Proceedings of the ASME 2017 12th International Manufacturing Science and Engineering Conference
MSEC2017
June 4-8, 2017, Los Angeles, CA, USA

MSEC2017-3012

RAPIDLY DEPLOYABLE MTCONNECT-BASED MACHINE TOOL MONITORING SYSTEMS

Roby Lynn
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

Wafa Louhichi
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

Mahmoud Parto
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

Ethan Wescoat
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

Thomas Kurfess
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA, USA

ABSTRACT

The amount of data that can be gathered from a machining process is often misunderstood, and even if these data are collected, they are frequently underutilized. Intelligent uses of data collected from a manufacturing operation can lead to increased productivity and lower costs. While some large-scale manufacturers have developed custom solutions for data collection from their machine tools, small- and medium-size enterprises need efficient and easily deployable methods for data collection and analysis. This paper presents three broad solutions to data collection from machine tools, all of which rely on the open-source and royalty-free MTConnect protocol: the first is a machine monitoring dashboard based on Microsoft Excel; the second is an open source solution using Python and MTConnect; and the third is a cloud-based system using Google Sheets. Time studies are performed on these systems to determine their capability to gather near real-time data from a machining process.

INTRODUCTION

The astonishing production of data has revolutionized many fields such as media, finance, healthcare and manufacturing. Experts now point out to a 4300% increase in annual data generation by 2020 [1]. Manufacturing produces more data than any other field, and thus the shifting to a smart and networked factory is essential to realizing higher productivity and efficiency [2]. Advances in both data collection standards and the networks to support those standards has enabled the optimization of production, equipment utilization and product quality while simultaneously decreasing energy consumption.

Machining is a decades-old manufacturing process that is in wide use because it is rapidly implementable, flexible and capable of producing high-quality parts from a wide range of materials. Historically, machining was performed on manual equipment that required an operator to turn handwheels to control the motion of a cutting tool. Today, the vast majority of production machining is done with computer numerical control (CNC) machine tools that use servomotors to create tool motion. As CNC machine tools rely on sensor feedback to control the cutting process, the output from these sensors can be harnessed to perform analysis of the machining process. These sensor data can be used to determine the production rate of a given machine, its current status and in the identification of potential problems with the machine itself. However, the collection of sensor data from the machine tool may present problems if a manufacturer is attempting to integrate data from many different types of equipment. Recently, machine tool builders have been embracing MTConnect as the method of choice for transmitting data from different types of CNC machines. MTConnect is an open-source, royalty-free communication protocol that allows for collection of sensor data from manufacturing equipment through a network connection. While MTConnect is gaining traction in some facilities, there is a lack of cost-effective and easily deployable tools to make use of the data collected from MTConnect.

Effective use of data generated by a manufacturing operation necessitates deployment of a system that can collect, store and utilize data from manufacturing equipment. These three steps, while dependent on one another, each require separate tools to implement. In the data collection stage, acquisition systems must be deployed to measure output from sensors on manufacturing equipment. Using MTConnect, standalone data acquisition systems are no longer required; rather, data collection can be performed by a computer that is on the same network as the machine tool to be monitored. Storage of the collected data can be performed by a database that is also networked with the data acquisition computer. Data storage allows for manufacturers to study historical trends, make predictions and analyze problems. Finally, visualization of both current and historical measurements is necessary to allow manufacturers to make use of the data.

Many large-scale manufacturers have developed proprietary methods for data collection storage from their manufacturing equipment, but these systems are highly-customized and purpose-built for a particular manufacturing operation. Many smaller producers do not have access to a cost-effective method of data collection from their manufacturing equipment, and thus much of the data generated in their production operations goes unused [3]. Small- and medium-size producers need lightweight and efficient ways to both collect and analyze the data generated by their processes. The recent trend towards free or open-source software allows for the development of a system that does not rely on proprietary and expensive commercial software. Additionally, open-source software can be freely modified by the user, which allows for customizability. This paper describes the development and implementation of multiple rapidly-deployable machine monitoring solutions that enable online near real-time data feedback and visualization.

BACKGROUND

The MTConnect Standard

The MTConnect Standard is an open-source and royalty-free protocol for data transmission from manufacturing equipment. The standard is a read only, XML format which allows communication from machine to machine and machine to operator. The standard uses Hyper Text Transport Protocol (HTTP) returning data in a web browser using commands such as /current and /sample [4]. These commands ask a MTConnect-compatible device to return a webpage that contains data collected by the sensors in the device. For example, a current command would cause the device to return the up-to-date readings of all data items it has available. An example of the XML data output from the machine tool when given an HTTP request is shown in Figure 1. The MTConnect agent is responsible for collecting data from the machine and creating the webpage when a request is received. Each time a current or sample command is issued to the machine, the MTConnect agent creates a header for the XML file. The first line is the timestamp of the data, the second line displays the agent sending data to the browser, and the next line says which version of MTConnect is used. The bufferSize is the number of

```
<Samples>
  <Load dataItemId="cl" timestamp="2017-01-
31T22:16:39.623784Z" name="Cload"
sequence="1510344">0</Load>
  <Angle dataItemId="cpos" timestamp="2017-01-
31T22:14:40.170085Z" name="Cdeg" sequence="1509772"
subType="ACTUAL">0</Angle>
  <RotaryVelocity dataItemId="cs" timestamp="2017-01
31T22:16:08.982719Z" name="Cfrt" sequence="1510223"
subType="ACTUAL">0</RotaryVelocity>
</Samples>
```

Figure 1. MTConnect Output using the Current Command

data items that are stored in the agent's circular buffer. Data in the buffer are organized by sequence number, indicating a datum's place. The first sequence number is the first instance of data and the last sequence number is the last instance of data in the buffer. If the sample command is used, data is returned between those sequence numbers. The next sequence number is the beginning of the next available sequence of data.

MTConnect provides access to a wide range of variables including load, program name and the emergency stop status. Each of these data items are organized according to the MTConnect standard. In the standard, machines and other factory equipment are defined as Devices. In the MTConnect output, the device acts as a title before displaying data; in Figure 1, the device is a Mazak 5-axis machining center. The universally unique identifier (UUID) corresponds with one machine and would also appear in the header [5]. The next grouping of terms are the components of the Device, which are predefined in the MTConnect schema and are grouped based on attributes of the machine [6]. Some components on the machine are defined as the controller and the linear and rotary axes. In Figure 1, the component displayed is rotary axis C, corresponding to the C-axis of this particular 5-axis milling machine. All of the data items corresponding to that axis are displayed underneath. Data items can be sorted into three different categories: Sample, Event and Condition [7]. Sample refers to values sent to the agent and then read at a specific time. A sample data item could refer to the load or the position on a certain axis. Event refers to the state that the machine is sending to the agent. Finally, condition data items refer to the machine's ability to function, such as emergency stop status.

MTConnect follows a protocol of processing data sent to the agent from the machine controller. The machine controller, using what is called the adapter, sends the MTConnect data to the agent using transmission control protocol (TCP) [8]. The adapter collects data from the machine controller and its associated sensors and packaging it into a format that is readable by the agent. The adapter can be software implemented on a modern CNC, or it can be a hardware adapter suitable for use on older machine tools. The signal flow of MTConnect data from the machine tool and to a user application is shown in Figure 2. The agent reformats data from the adapter based on XML schema. The XML schema are typically stored in an XSD file [4]. The agent then corresponds

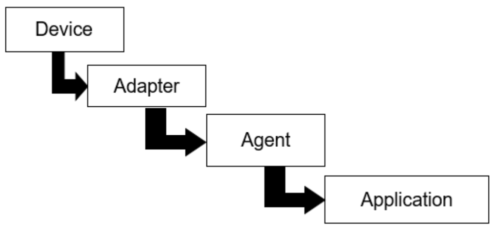


Figure 2. MTConnect Signal Flow from Device to Application

the schema with the decices.xml file to output data. The Devices.XML file houses the machine-specific data items. As data are sent to the agent over time, data items are matched to their corresponding elements in the XML file. The adapter sends the data in the following format: timestamp | (data item ID, Name or Source) | data output. Data not displayed in this format is not usable by the agent and will report an error. On many modern CNC machine tools, the adapter is pre-installed and capable of sending machine data to an agent. A simple network connection between the machine tool and the factory network will allow users to make use of MTConnect data. However, because legacy machine tools comprise a large portion of assets deployed by manufacturers, implementation of an MTConnect-based system can be a challenge due to the cost of third-party hardware adapters that must be installed on the machines to make them MTConnect-compatible. This can be a roadblock for some facilities, especially SMEs.

Related Work Using MTConnect

The concept of cyber-physical systems (CPS) in manufacturing spans a number of interrelated fields, including big data, networked manufacturing and cloud computing [9]. One of the key elements in a networked manufacturing system is the connection of manufacturing equipment to other devices on a factory's network; this allows manufacturers to leverage and communicate with their shop floors more easily. Modern facilities are increasing connectivity throughout the factory using various network strategies that can lead to higher productivity [10], [11]. However, for effective networking within the factory, manufacturing equipment must be capable of communicating using a common protocol. MTConnect is frequently the protocol of choice for machine tool applications, as it is widespread and simple to use [12]. For these reasons, MTConnect has been implemented in many applications previously. Atluru and Deshpande performed a study on the smart machine platform initiative (SMPI) test-bed with a GE-Fanuc controller in order to data monitor the machine tool [13]. The MTConnect adapter on this controller was built in C++ and would transfer machine data to the MTConnect agent in the XML format. This study validated the use of MTConnect in a shop floor environment. The data was gathered using a program written using the Ruby on Rails platform and stored in a SQLite3 database. The MTConnect data was then used in a Java application in order to reduce volumetric error for machine tool metrology. Ridwan and Xu developed a CNC system that enabled optimization of the federate in a machining process to decrease machining time and increase part quality. The system relied on a Knowledge-Based Evaluation module that received process data using the MTConnect protocol in order to modify STEP-NC data during the machining process [14]. Shin, *et al* demonstrated a virtual machining model to analyze a machining process using MTConnect data. The authors employed STEP-NC as feedforward and demonstrated interoperability with MTConnect and STEP-NC in a two-axis turning operation [15]. Franca, Torrisi and Bottene used MTConnect data in order to collect machine tool probe results from an inspection routine [16]. The MTConnect adapter for this study was developed in C# in order to transfer machine data to the agent. A Java program was developed to receive the XML data from the agent using HTTP requests. The Java application parsed the XML file to separate the desired information. In contrast to these two local implementations, Edrington, *et al* demonstrated a web-based monitoring system using MTConnect [17]. The MTConnect agent used in this study was given Internet access by assigning it a global static IP. The agent was then monitored using the DMG Mori Seiki Messenger server. The Messenger gathered the agent's data, analyzed and stored them in a database; additionally, an XML file was produced and hosted to enable users to monitor the data with any internet-enabled device.

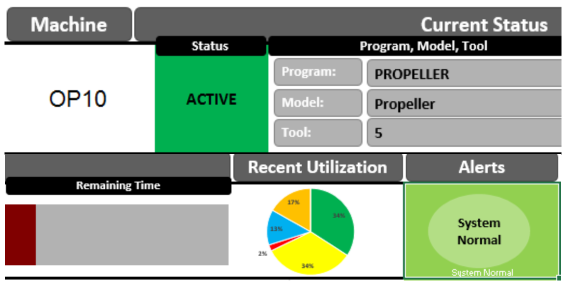
The current work presents both local and cloud implementations of MTConnect-based machine tool monitoring systems that are capable of collecting data output from multiple machines. These systems were developed using a variety of tools to enable efficient data collection, storage and visualization. The data collected using these systems can be used to monitor production status of machine tools historically or in a dashboard format. The remainder of this paper is organized as follows: first, a Microsoft Excel implementation of a machine monitoring system is presented; next, a solution relying on a MySQL database is discussed and analyzed; finally, options for migration to a cloud solution are presented and analyzed.

MICROSOFT EXCEL IMPLEMENTATION

Wescoat and Lynn demonstrated an interactive and rapidly-deployable machine tool monitoring system that was built using Microsoft Excel at the International Manufacturing Technology Show 2016 [18]. This implementation organized data in a dashboard format that would allow both managers and machine operators to reliably check and save machine data. Additionally, the dashboard would allow an operator to easily check all of the machines under his or her management. Another reason for creating a multiple machine dashboard is implementation for an automated machine cell. As parts are run through the machine cell, operators cannot check parts without stopping the other machines in the cell and losing production time. Utilizing a dashboard populated by MTConnect data, operators can effectively and efficiently check multiple machines.

The platform for the Operator Dashboard was Microsoft Excel. Excel is a powerful tool that allows for both back-end VBA code development and front-end analytics using cell formulas. The first step in the creation of a machine database and interactive website was the use of the front- and back-end tools. Excel served as both the database of data and the front panel that interacted with the machine operators, engineers and managers. The machines' MTConnect agents provided easily-accessible data at a specific time interval using the VBA code and query tables from Excel. Excel has a built in XML parser that is able to process and map data to specific cells.

As an example of MTConnect data usage, utilization pie charts were created by measuring the amount of time that a given machine spent in different execution states. The execution state defines whether the machine is stopped, in manual mode or running full production. The dashboard displayed this pie chart to track the percentage of a work shift that each machine is in a specific state. The output of the dashboard is shown in Figure 3. Managers and operators can



OP10

ACTIVE

Program: PROPELLER

Model: Propeller

Tool: 5

Figure 3. Typical Output of Machine Performance Based on MTConnect Data

utilize this pie chart to easily make sure that their machines are being optimized.

The Excel solution was both easily implementable and low cost, as Excel is a common item in the workplace. The dashboard can be uploaded and after going through the setup process can run smoothly over the course of a work shift. The data collected by the dashboard can also be sent to other networked dashboards to be analyzed and stored separately.

MYSQL IMPLEMENTATION

Machine monitoring systems can also be implemented using open-source tools that do not require access to Microsoft Excel. An implementation of a Linux-based server as the MTConnect client was created to demonstrate the usefulness of these open-source tools in the manufacturing environment. These tools can be used for the three crucial steps in a successful machine monitoring application: data collection, storage and visualization. A Python application was developed to send HTTP requests to the MTConnect agent and receive the resulting XML data. The application then parsed the data to the desired information and store them in a local MySQL database. The web server also provides a secured read-only remote access to the data stored in the database.

Data Collection

An MTConnect client application was developed in Python to collect data from networked machine tools by sending HTTP requests to the MTConnect agents of the machines. Python was chosen as the platform with which to develop the application, as it is a free and open-source software. Python has many attractive qualities; for instance, it is both easy to read and relatively simple to write thanks to the rich, well-designed built-in libraries and the availability of many third-party open-source libraries and modules.

Communication between the MTConnect client application and the MTConnect agents was accomplished using the TCP/IP protocol. As is standard practice with TCP/IP communication, the client initiates a TCP/IP connection with the HTTP server that it wishes to retrieve data from. In this case, the servers are the MTConnect agents of the machine tools to be monitored. The client sends /current commands to the agents to get up-to-date XML pages containing real-time data. The HTTP requests were performed using Python Requests, an HTTP library for Python [19]. To address issues of cyber security, the MTConnect client runs on a front-end Linux computer and communicates with the CNC machines through a local Ethernet network. Such a system is inexpensive to implement, immune to external security threats and has low network latency. The architecture of the local network that was designed and implemented for this system is shown in Figure 4.

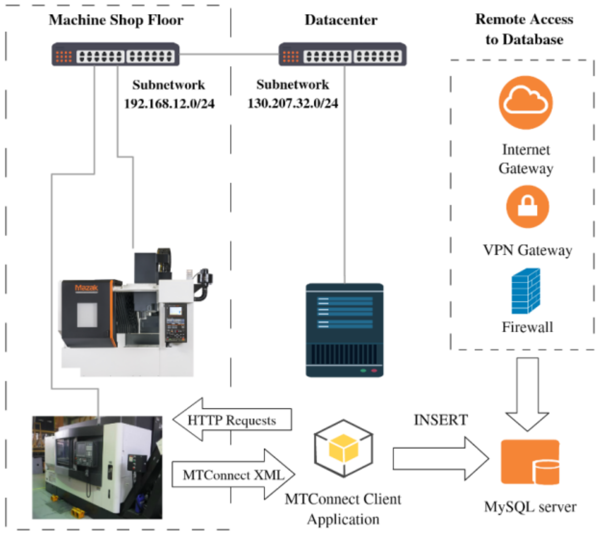


Figure 4. Network Architecture

The MTConnect client is a multi-threaded application that allows asynchronous data collection from multiple different machines. The multithreaded application makes the data collection from each machine independent from the other ones; it is not necessary to wait for one machine to answer before polling the data from the other machines; additionally, the multi-threaded application is resistant to incidents, such as communication interruptions or machine shutdowns. The pseudo-code describing the procedure for launching a thread for each individual machine is shown in Figure 5.

```
/*Create a n empty list of Threads*/
threads = []
/*Create a list of machines IP addresses*/
ip_list = [IP]
/* Create a Thread for every IP address and execute the
function main_script(IP)*/
for ip in ip_list:
        threads[new thread(IP)]
```

Figure 5. Multithreaded Data Collection

In the case that the HTTP server of an agent doesn't respond due to a shutdown, the thread tries to reestablish a connection every second using the Python event scheduler [20]. This not only allows for the capture of connection errors, but also data collection and storage resumes almost immediately once the agent is back online. The design of this application allows for near real-time knowledge of factory-floor machine status data. The application is modular, as it is simple to adjust as more machines need to be added. The addition of a new machine to the system is as simple as connecting the machine to the switch on the shop floor and adding its IP address to the list of IP addresses to be monitored in the Python script.

Once the XML data has been received from the agent, the Python application parses the XML into a form that is suitable for storage in a database. Fortunately, Python does not suffer from a scarcity in XML libraries. For this implementation, the ElementTreeXML API was used to parse the XML file pass the data along to a database. To gather data as quickly as possible, each thread sends a new HTTP request to the MTConnect agent it is monitoring immediately after completion of the previous request and parse.

Results and Analysis of Data Collection

Time taken by the application to gather and parse the MTConnect XML files can affect the performance of the system, and must be analyzed to determine how quickly the application can record new data. The time taken by each of the main functions of the MTConnect client application was measured for one thread to enable future optimization of the application. Sequences of code that take relatively long amounts of time to execute can be targeted first when attempting to improve the performance of the code. Due to network fluctuations and the fact that the data in the current XML file returned by the agent are not static, the time taken in each step of the data collection procedure has some variability. A total of 10 measurements were taken and the average and standard deviation of those measurements are shown in the second column of Table 1. The reported data show that getting the current MTConnect XML file from the HTTP server is only 0.47% of the total execution time of the whole application. This is expected, since the server and the client are close and part of the same LAN. Parsing the XML file takes 5.68% of the total time which is acceptable as parsing is considered among the main functions done in order to collect the data and store it. The rest of the time taken by the application is due to communication with and insertion into the database.

Table 1. Performance of Data Collection Application

Function	Average Time (s)	Percentage of Total Time (%)
HTTP GET	0.0042 ± 0.0002	0.47 ± 0.023
Parsing	0.0501 ± 0.0009	5.68 ± 0.102
Total	0.8818 ± 0.1601	–

Data Storage

MySQL Community Edition is a free and open-source database with many advantages such as scalability, high performance and data protection. A user that has all rights on the database is able to create the tables and store the data. The database tables were designed based on the fact that all of the data shown on the MTConnect XML page were to be stored, offering flexibility for later use. Moreover, the database must allow retrieval of the data in an efficient way, regardless of the number of the machines on the shop floor and the amount of data already in the database.

A one-to-many relationship is one such that each row in a table is linked to one or more rows in another table. There is a one-to-many relationship between a machine and its accompanying data; there is also a one-to-many relationship between a component_stream and the data it contains. As a result, a relational database was created based on three tables: a machines table, a component_stream table and an entity table that contains all the data. The one-to-many relationship allows frequently used information to be saved only once in a table and referenced many times in all other tables. The design of the database is shown in Figure 6. More tables could have been

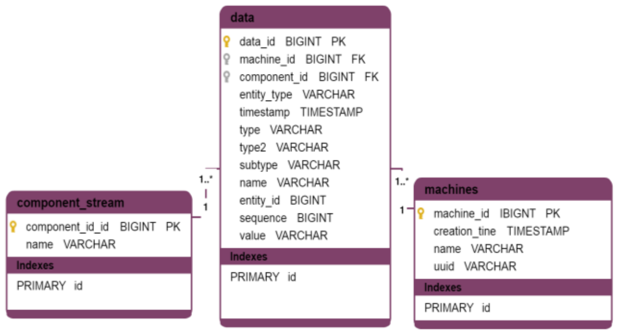


Figure 6. Relational Database Structure

created to have a normalized database, but that would require the frequent joining of different tables. Because the join method is costly operation, denormalizing a part of the database helps to expedite queries that require many table joins. MySQL executes joins between tables using a nested-loop algorithm that reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join [21]. This process is repeated as many times as there remain tables to be joined. Locating a record is quite a costly operation which may take dozens of times as long as the pure record scanning depending on the size of the table.

The main difficulties that were faced during the process of storing data reside in the fact that some data formats are different for different machines. For example, the timestamp of one machine tool on the network had one more digit than another machine tool, which necessitates processing of the timestamp into the proper format for the MySQL database. Furthermore, the MTConnect server sends the same data if it receives a request while some of data have not changed. Therefore, a method to avoid storing duplicated data is required before inserting in the database.

Results and Analysis of Data Storage

Considering that the aim of a machine tool monitoring system is to collect and store near real-time data as quickly as possible, the speed of the application at every step is crucial. The main functions related to the database on the MTConnect client are INSERT queries, which store data into the database; SELECT commands, which gather records from the database; and COMMITs, which upgrade the database. Each of these operations was timed for this implementation, and the results of ten trials are presented in Table 2. Since the machines and the component_stream tables are small, the time taken to perform the SELECT query is negligible.

Table 2. Performance of Database

Function	Average Time (s)	Percentage of Total Time (%)
INSERT	0.0304 ±0.0414	3.45 ± 4.704
Table Checks	0.0068 ±0.0052	0.771 ± 0.590
COMMIT	0.7440 ±0.0532	84.37 ± 6.033

The INSERT query is made after parsing a line of the MTConnect XML data and stores the entire line into a row in the database with each data item in the corresponding column. The INSERT is then executed as many times as there are lines of available data in the XML file. Since the database is local to the data collection computer, the INSERT takes only 3.45% of the total execution time. Before inserting the data in the database, some checks are performed to ensure the relations between the tables. The time taken to perform these checks is shown in the second row of the table. The first check is the comparison of the machine name in the XML file to the machine names stored in the machines table. If the current machine does not exist, a new row is created and the new machine is stored automatically. The same process is done with component streams. This offers flexibility when adding new machines to monitor, as no operator involvement is needed to create space for a new machine in the database. If either the machine or component stream is already stored in the machine or component stream tables, the machine_id is retrieved and kept in a local variable to be used as a foreign key when the INSERT is executed.

The COMMIT statement ends a transaction with the relational database and makes all changes visible to other users. This is done in a series of operations. First, a new log record describing the COMMIT to the log in RAM is added; next, all log records that have not been written to the disk, including the most recent record, are written to the disk; finally, the COMMIT is completed once the operating system reports that the disk write was successful. As multiple threads are inserting data into the database simultaneously, it is necessary to COMMIT after every INSERT. This process takes a significant amount of the total time reported in the table.

Data Visualization

A front-end website that enables data visualization was implemented to utilize the near real-time data in the MySQL database. A website was chosen to allow for simple accessibility and user-friendly interface. To allow visualization of machine data by factory personnel, the website can display updated graphs of various data items from the machines' MTConnect output. A Python program using the Flask API was used on the web server to retrieve the data from the database. Remote database access was needed to allow the Flask application to retrieve the data from the database; the MySQL server allows for remote access to a user that is given specific privileges on a specific database. Thus, the webserver was set up as a read-only user and given access to the database. Once the Flask application retrieved the required data from the database, they were used to generate plots for use in the factory. The Data-Driven Documents (D3.js) library was used for plotting the data in the browser. In order to make the data suitable for use with D3, the data were parsed into the JSON format before being passed to the D3 JavaScript in templates. Figure 7 shows an example of JSON data of the machines table.

```
[{"machine_id": 1, "creation_time": "2016-09-08 16:39:35",
"machine_name": "Mazak"}, {"machine_id": 2,
"creation_time": "2016-09-09 19:02:51", "machine_name":
"OKUMA.Lathe"}]
```

Figure 7. Example of JSON Data from the Database

Figure 8 shows the first page of the website, which displays the list of equipment on the shop floor. The images of the equipment are used as links to the page specific to the selected equipment. Figure 9 shows the rolling menus that display the list of the programs that have run on the machine.

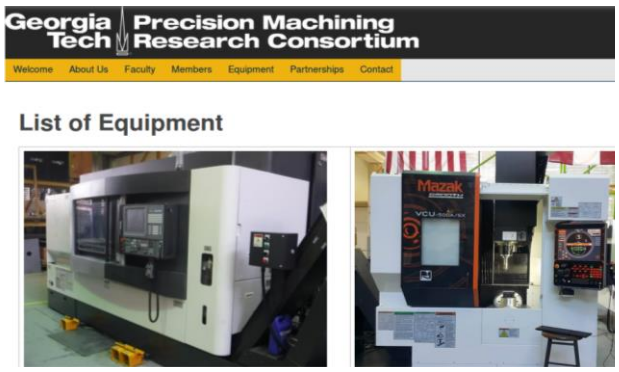


Figure 8. Equipment on the Shop Floor

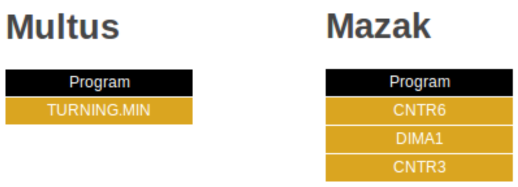


Figure 9. Rolling Menus for each Machine

Graphs of various data items can be displayed by program, allowing for analysis of the machining process throughout the run. An example of a plot of the spindle speed data over time for a given lathe program is shown in Figure 10.

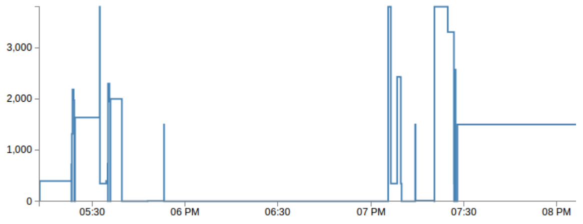


Figure 10. Spindle Speed Chart

System Optimization

Optimization of both execution time and storage efficiency is crucial for near real-time applications. The MTConnect client application was designed to run as fast as possible by avoiding unnecessary operations. For instance, the connection to the database is done only once for each thread, and maintained as long as the application is running. Additionally, instead of storing the entire MTConnect current page after every HTTP request to the machine, only the available data items that have changed since the last request are stored. In order to avoid storing duplicate data, the two possible solutions were explored: the first was to delete and replace old data if it exactly matched data obtained by a newer /current request. The data were considered to be unique if the follow three criteria were met: the machine_id matched the previous data set, the timestamp was newer than the previous data set, and the sequence number was different from the previous data set. Although this solution is simpler to code and allows functionality with all data sets, it is time consuming as its execution speed is dependent on the size of the database. The second, and more efficient, solution was avoiding MySQL queries and using a local dictionary that was updated every time a data record was stored. To ensure data integrity, a database commit was executed after every insert. Although this procedure is time consuming, it ensures data recovery is possible and guarantees that the available data are actually correct. Committing was performed after inserting all the data of a component stream, as it is more efficient than committing after every row.

MIGRATION TO A CLOUD SOLUTION

Globalization is one of the common challenges of manufacturing and has increased the need of cloud manufacturing (CM) [9]. The benefits of CM include decreasing the costs and increasing the scalability of a manufacturing process by providing access to the resources through the cloud [22]. Providing the data to any Internet-connected device anywhere in the world is one the main advantages of the cloud concept, and the global availability of data has been beneficial to the manufacturing industry [23]. As a result, in order to access the stored data, there is no need to connect to the local networks and databases. Internet-enabled devices can access the data with commonly used protocols, such as HTTP. Since MTConnect uses standard XML and HTTP protocols, it is highly compatible with web communications.

Providing a reliable database can be expensive and challenging. Additionally, companies providing cloud services usually update their systems frequently to provide the fastest and most reliable services to their customers. Therefore, the use of cloud services could eliminate the need of manufacturers to purchase servers and manage their own database. Another advantage of a cloud solution is cloud computing (CC), which enables the construction of web apps that run scripts and programs on the cloud. Using CC, both computation and data analysis can be performed on a cloud platform; this can be beneficial to the manufacturing industry, as some applications may not need to have a separate computer for data analysis. Additionally, low-power devices, such as Internet-of-Things (IoT) platforms or embedded control systems, can make use of CC to perform computationally-intensive operations that would not be possible with the available computing power of the device itself.

Three different methods of publishing MTConnect data to the cloud, specifically to Google Sheets, are described below. In order to communicate with a Google Sheet, the creation of a web app using Google Scripts is required. The data to be published on a given spreadsheet can then be transmitted with a HTTP request to the web app. Cloud solutions demonstrated below are categorized based on whether the machine is connected to the Internet or if it is only accessible using a local network.

Method 1: Publishing Data from Machines Not Connected to the Internet

In this method, a program running on a computer connected to a local network sends HTTP requests to an MTConnect agent to receive the machine's data in the XML format. The data can be either parsed locally and then transferred to the web app, or it can be sent to the web app and then parsed in the cloud. The architecture for this solution is shown in Figure 11. In order to send the whole data with the HTTP request as a parameter, the received XML should be converted to a HTTP compatible format such as JSON. The web app will then be able to parse the JSON data and insert them into a spreadsheet. Both historical and current data can be retrieved from the spreadsheet by sending HTTP requests to the web app which provides access to the data for all internet connected devices.

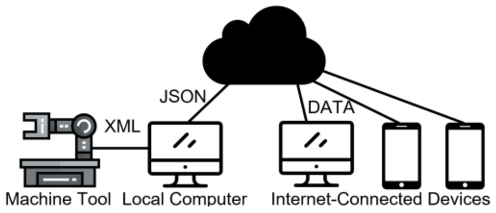


Figure 11. Publishing Data from Local Machines to the Internet

Method 2: Uploading the MTConnect XML to a Host

Similar to the previous method, this method relies on a computer connected to the local network. In this case, the computer is used to execute a program that sends HTTP requests to MTConnect agents and directly upload the returned XML files to a web host. The architecture for this method is shown in Figure 12. In this configuration, Internet-connected

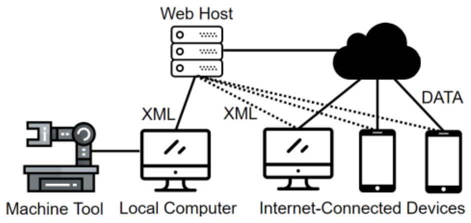


Figure 12. Publishing MTConnect XML on a Host

devices to have direct access to the XML file generated by MTConnect in a near real-time. As a result, the machine tools are not connected to the Internet; however, Internet-connected devices can collect the data by reading the XML file from the host server. A web app, such as a Google Apps Script Web App, can read the XML file from the host, parse the data, and publish the information in a Google spreadsheet or to a cloud database. In addition to getting the latest data from the XML file, both current and historical data can be retrieved from the spreadsheet by sending HTTP requests to the web app as was done in the first method.

Method 3: Publishing Data from Internet-Connected Machines

In this method, the MTConnect agent has a public static IP. Therefore, since the agent is connected to the internet, there is no need for a separate computer to gather the data. Instead, a

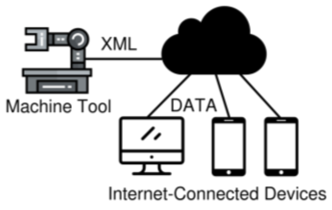
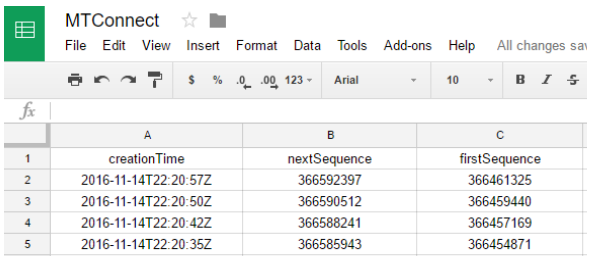


Figure 13. Publishing Data from Internet-Connected Machines

web app, for instance one made with Google Apps Scripts, can communicate directly with the agent using HTTP requests, receive the XML file, parse the received data, and store them in a cloud database. The architecture for this method is shown in Figure 13. Similar to the other two methods, the web app can also provide the current status as well as historical data of the machine to the other Internet-connected devices.

Performance Analysis of Cloud System

The first two methods discussed above were implemented and studied to determine their performance in a shop floor environment. A C# program was written that performed all communication and parsing of the data. The program communicated with the agent using HTTP requests and then parsed and converted the received XML into JSON. Next, it transferred the JSON string with a HTTP request to a web app written in Google Scripts. The web app received the data, parsed it and published it into a specified Google Sheet. A screenshot of the published data to the Google Sheet is shown in Figure 14. Timings for different tasks of this method for


MTConnect
File Edit View Insert Format Data Tools Add-ons Help All changes sa
Arial
10

	A	B	C
1	creationTime	nextSequence	firstSequence
2	2016-11-14T22:20:57Z	366592397	366461325
3	2016-11-14T22:20:50Z	366590512	366459440
4	2016-11-14T22:20:42Z	366588241	366457169
5	2016-11-14T22:20:35Z	366585943	366454871

Figure 14. Published Data on a Google Sheet

inserting eight elements in Google Sheet are also shown in Table 3. As discussed in the first method, part of the script of

Table 3. Experimental Results of Transmitting JSON to the Web App and Parsing the Data in the Cloud

Task	**Time (s)**	**Percentage of Total Time (%)**
Local Conversion of XML to JSON	0.0002 ± 0.0001	0.0075 ± 0.0019
Transmit JSON to Web App	0.5270 ± 0.2400	19.8695 ± 9.0488
Parse and Insert into Spreadsheet	2.1251 ± 0.3805	80.1229 ± 14.3460
Total	2.6523 ± 0.6206	–

this web app was dedicated to transmitting the data as the HTTP response. The data, therefore, could be retrieved by communicating with the web app.

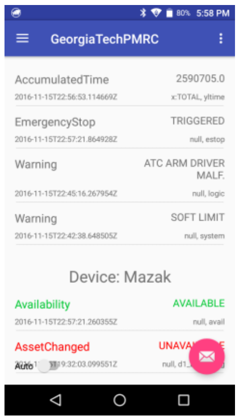
As discussed in the first method, the data could also be parsed locally and then transferred to their corresponding cells on the Google Sheet. The timing results of this experiment, which parsed eight elements and published them one-by-one to the Google Sheet are shown in Table 4. Comparing the total time in Tables 3 and 4, it is apparent that parsing the data locally and sending each element with an HTTP request to the spreadsheet is less efficient; in this case, local parsing and

Table 4. Experimental Results of Parsing the Data Locally and Transmitting them to the Google Sheet

Task	Average Time (s)	Percentage of Total Time (%)
Parse XML Elements	0.0010 ± 0.0005	0.0261 ± 0.0130
Serially Transmit Elements to Google Sheet	3.8320 ± 0.2800	99.9739 ± 7.305
Total	3.8330 ± 0.2805	–

serial transmission requires approximately 144% more time than transmitting the whole data set and parsing it in the cloud.

For the second method, a Linux computer with two network cards was used. One card was connected to the local machine shop's network and the other was connected to the Internet. C# code was executed on the computer that sent HTTP requests to MTConnect agents, received XML files and uploaded them to a host using Secure File Transfer Protocol (SFTP). This method had an approximate update speed of 2.1041 ± 0.1660 seconds over the course of eight trials. Using this method, all Internet-connected devices can get access to the original XML file on the host, even though the machine tool is not directly connected to the Internet. Additionally, an Android app was developed that displayed the MTConnect data that were collected using this method. A screenshot of the Android app is shown in Figure 15.



80% 5:58 PM

GeorgiaTechPMRC

AccumulatedTime	2590705.0
2016-11-15T22:56:53.114669Z	x:TOTAL, yltime
EmergencyStop	TRIGGERED
2016-11-15T22:57:21.864928Z	null, estop
Warning	ATC ARM DRIVER MALF.
2016-11-15T22:45:16.267954Z	null, logic
Warning	SOFT LIMIT
2016-11-15T22:42:38.648505Z	null, system

Device: Mazak

Availability	AVAILABLE
2016-11-15T22:57:21.260355Z	null, avail
AssetChanged	UNAVA[illegible]
Auto [illegible]9:32:03.099551Z	null, d1_[illegible]

Figure 15. Android App for Displaying MTConnect Data from a Web Host

DISCUSSION AND FUTURE WORK

There are advantages and disadvantages for both local and cloud solutions. Either solution could be the best choice depending on the type of application. One of the main advantages of storing the data locally is cybersecurity. Inserting and retrieving data from a local database is not only faster than a cloud solution, but it also does not introduce additional security vulnerabilities that are introduced by an Internet connection. Additionally, isolating the machines from external devices and from the Internet is crucial. Many machine tools run legacy operating systems and are left completely vulnerable to modern cybersecurity threats. In the local database solution presented in this work, the CNC machines were only exposed to a small local network and thus were isolated from external threats.

In the local implementation, the data management server that ran the MTConnect client application and the database was equipped with two network cards. This allowed the server to communicate with the CNC machines' subnetwork while also communicating with the larger factory network to host the visualization webpage. Another solution that isolates the CNC machines from the factory network is to use a managed network switch. A production-hardened hardware platform, the Mazak SmartBox, has been installed on the factory floor and will be the focus of future work in this area. The SmartBox, shown in Figure 16, connects manufacturing equipment and

Figure 16. Mazak SmartBox

allows data collection safely by isolating the machines on independent virtual local area networks (VLANs). It is also possible to separate the machines on the shop floor by groups using multiple different VLANs to limit possible attacks.

The SmartBox has a separate VLAN for an embedded Linux kernel that operates as an MTConnect agent to collect data from multiple machine adapters. This allows for the MTConnect agents built into the machine tools to be shielded from the outside network. This is an example of fog computing, where computational power is decentralized and distributed throughout a network. Fog computing is a promising concept for the future of the IoT and big data. The Linux core is accessible from the outside network to allow devices to access manufacturing data, and only this core, not the machine tools, will be affected in the case of a cybersecurity breach.

While the local solution has distinct advantages over the cloud solutions, implementation of a local solution necessitates both the purchase and management of database hardware and other servers. Additionally, remote access must be configured for the users who are granted some privileges on the database. Only computers inside the VPN can access the database and a firewall is configured on the MySQL server to filter by IP address. Easy access to the stored data from Internet-connected devices outside the local network might be challenging or even sometimes impossible, as not all devices are compatible with all databases. The cloud solutions presented in this work are free to use and provide an attractive alternative to the local solution. Future studies will explore larger-scale cloud implementations of MTConnect-based monitoring systems in order to further explore their suitability in a manufacturing environment.

CONCLUSION

To address the need for a smart manufacturing shop floor, MTConnect machine monitoring systems were developed and presented in this paper. These systems were used to collect data from multiple CNC machine tools and display those data on user-friendly and accessible interfaces. Three distinct tools were developed and described: a machine dashboard based on Microsoft Excel dashboard; an MTConnect client application written in Python that interfaced with a MySQL database and custom website; and various cloud solution and an accompanying Android application. Results from time studies on the performance of these tools were presented, analyzed and discussed. Solutions for optimization to improve the speed of data collection were implemented and presented. Issues of cybersecurity were addressed, and ideas for future work were presented and analyzed for viability.

ACKNOWLEDGMENTS

This work was supported by NSF grants CMMI-1547093 and IIP-1631803.

REFERENCES

[1] "Data rEvolution." CSC Leading Edge Forum, 2011.

[2] R. Lynn, A. Chen, S. Locks, C. Nath, and T. Kurfess, "Intelligent and Accessible Data Flow Architectures for Manufacturing System Optimization," in *IFIP Advances in Information and Communication Technology*, vol. 459, 2015, pp. 27–35.

[3] E. J. Martin, "Dark Data : Analyzing Unused and Ignored Information," *EContent*, vol. 39, no. 5, pp. 6–8, 2016.

[4] W. Sobel, "MTConnect Standard Part 1 - Overview and Protocol," *The Asociation for Manufacturing Technology*, pp. 0–70, 2012.

[5] W. Sobel, "MTConnect ® Standard Part 2 – Device Information Model," 2014.

[6] W. Sobel, "MTConnect ® Standard Part 3 - Streams Information Model," 2014.

[7] B. Kiefer and J. Evans, "MTConnect Application Development."

[8] D. Wickelhaus and X. Wang, "Data Acquisition : Building an MTConnect Adapter," 2015.

[9] B. Buckholtz, I. Ragai, and L. Wang, "Cloud Manufacturing: Current Trends and Future Implementations," *Journal of Manufacturing Science and Engineering*, vol. 137, no. 4, p. 40902, 2015.

[10] D. Li, "Perspective for smart factory in petrochemical industry," *Computers and Chemical Engineering*, vol. 91, pp. 136–148, 2015.

[11] D. Ivanov, A. Dolgui, B. Sokolov, F. Werner, and M. Ivanova, "A dynamic model and an algorithm for short-term supply chain scheduling in the smart factory industry 4.0," *International Journal of Production Research*, vol. 54, no. 2, pp. 386–402, 2016.

[12] P. Kuhnell and A. Deshpande, "The Open-Source Manufacturing Stack."

[13] S. H. Atluru and A. Deshpande, "Data to Information: Can MTConnect Deliver on the Promise?," *Transaction of NAMRI/SME*, vol. 37, pp. 197–204, 2009.

[14] F. Ridwan and X. Xu, "Advanced CNC system with in-process feed-rate optimisation," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 3, pp. 12–20, 2013.

[15] S.-J. Shin, J. Woo, D. B. Kim, S. Kumaraguru, and S. Rachuri, "Developing a virtual machining model to generate MTConnect machine-monitoring data from STEP-NC," *International Journal of Production Research*, vol. 54, no. 15, pp. 4487–4505, Aug. 2016.

[16] T. V. França, N. M. Torrisi, and A. C. Bottene, "CNC Machine Tool Monitoring Using MTConnect Communication Architecture," *22nd International Congress of Mechanical Engineering*, no. Cobem, pp. 6660–6669, 2013.

[17] B. Edrington, B. Zhao, A. Hansel, M. Mori, and M. Fujishima, "Machine monitoring system based on MTConnect technology," in *Procedia CIRP*, 2014, vol. 22, no. 1, pp. 92–97.

[18] E. Wescoat and R. Lynn, "Monitoring Machines using Excel and MTConnect: Tracking Machine Utilization using Off-the-Shelf Products." AMT MTConnect Student Challenge, Chicago, IL, 2016.

[19] "Requests: HTTP for Humans." Python Requests Documentation, 2016.

[20] "sched - Event Scheduler." Python 2.7.12 Documentation, 2016.

[21] "Nested-Loop Join Algorithms." Oracle MySQL 5.7 Reference Manual.

[22] L. Wang, "Machine availability monitoring and machining process planning towards Cloud manufacturing," *CIRP Journal of Manufacturing Science and Technology*, vol. 6, no. 4, pp. 263–273, 2013.

[23] X. Xu, "From cloud computing to cloud manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 1, pp. 75–86, Feb. 2012.