNMP       | 1734   | 4963   | 148   | 261   | 353         |
| Host       | 2023   | 2616   | 729   | 611   | 828         |
| VM         | 268    | 366    | 115   | 190   | 260         |
| IPTables   | 383    | 474    | 111   | 165   | 224         |
| Comparison | 100.00 | 190.99 | 25.02 | 27.84 | 37.77       |

The original JSON message sizes for data collected from different sources are recorded. Message size using different data compression methods is compared. Only the size of the “fillers” is counted, since the “frame” will only be transferred once. We can see that separating metric fields and value fields (marked as “Split”) can achieve better data comparison ratio than first Gzip compression then Base64 encoding, the average size of the “fillers” is 25.02 % of the original JSON message. Also, we can see that a message encoded in XML is about 2 times the size of the original message encoded in JSON.

Figures 8a and 8b show benchmark results of MongoDB save operation performance. We benchmarked MongoDB save operation performance. The data save operation is the main database access operation in a data center data collection system. Data is inserted in the MongoDB in different cases trying to increase the actually total save operators per second (SOPS) to the capacity of MongoDB. The typical size of each inserted data is about 2,877 bytes.

Figure 8a shows time average save operations per second (SOPS): the average SOPS from the start time to the current time for different data insert processes. Figure 8b shows all time average SOPS: the average SOPS from the start time to the end time for the cases A to F in figure 8a. The error bars show the standard deviations.

As shown in Figures 8a and 8b, multiple processes on a single server are used to collect data then insert in MongoDB at first. However, SOPS will not increase when the thread number is increased from 10 to 20 on a single host. When data are inserted on 2 servers, each with 10 processes, the overall data insert speed increases (case D). This indicates that data collection on a single server is the bottleneck. When saving sample data without data sourcing, the performance increases greatly (case E and F). And the all-time average of save operation per second is above 9,000.

In the MongoDB benchmarking, we use a single-node mode MongoDB server. MongoDB also supports advanced techniques such as clustering and sharding, which can further improve its performance.

The data generated and stored in the embodiments described above may be used in many different ways. The sourced data can be transferred to stream processing systems for online analysis to accommodate real-time data center management. Or the historical data stored in the database can be exported for offline analysis to gain insight into the data.

Examples of possible applications include traffic load balancing using OpenFlow, workload balancing using VM migration, and power consumption optimization using iDRAC. All these applications use collected data before it is saved in MongoDB. The next description below will present implementations and preliminary results of these applications.

With the promise of easier network management through programmatic interfaces, OpenFlow/SDN is a rising star in the networking domain. Combining network

bandwidth usage data and programmatic control of OpenFlow, traffic flow migration can be implemented to migrate traffic flow from heavy-loaded links to light-loaded ones.

Figure 9a illustrates the application of an embodiment of the present invention to a network topology with two paths. Figure 9b illustrates the change in traffic size between the two paths under management using an embodiment of the present invention.

As shown in Figure 9a, a network 900 comprises a source server 910 and a destination server 920. The network comprises three switches 930. There are two paths between the source server 910 and the destination server 920 through the network 100: a short path 940 and a long path 950. A data center management system 960 as described above is configured to monitor the traffic flow in the switches 930 and to control the switches 930 using the monitored data. As shown in Figure 9a, the data center management system 960 receives traffic flow data 962 from the switches 930 and controls the switches with traffic flow control signals 964. The embodiment shown in Figure 9a may also include monitoring servers as described above with reference to Figure 1.

The data center management system 960 may control the traffic flow as follows. If the traffic size difference exceeds a predefined threshold, for example 1 GB, OpenFlow rules for the current path will be cleared and new OpenFlow rules will be inserted for the other path.

Figure 9b shows the change of traffic size difference between the long path 950 and the short path 940, when a 4 GB large file is transferred from the source server 910 to the destination server 920 over the network 900. The traffic is balanced over the two paths without breaking the connection for data transfer.

Another application of embodiments of the present invention is virtual machine (VM) migration. VMs are heavily used in cloud environments, giving the advantages of fine-grained computing and storage resource virtualization. VM migration is a

common operation to relocate a VM without discontinuing provided services. It can be used to consolidate VMs in a smaller amount of servers to save energy, or accommodate traffic patterns of hosted services for performance optimization.

We implemented a VM migration application to migrate a VM when the host's free memory size is low, leveraging host memory usage data collected via our data collection system and VM management API provided, libvirt in our case. The VM is based on kernel-based virtual machine (KVM) and is configured with 30 GB memory and an exploit program will occupy its memory quickly.

Figure 10 shows the migration of a virtual machine from a source to a destination. As shown in Figure 10, the VM will be migrated (live migration is used) to another host when the current host's free memory size is less than 40 GB. After a migration time of about 39.2 s (can vary for VMs with different memory and disk sizes), the memory occupied by the VM on the source server is released, and free memory size increases to its original level.

A further application of embodiments of the present invention is server power management. A common practice to save energy in data centers is to power-off devices that are not actually used or set them to idle mode. For servers particularly, Out-of-band management solutions such as iLO and iDRAC provide interfaces for server power management operators including power-off and power-on servers. The server power-off function can be used to power-off server with high temperature to ensure safety, or server in idle mode to save energy. The server power-off function can be used to power-off server with high temperature to ensure safety, or server in idle mode to save energy. The server power-up function can be used to power on these servers to resume or scale-up services.

Figure 11 shows an implementation of server power management for device safety. As shown in Figure 11, the server is caused to power off when CPU temperature is high and power on after a certain time for cooling. We implement a power management application using the iDRAC and CPU temperature data collected via our data collection system. An exploit program will cause the CPU usage of a server

increasing to nearly 100 %, heating its CPU. As illustrated in Figure 11, when the server's CPU temperature is over a threshold, 56 °C (for example), it will be powered off for cooling. Then after a time (for example 30 s), the server is powered on and its power consumption increases.

Those of skill in the art will appreciate that applications of embodiments of the present invention are not limited to those described above. As described above, combining the data collected from all the data center devices and management interfaces provided by these devices, a data center can be managed as a whole system for performance optimization. Individual sample combines data sources and control in different domains, for example, network monitoring using SNMP and network control using OpenFlow, server monitoring and VM management, server temperature monitoring and server power management. Each of the applications demonstrates the feasibility of aspects that can be combined in higher level applications. For example, VM migration and server power management can be combined to save energy by consolidating VMs and powering off unneeded servers.

Whilst the foregoing description has described exemplary embodiments, it will be understood by those skilled in the art that many variations of the embodiments can be made within the scope of the present invention as defined by the appended claims.

## CLAIMS

1. A method of managing a data center, the data center comprising a first data center device; and a second data center device, the method comprising:
  - extracting running status information for the first device in a first data format;
  - converting the running status information for the first device from the first format to a data transfer format;
  - sending the running status information for the first device in the data transfer format to a management server for the data center over a messaging bus;
  - storing the running status information for the first device in a management database on the management server;
  - extracting running status information for the second device in a second data format;
  - converting the running status information for the second device from the second format to a data transfer format;
  - sending the running status information for the second device in the data transfer format to a management server for the data center over the messaging bus;
  - and
  - storing the running status information for the second device in a management database on the management server.
2. A method according to claim 1, wherein extracting running status information for the first device comprises executing an adapter configured for the first device.
3. A method according to claim 2, wherein the adapter configured for the first device has an associated configuration file and executing the adapter configured for the first device comprises executing the adapter configured for the first device according to the associated configuration file.
4. A method according to any preceding claim wherein the running status information for the first device and the running status information for the second device are sent at regular time intervals.

5. A method according to claim 4, wherein timing of the sending of the running status information for the first device and the sending of the running status information for the second device is set by a signal distribution calculated by the management server.
6. A method according to any preceding claim, wherein the data transfer format is a text based data format.
7. A method according to any preceding claim wherein the data format comprises a series of indications of metrics and a series of corresponding values for the metrics.
8. A method according to claim 7, wherein sending running status information for the first device comprises sending a first set of data at a first time and a second set of data at a later time, and wherein the first set of data comprises indications of metrics and corresponding values, and the second set of data comprises a set of values with the indications of metrics omitted.
9. A method according to any preceding claim, wherein the running status information for the first device comprises dynamic data which changes over time and stable data which indicates the configuration of the first device, and wherein the dynamic data is sent at regular time intervals and the stable data is sent when a change in the configuration of the first device is detected.
10. A method according to any preceding claim, further comprising analyzing the running status information for the first device and / or the second device at the management server and sending control signals for the first device and / or the second device over the messaging bus.
11. A management system for a data center, the data center comprising a first data center device; and a second data center device, the management system comprising: a first monitoring server; a second monitoring server; a management

server and a messaging bus coupling the first monitoring server, the second monitoring server and the management server,

the monitoring server being configured to extract running status information for the first device in a first data format; convert the running status information for the first device from the first format to a data transfer format; and send the running status information for the first device in the data transfer format to the management server for the data center over the messaging bus;

the second monitoring server being configured to extract running status information for the second device in a second data format; convert the running status information for the second device from the second format to a data transfer format; and send the running status information for the second device in the data transfer format to a management server for the data center over the messaging bus;

the management server being configured to store the running status information for the first device in a management database on the management server; storing the running status information for the second device in the management database on the management server.

12. A data center according to claim 11, wherein the first monitoring server is configured to execute an adapter configured for the first device to extract the running status information for the first device

13. A data center according to claim 12, wherein the adapter configured for the first device has an associated configuration file and executing the adapter configured for the first device comprises executing the adapter configured for the first device according to the associated configuration file.

14. A data center according to any one of claims 11 to 13 wherein the running status information for the first device and the running status information for the second device are sent at regular time intervals.

15. A data center according to claim 14, wherein timing of the sending of the running status information for the first device and the sending of the running status

information for the second device is set by a signal distribution calculated by the management server.

16. A data center according to any one of claims 11 to 15 wherein the data transfer format is a text based data format.

17. A data center according to any one of claims 11 to 16 wherein the data format comprises a series of indications of metrics and a series of corresponding values for the metrics.

18. A data center according to any one of claims 11 to 17, the first monitoring server is configured to send a first set of data at a first time and a second set of data at a later time, and wherein the first set of data comprises indications of metrics and corresponding values, and the second set of data comprises a set of values with the indications of metrics omitted.

19. A data center according to any one of claims 11 to 18, wherein the running status information for the first device comprises dynamic data which changes over time and stable data which indicates the configuration of the first device, and wherein the dynamic data is sent at regular time intervals and the stable data is sent when a change in the configuration of the first device is detected.

20. A data center according to any one of claims 11 to 19, wherein the management server is further configured to analyze the running status information for the first device and / or the second device at the management server and to send control signals from the management server to the monitoring server and / or the second monitoring server over the messaging bus.

21. A computer readable carrier medium comprising processor executable instructions which when executed on a processor cause the processor to carry out a method according to any one of claims 1 to 10.

22. A method on a monitoring server for monitoring a device of a data center, the method comprising:

extracting running status information for the device in a first data format;

converting, the running status information for the device from the first format to a data transfer format; and

sending the running status information for the device in the data transfer format to a management server for the data center over a messaging bus.

23. A method according to claim 22, wherein extracting running status information for the first device comprises executing an adapter configured for the device.

24. A method according to claim 23, wherein the adapter configured for the device has an associated configuration file and executing the adapter configured for the device comprises executing the adapter configured for the device according to the associated configuration file.

25. A method according to any one of claims 22 to 24 wherein the running status information for the device is sent at regular time intervals.

26. A method according to claim 25, wherein timing of the sending of the running status information for the device is set by a signal distribution received from the management server.

27. A method according to any one of claims 22 to 26, wherein the data transfer format is a text based data format.

28. A method according to any one of claims 22 to 27, wherein the data format comprises a series of indications of metrics and a series of corresponding values for the metrics.

29. A method according to claim 28, wherein sending running status information for the device comprises sending a first set of data at a first time and a second set of data at a later time, and wherein the first set of data comprises indications of metrics

and corresponding values, and the second set of data comprises a set of values with the indications of metrics omitted.

30. A method according to any one of claims 22 to 29, wherein the running status information for the device comprises dynamic data which changes over time and stable data which indicates the configuration of the device, and wherein the dynamic data is sent at regular time intervals and the stable data is sent when a change in the configuration of the device is detected.

31. A monitoring server for monitoring a device of a data center, the monitoring server being configured to extract running status information for the device in a first data format; convert the running status information for the device from the first format to a data transfer format; and send the running status information for the device in the data transfer format to a management server for the data center over a messaging bus.

32. A monitoring server according to claim 31, being further configured to execute an adapter configured for the device to extract the running status information.

33. A monitoring server according to claim 32, wherein the adapter configured has an associated configuration file and executing the adapter comprises executing the adapter according to the associated configuration file.

34. A monitoring server according to any one of claims 31 to 33, wherein the running status information for the device is sent at regular time intervals.

35. A monitoring server according to claim 34, wherein timing of the sending of the running status information for the device is set by a signal distribution received from the management server.

36. A monitoring server according to any one of claims 31 to 35, wherein the data transfer format is a text based data format.

37. A monitoring server according to any one of claims 31 to 36, wherein the data format comprises a series of indications of metrics and a series of corresponding values for the metrics.

38. A monitoring server according to any one of claims 31 to 37, being configured to send a first set of data at a first time and a second set of data at a later time, and wherein the first set of data comprises indications of metrics and corresponding values, and the second set of data comprises a set of values with the indications of metrics omitted.

39. A monitoring server according to any one of claims 31 to 38, wherein the running status information for the device comprises dynamic data which changes over time and stable data which indicates the configuration of the device, and wherein the dynamic data is sent at regular time intervals and the stable data is sent when a change in the configuration of the device is detected.

40. A computer readable carrier medium comprising processor executable instructions which when executed on a processor cause the processor to carry out a method according to any one of claims 22 to 30.