Journal of Manufacturing Systems 45 (2017) 236–247

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

journal homepage: www.elsevier.com/locate/jmansys

Full length article

A flexible data schema and system architecture for the virtualization of manufacturing machines (VMM)

Atin Angrish, Binil Starly*, Yuan-Shin Lee, Paul H. Cohen

Edward P. Fitts Department of Industrial and Systems Engineering, Carolina State University, Raleigh, NC 27695, United States

ARTICLE INFO

Article history:
Received 2 June 2017
Received in revised form 3 October 2017
Accepted 7 October 2017
Available online 26 October 2017

Keywords:
Cyber-physical manufacturing
Smart manufacturing
Digital twins
MongoDB
NoSQL

ABSTRACT

Future factories will feature strong integration of physical machines and cyber-enabled software, working seamlessly to improve manufacturing production efficiency. In these digitally enabled and network connected factories, each physical machine on the shop floor can have its 'virtual twin' available in cyberspace. This 'virtual twin' is populated with data streaming in from the physical machines to represent a near real-time as-is state of the machine in cyberspace. This results in the virtualization of a machine resource to external factory manufacturing systems. This paper describes how streaming data can be stored in a scalable and flexible document schema based database such as MongoDB, a data store that makes up the virtual twin system. We present an architecture, which allows third-party integration of software apps to interface with the virtual manufacturing machines. We evaluate our database schema against query statements and provide examples of how third-party apps can interface with manufacturing machines using the VMM middleware. Finally, we discuss an operating system architecture for VMMs across the manufacturing cyberspace, which necessitates command and control of various virtualized manufacturing machines, opening new possibilities in cyber-physical systems in manufacturing.

1. Introduction

The variety and velocity of machine data on a factory floor and cheaper methods to store, compute and analyze this data for real-time technical and business decision making are major driving forces to improve factory productivity. These developments are motivating the manufacturing information technology industry to re-examine traditional machine control and networking architectures present in manufacturing shop-floors. In discrete manufacturing job shops, such as those in machining services, networking multiple manufacturing machines can be difficult to achieve due to a variety of reasons. This include, interoperability concerns between machine controllers from various vendors, outdated hardware controllers and a legacy infrastructure that is incapable of handling latest network communication protocols. This hinders the transformation of physical factory floors to the digital era. Advancements made in machine communications standards (example – MT-CONNECT [1], OPC/UA [2], MQTT [3]) have certainly lowered the barriers of machines communicating with centralized information systems (such as MES and ERP systems).

The era of digitalization of manufacturing processes and its networking for efficient utilization of manufacturing resources requires a shift in the way data from manufacturing machines is handled, stored, retrieved and computed for actionable insights. Key manufacturing trends that are driving the need to rethink how data from factory floors and its extended enterprise are organized, include:

- *The need for enterprises to quickly respond to evolving market demands*: Democratization of manufacturing is leading to increasing demands for personalization or customization of products. Reconfiguration of physical infrastructure rapidly based on changing market needs and demands is necessary. Production in quantities of one, such as in personalized medical products, to quantities less than one hundred will require new ways in which data from manufacturing systems is 'pulled' to satisfy market demands. This pull of information is necessary to assess distributed capability and dynamic capacity across job-shop and production floors regionally and globally.
- *The shift to connecting product lifecycle data with the manufacturing processes*: The digital thread concept linking product information throughout its lifecycle from conception to production, then use and final disposable requires that we have fundamentally new ways in which information is linked across its various points of

* Corresponding author.
E-mail addresses: bstarly@ncsu.edu, starlyb@gmail.com (B. Starly).

https://doi.org/10.1016/j.jmsy.2017.10.003
0278-6125/© 2017 The Society of Manufacturing Engineers. Published by Elsevier Ltd. All rights reserved.

use [4]. Today, data generated during a product lifecycle is stored in siloed information systems – from PDM/PLM, to ERP/MES, to unique databases maintained by end customers. Integrating data across these disparate systems and linking across them in meaningful contexts across multiple enterprises in the supply chain can be challenging and expensive to implement.

- *Increasing granularity of data collected from manufacturing machines*: With the ubiquity of sensors (RFID and machine mounted sensors), wireless networking and cloud storage, collecting and storing large amounts of data is not a bottleneck. Analytics conducted on manufacturing data can be beneficial beyond just monitoring, but useful in predictive and prescriptive maintenance [5]. It can also be used to inform key shop-floor decisions such as production scheduling and supply chain performance [6–8].
- *The scrutiny placed on machine-to-machine connections in a network*: With all this connectivity, machine generated data requires context and meaning. Semantic models must include both explicit (raw signals, feature attributes) and implicit (operator name, part name, description), compounded with a time-reference to make sense of large machine generated data. The consumer of this data is presented with customized visualization and context to make data-driven decisions.

In a key technical position paper by Lee and Bagheri et al., the authors proposed a high-level cyber-physical system (CPS) architecture towards digital factories [9]. The primary intended purpose of CPS in factory floors is to manage Big Data from factory floors and leverage the network connection ability of machines to create the goal of resilient, intelligent and self-aware machines. Five tiered elements are proposed in the CPS architecture – 1) Connection to the physical machines; 2) Conversion of Data to Information for each individual machine; 3) Aggregating this information across a fleet of machines through 'digital twins' of machines; 4) Cognitive architecture which help synthesize the information for decision making both within and beyond an enterprise, and finally, 5) Configuration, where data insights help decision making support by individual machines themselves or by humans within the loop. This paper relates to a system architecture and a flexible data storage schema surrounding Levels 2 and 3, to enable higher order elements of a manufacturing focused CPS. At these levels, we focus on a data architecture that supports the concept of a 'digital twin' for a machine or more specifically, 'Virtualized Manufacturing Machines'.

The virtual manufacturing machine (VMM) of a physical machine encapsulates its physical capabilities (static), in-process data (dynamic) generated by controllers and any external sensors attached to the machine. The core function of a VMM is to present a middleware architecture that abstracts hardware level specifics of machines on the shop-floor and then provide a programmatic interface to allow higher order information systems to feed into service applications. The community has seen a plethora of technical phrases utilized to signify the data generated from physical machines. These terms range from 'digital twins', 'Cyber-twins', 'digital machine analogs', 'digital machine shadows' etc. At its technical core, the concept of a virtualized version of the physical machine signifies a data model that encapsulates technical specifications, machine data, and information relationship about a physical machine and its environment which then represents the machine in near real-time states within cyberspace.

The cyber-twin of a physical machine can reside within the machine's own computing system, at a server with close proximity to the machine or at a remote external cloud location. It is intended to tap in between the control/communications systems of a machine and higher order execution systems. This streaming data from a machine populates a data structure, which allows it to be analyzed within the context of manufacturing. The VMM for a machine must also provide its version of the data for cross-analysis among various other types of machines on a factory floor. We focus this study to the first part, a strategy to store structured and unstructured data from machines and have that data be made available to third-party applications.

This paper is organized as follows. In the following section, we describe components of the proposed system architecture for the virtualization of manufacturing machines. Further, we provide details on the design of a document based database schema and then critically evaluate the schema structure by testing it against two query types on streaming machine data from a metal based additive manufacturing machine. This paper will discuss the use of an unstructured database (MongoDB) as the core backbone infrastructure that instantiates the virtual manufacturing machine. The paper also proposes a high-level operating system architecture when multiple VMMs for the various manufacturing machines on a shop-floor are operating in a cyber-physical manufacturing space. We demonstrate two app cases for the VMM, one each for a low end, partially open sourced 3D printer and a high-end closed source metal 3D printer.

2. Components of the virtual manufacturing machine (VMM)

The ultimate goal of any cyber-physical manufacturing system is the ability for global enterprises to quickly respond to business and customer demands while containing costs and maintaining operational flexibility. Wang et al. [10]. discusses various types of cyber-physical systems in manufacturing. Their work expands on the examples of CPS in manufacturing such as multi-agent systems [11–13] adaptive manufacturing systems [14,15], model driven manufacturing systems [16] and cloud manufacturing [17,18]. To this end, there is an imminent need to consider advanced database systems which allow large scale storage and retrieval needs for cyber-physical systems.

In recent work, Kang et al. [19]. developed a NoSQL data store using MongoDB system for supply chains in manufacturing and performed an assessment of their traceability system with a simulation using streaming RFID data. Several papers have discussed the feasibility of deploying NoSQL type data stores for storing large volumes of streaming data from small sensors used in IoT applications [20–22]. In their studies, they have demonstrated various aspects of NoSQL databases and analyzed their performance with respect to relational databases. Boicea et al. [23] demonstrated the performance of MongoDB vs relational databases where they compared the performance of both the database systems in terms of insertion, deletion and update speeds and recommended the use of non-relational databases where high speeds are needed. Liu [24] discussed the auto-sharding capabilities of MongoDB and developed strategies to improve the concurrent read/write performance of the data structure. Several organizations have implemented MongoDB as back-end IT systems for crunching through various schema-less data and as a way to store and present data in web compatible formats [25]. Most recently, NIST has started a "Materials Genome initiative" which uses MongoDB at the backend with RESTful services to enable third-party software integration [26,27].

NoSQL databases are a distinct class of databases that do not have a prescribed schema when compared to conventional SQL based databases. The most common of such flexible schema data type stores include graph and document based stores. In document based stores, data is stored in the form of key-value pairs known as the JSON format (javascript object notation). The documents form the atomic units for a NoSQL database. This enables the NoSQL databases to efficiently handle unstructured data generated from a variety of sources. Multiple documents belonging to a same named group is called a collection. Multiple collections form a NoSQL database, which is a container for holding a single or multiple collections. Documents and collections in a NoSQL structure are analogous to records and tables in a traditional SQL structure respectively.

One of the primary differences between SQL(relational) databases and NoSQL databases is that the relational databases adhere to the principle of ACID compliance (Atomicity,Consistency,Isolation and Durability) [28] while the NoSQL databases stick to the CAP theorem [29]. This implies that the relational databases will enforce data integrity and the NoSQL databases will place a higher priority to availability of the database. This implies a SQL server is more preferable when considering applications such as banking or retail transactions where the data volumes are low and concurrency of transactions is important. NoSQL databases are unable to fulfil the consistency and concurrency requirements instantly, while focusing primarily on availability. This does not mean that consistency is not reached in a NoSQL database, it is just reached in due time. Scalability and performance take a front seat in case of NoSQL databases compared to SQL, which makes NoSQL suitable for large data volumes such as sensor and diverse manufacturing data types.

For the purposes of this study, we have used MongoDB, an open source NoSQL database with extensive use across information technology applications. Its use in manufacturing contexts are nascent and still developing. We have selected to use NoSQL over SQL structured databases for the development of VMMs based on two primary reasons:

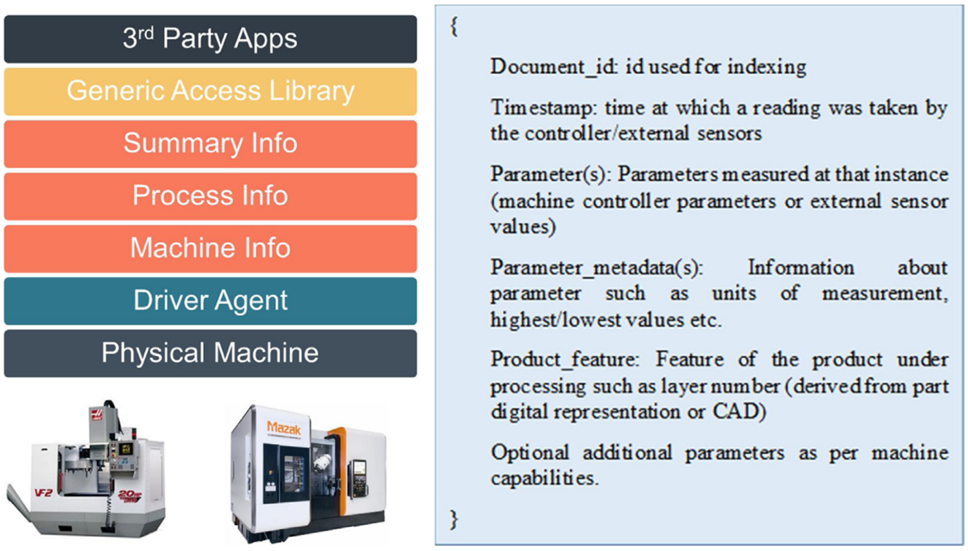
a **Generality**: Multiple types of manufacturing machines exist with different capabilities. A flexible schema type data store can help intake as much data as necessary from various machines without any modification to the data structure itself. Therefore, addition or removal of sensors or adding new capabilities to a machine will not cause data relationships to be invalid. This is unlike a schema dependent SQL structure which will need to have a pre-determined table that must be changed if the machines capabilities are changed. NoSQL document structures enables us to ignore the schema of the data and still maintain the data in a manner that maintains data integrity and validity.

b **Scalability**: SQL databases scale vertically, while document type NoSQL databases scale horizontally through the process of 'sharding' [30]. NoSQL databases enable us to scale the data by distributing it across multiple servers. This enables us to increase the storage and performance as the volume of data increases. In contrast, as the amount of data increases, the ideal way to scale a typical SQL database is by increasing the hardware and performance capabilities of a system of servers. Manufacturing machines have much faster write requirements than read or transaction speed requirements. Processes such as metal fabrication for a single part fab can easily generate TBs of data depending on the amount and type of data collected. Therefore, a horizontal scalable solution is desired to optimize performance, particularly when many types of machines exist on a factory-floor.

The primary purpose of this research is to utilize the distributed nature of the modern NoSQL database to create a virtualized version of the physical manufacturing machine. This is intended to store the data generated by a machine but also to create an economically feasible and scalable middleware architecture that can be implemented on industrial floors. The proposed architecture for a generic VMM for any physical machine is composed of 3 parts: A Driver, a Database and a Generic Machine Access Library (Fig. 1A). Each of these components are described below:

1. The **Driver** is a hardware/software machine adaptor that handles communications between the physical machine's controller and the network. The main function of the driver is to retrieve data from machine specific hardware systems and process it in a form compatible with higher order systems. The driver codes will have functions to collect appropriate controller/sensor data, identify the product feature being processed within the machine, collect sensor state, format it as a defined object and then push it into the database at a set polling frequency. Technologies such as MT-CONNECT for CNC machines can be used as an enabler to extract data out of the machines. Many legacy machines simply do not have this capability and therefore drivers must be programmed to extract this data out of the machine.
2. The **Database** implemented in this NoSQL type document database is divided into 3 layered data structures:
   a) Raw Data Layer: This layer of data structures contains the raw in-process data generated during the manufacturing of a part on the machine. The drivers continuously collect information from the machine and push it into the database – a raw data layer in the form of documents. A sample document structure is shown in Fig. 1B. A point to note is that there is no necessity for the documents schema to be similar within the collections contained in the database.
   b) Information Layer: These data structures pertain to mostly static information about the machine, most of which is populated during the instantiation of both the physical machine and its virtual version. It contains a list of the machine's assets, capabilities, sub-systems etc. Information contained here is retrieved by higher order information systems when required.
   c) Summary Layer: This layer is built on top of the raw data layer to provide applications with information about the machine's in-process info. This can include calculating feature attributes of collected data to reduce overhead and provide pre-calculated status data to systems that request it. This is especially useful when such information is calculated during a machine's downtime. When configured properly, it helps to enable rapid response to external applications that request information from a machine.
3. The **Generic Machine Access Library (GMAL)**: This library is a set of Application Programming Interface (API) functions which allow end-users to build third-party applications for the machine. The library provides functions to pull data out of the raw-data layer, thereby providing a layer of abstraction to third-party software developers. The library also offers an object oriented approach to programming. As a result, the application developers simply have to initialize an object referring to the machine and they can directly monitor and control the machines, provided access rights are granted. It offers several functions such as getMachineData(layer) [for pulling specific layer data] or sendCommand(command) [for sending commands to the machine]. The generic access libraries give us the ability to analyze and retrieve information intelligently. It is the job of the drivers below the GMAL layer to translate the requests to specific machine level code.

The machine's driver and the associated database are specific to a machine. Therefore, if we have N machines on a shop-floor, we will have N such frameworks. However, the library on top of them can be used for all the machines. Therefore, a single app can be used for accessing and analyzing the process information from multiple machines. Each machine on a shop-floor will have its own unique virtualized version associated with it. Analogous to the virtualization of computer servers, a hypervisor is required to manage and control all of the VMM's of machines in cyberspace. A 'hypervisor' or an operating system for VMMs is described in Section 4 of this paper. The following section will describe the design of the



{

Document_id: id used for indexing

Timestamp: time at which a reading was taken by the controller/external sensors

Parameter(s): Parameters measured at that instance (machine controller parameters or external sensor values)

Parameter_metadata(s): Information about parameter such as units of measurement, highest/lowest values etc.

Product_feature: Feature of the product under processing such as layer number (derived from part digital representation or CAD)

Optional additional parameters as per machine capabilities.

}

Fig. 1. (A) VMM Stack Architecture; (B) Sample Document Content for a Machine.

MongoDB document and the evaluation of its various organization types that is suited for manufacturing.

3. Design of the document based model of virtual manufacturing machines (VMM)

3.1. Document type database schema for virtual manufacturing machines (VMM)

The importance of the document based data model as opposed to the structured table type schema often seen in conventional SQL databases lies in its ability to accommodate varying data schemas. Hence such document type unstructured schemas can be better suited to building a data store for streaming data from various manufacturing machines on a shop-floor. An unstructured data schema should be able to accommodate any kind of data from any type of machine with minimal involvement in setting and defining the schema to allow data ingestion from a machine. The document type database offers convenient key-value pairing which allows us to construct complex data structures for the VMM. While a document representing a VMM does not necessarily have to be tied down to a schema, its organization is critical to the proper functioning of a VMM. This selection of the document organization is based on – the amount of data streaming in, the need for high querying/read operations on the data and the degree of scalability desired. There are 3 aspects to consider when designing the schema for the unstructured database: the selection of the primary key, the documentation organization type and the sharding strategy.

Primary Key Selection: Similar to traditional structured databases, primary keys serve as unique keys for each record in the database. Given that machines can change over time, selection of a suitable primary key reduces ambiguity and speeds up retrieval of data. The two generic primary keys suited to represent manufacturing machines especially with time-series based data, will be the default document ID and a timestamp. A document id is a simple ascending index for the documents which are generated every instance during the machine run (that is, if the machine is polled at a frequency of 10 Hz for 1 min, it would imply the machine has generated 600 documents with document_id ascending from 1 to 600). A feature of the document id as a primary key would imply each document would then hold only one data point, say, a sensor value at a particular time instant polled from the machine. The downside is that a machine reporting many different sensor values can end up creating millions of documents based on the set data frequency. This can bog down the system particularly during a read/compute operation when requested by a higher level application.

The second type of primary key is a timestamp based primary key. This would imply that a buffer collects data from the machine for a specific period and then stores the data generated by the machine. Once the time period has elapsed, it attaches a time stamp to all of the data generated during that period and then uses the time-stamp as the primary key. This way, each document stores a lot more data content compared to a document ID based method. The benefit of the time-stamp based primary key is that it would be easy to query back events that may have occurred between time periods, which may help link to specific process plan steps in the processing of a part. A similar document ID based key can also help retrieve the same data but it comes with a computational overhead required to process the request and report back the same results. With machines that report a lot of data within milliseconds, a timestamp based method is far more efficient at storing data.

Document Organization Type: Another consideration is the way in which documents within the collection is organized, particularly if a timestamp based primary key is selected (Fig. 2). If each document is based on a timestamp, then it is possible that data from many types of sensors/axes are being reported. In such a scenario, we must consider how to organize the data record within a document. There are two possible methods – a flat document and embedded (nested) documents. A flat document structure implies that each document contains only one specific schema of data in each document. An embedded document structure utilizes nested documents within the data parameters to represent multiple values or characteristics of the data generated.

Sharding strategy: The NoSQL databases may be sharded and/or not sharded depending on whether the VMM for a particular machine needs to be scaled horizontally or vertically. 'Sharding' is the mechanism to split a database into smaller, faster more nimble parts. This choice of centralization versus decentralization of data storage affects the rate at which manufacturing data can be accessed by higher order IT systems or by apps that need to fetch data about the machine. There are two types of sharding strategies – 1) Ranged Sharding, in which the primary key itself becomes the shard key. The raw data is split into chunks that are sent to different shards of the system. The configuration server holds the shard addresses and the shard keys tell the DB where the data is stored in a sequential manner. 2) The second type is Hashed Shard-

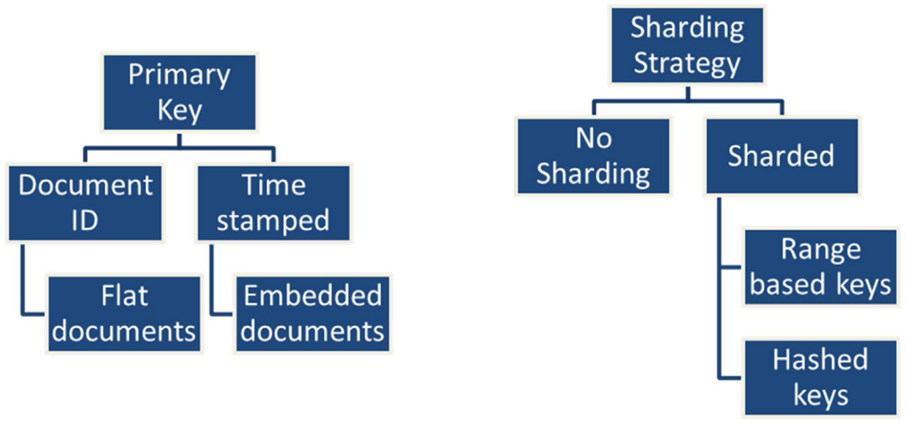


Fig. 2. Different strategies for storage of streaming machine data during the process of part fabrication.

ing. This implies that the system first computes a hash of the shard key and then uses these hashes to address chunks. While hashed sharding might seem illogical, it can be useful to ensure that all the shards are used uniformly to balance the load across shards for read/write requests. The downside is that there could be a performance loss especially during a read operation since the key must be first recovered before data is submitted back to the application.

3.2. Experimental evaluation

Based on the different types of factors that can be combined to create the data store (Fig. 2), we construct 6 different configurations of document based data structures (Table 1). Based on the 6 possible configurations, we decided to test our VMM model with two objectives: identification of the optimal configuration for querying speed and the ease of building apps using the VMM stacked architecture. This experiment was conducted with data collected from a metal based additive manufacturing machine – Electron Beam Melting (ARCAM, Sweden) with data logs created during the fabrication of the part to be organized in different ways within MongoDB. The ARCAM machine was chosen primarily because it is a closed system with no standardized external communication protocols. The machine also represents an example of a complex fabrication process whose scientific basis and modeling is still being explored. A driver was written for the collection of data from the ARCAM EBM machine and it was then directed into the MongoDB system. The ARCAM EBM data is an example of a data stream with varying schema structure depending on the part being fabricated. The data generated by the controller is not periodic. The machine may record 5 data points at one instance and then possibly 10,000 data points at another instant in time. The type of data collected also varies depending on the amount of process monitoring equipment attached to the system. The drivers were written to parse through the data and then sent to the data store in the 6 configurations noted in Table 1.

The sharding setup was as follows: The system comprises of two shards, one configuration server and 1 mongos server (Fig. 3). This setup is similar to the one set up by Kang et al. [14]. For a detailed description of the MongoDB configuration, the readers are directed to its manual which describes the function of the mongod and mongos components of the database [29].

A randomized test was conducted with the 6 configurations. We conducted 10 replicated queries with 2 different kinds of query types. One query was a simple query where the system is queried for a manufacturing parameter either with a document id or with a time stamp. The other query is a more complex one where the system is required to check for the existence of a particular manu-

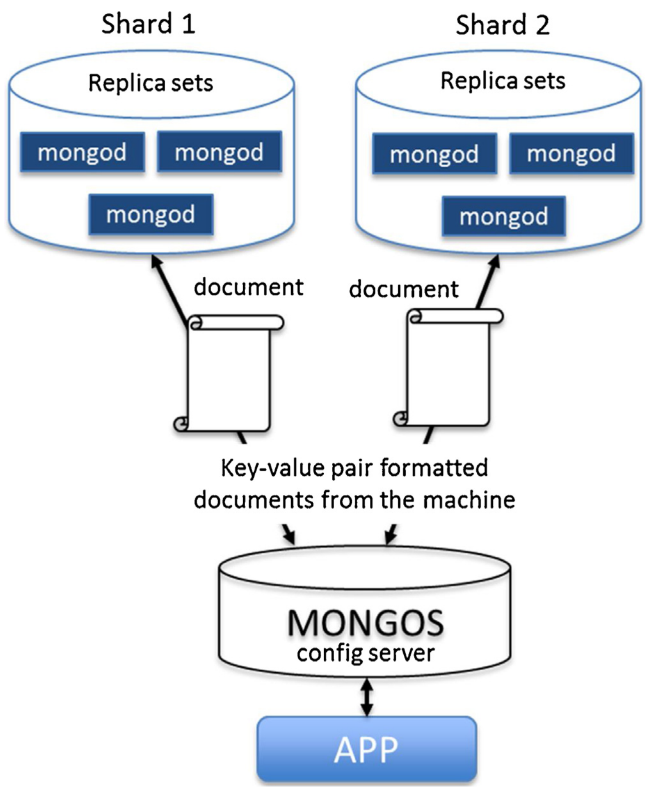


Fig. 3. Representative Experimental setup of the Physical Organization of the Data by the server and associated storage drives. Any number of shards can be configured to make the system scalable across machines and factory floors.

facturing parameter, perform calculations and then return a usable list to the app. After each query execution, the response speed was calculated. The queries are as follows in Table 2. Based on the above experiments, we generated a matrix of size 6×10 for each kind of query. The response times were measured and subjected to a Tukey-HSD pairwise comparison. The TukeyHSD tests for significant differences in the means for different configurations, taken two at a time for analysis.

As shown in Fig. 4A and B, the sharded configurations outperform the unsharded configurations in general which plays to the strength of having sharded MongoDB database configurations. This supports the findings of Kang et al., which indicate that increasing the number of shards significantly increases the query response rate. Timestamped based documents are also much better in terms of organization versus just using document-id as the primary key.

Table 1
All possible combinations for document structures.

Configuration	Primary Key	Document type	Sharding strategy	Shard key type
C1	Timestamp	Embedded	Sharded	Hashed
C2	Timestamp	Embedded	Sharded	Unhashed
C3	Document_id	Flat	Sharded	Hashed
C4	Document_id	Flat	Sharded	Unhashed
C5	Timestamp	Embedded	Unsharded	N/A
C6	Document_id	Flat	Unsharded	N/A

Table 2
Query Types and Queries.

Query Type	Query	Parameters
Simple (querying for a key value pair to return all document(s) which match the criteria)	db.rawLayer.find ({document_id: value})	Document_id Value: a random value generated by the program
Complex (queries requiring computations before results are returned)	records = db.rawLayer.find({"$and": [{"time_second": {"$gte": search}}, {"Core_Cache_Performance_MaxCycleTime": {"$exists": True}}]}, {"Core_Cache_Performance_MaxCycleTime": 1, "time_second": 1, "_id": 0})	Time_second: timestamp after which we need the results Core_Cache_Performance: system parameter being measured

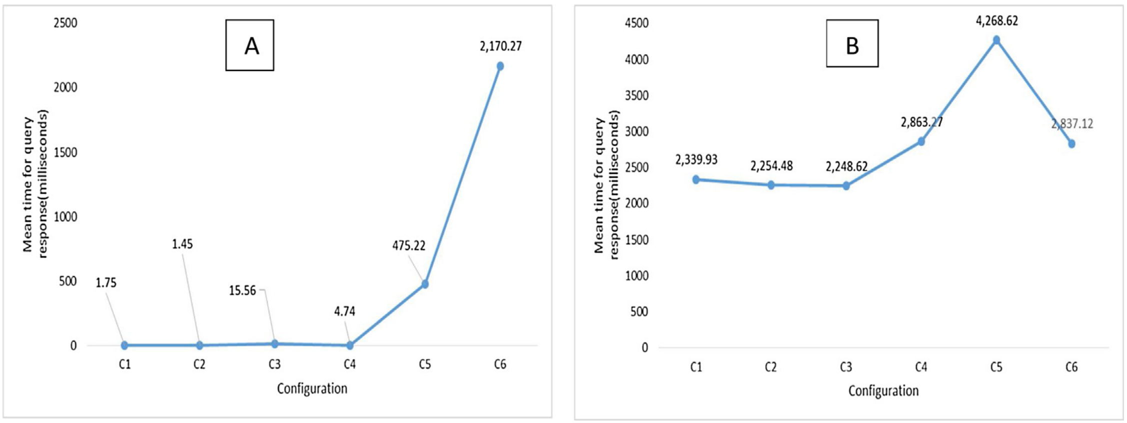


Fig. 4. (A) Simple Query response time vs Configuration; (B) Complex Query response time vs Configuration.

Query times are significantly different from the others (if the value of p adjusted is <0.05, the difference is significant). Clearly, from the two analyses, we can say that the main selection on document organization type has to be amongst C1, C2 and C3 configurations. Fig. 4 suggests that C1 or C2 should be the best configuration especially with machines that report various types of data. For example, a metal 3D printer may collect X-ray images after every processing layer along with data from its various sensors. These two different data types will require nested forms of document organization to ensure easy correlation. The C3 configuration, with the flat documents is best served for simple machines. Given the needs of storing large amounts of data, a sharded database makes more sense. Therefore, we recommend, C1, i.e. a timestamp based document structure storing data in a sharded manner through a hashing function for storing streaming data from machines.

4. A high-level architecture for an operating system for virtualized manufacturing machines

For anyone who has managed a large manufacturing shop floor can attest, multiple manufacturing machines are difficult to manage. When its digital counterpart, exists in cyberspace, managing them will be critical. Virtual instances of multiple machines will necessitate a need for an operating system that manages VMMs for machines on a shop-floor. This section provides a solution approach that may be able to address the networking of physical assets on a manufacturing floor to the highest enterprise level. The solution provides programmatic interfaces which abstract lower level functionality to help build tools for simulating, scheduling, monitoring and validating of processes for both small lot and large production manufacturing. The solution also possibly accounts for the heterogeneous physical and software assets deployed on a manufacturing floor. More importantly, the approach harnesses the significant data generation and consumption capability of manufacturing machines as the industrial world gears up to an era of smart manufacturing.

Similar to a traditional operating system which manages both hardware and software resources of a computer, an operating system needs to be built from the ground up to control the various physical and data assets available on a manufacturing shop floor. This operating system will also enable the rapid reconfiguration, simulation, scheduling of resources within the enterprise and its extension across the manufacturing eco-system. We call this operating system for cybermanufacturing as **'CyMOS'**. Besides allocation of compute resources and management of VMMs of various machines on the factory floor, CyMOS provides a programmatic interface for the data generated and consumed by the physical machine assets on a production floor. It integrates data from the lowest level of a manufacturing enterprise – Physical Machine Layer (Layer 1 in Fig. 5) to the highest level of the enterprise user through

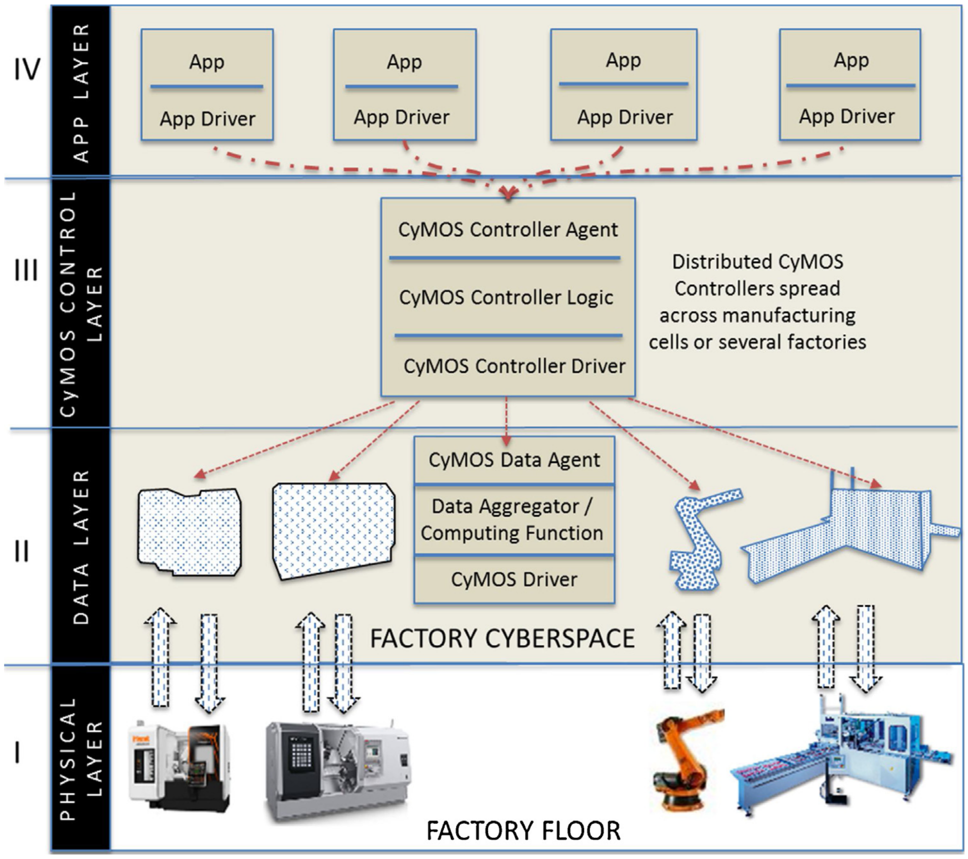


Fig. 5. Four Layers of the CyMOS architecture. Layers II–IV exist in cyberspace, while Layer 1 represents the physical machine assets and necessary adaptors to communicate with layers above it.

a set of middleware layers – Data Layer, Control layer, and the App Layer. We have briefly highlighted the function of each layer that comprises the CyMOS architecture.

Physical Machine Layer—This layer represents the bottom portion of the stack. It represents the edge from which a host of data is collected and to be controlled. This layer also represents the software and hardware adaptors attached to the machine that are necessary for its communication with the layers above it.

Data Layer—This layer represents the data communication from and to the physical machine asset. Each machine on the shop floor will have its individual virtualized state. This virtualization is enabled by a database instance structured to receive real-time streaming data from the machines. Drivers written at this data layer serves to communicate with the proprietary interfaces of the individual machines. Data analysis algorithms built in this data layer serve to aggregate and summarize information to be made available to higher level system calls. At the upper level of this layer, each virtualized state of the machine has an agent that communicates with the CyMOS control layer. This agent is responsible for translating instructions from the control layer and generating results for requests about the physical machine made available to any software app or device that requests it.

CyMOS Control Layer—At this level is essentially a hypervisor that controls the virtualized states of each machine on the production floor. It contains information about the virtual machines in its network, current status of machines and other information gleaned from external data sources (such as an ERP or MES system). At the bottom of this layer, is the necessary driver that communicates with each individual virtualized machine. At the middle of the stack are higher level computational algorithms that aggregate data from the virtual states. It is able to translate input requests from Apps that sit on top of the CyMOS architecture and then fetches valid data requests from the respective virtualized state. The CyMOS control layer can also interface with other external databases either within or outside the enterprise for processing instructions to the layers below it. Depending on the complexity of the factory-floor, there could be multiple distributed controllers spread across factory floor in an enterprise that spans multiple geographic locations.

App Layer—Apps written by any third-party developer on any preferred programming interface will be written to suit business level requirements. These decisions can range from tracking a product across multiple machines, assessing the carbon footprint of parts manufactured across machines or approved apps that sends specific instructions to change how a physical machine needs to operate. Numerous apps can be thought of that are suited for the manufacturing environment. The Apps need only to interface with the CyMOS control layer without the need for the developer to know means of communicating the specific input requests to the manufacturing machines below the layer. CyMOS handles the processing of requests from the app and handles necessary communication with established manufacturing execution systems. It must also communicate with the virtualized state of the machines. It may even have necessary communication interfaces with other CyMOS systems from another enterprise.

CyMOS can utilize existing industrial hardware for manufacturing, computing and factory floor networking. Work conducted through MT-CONNECT, OPC/UA and other industrial communication protocols allow communication from the machine assets to higher level software systems above it (interface between Level 1 and Level 2 in Fig. 5). Core CyMOS involves demonstrating the transfer of real-time streaming machine data and have it distributed across multiple computing systems – beginning with servers installed within the machine to the manufacturing cell to the factory and to higher level enterprise systems above it. This real-time streaming of data from the apps to the machines and vice versa enables manufacturers to maintain a real-time virtualized state of their physical machines. Programmatic interfaces allow data computation to be carried out on this virtualized state to solve a business problem, such as for process planning and scheduling a new variant of an existing part or automated tool path verification analysis. In addition, CyMOS can enable other third-party hardware devices such as Augmented Reality goggles and mobile computing devices to access information from the virtualized state of the machine in near real-time without overloading the machine controller. More importantly, because CyMOS is an operating system, it 'knows' the state of the machine and consequently the entire factory floor to allow job routing, job status monitoring and synchronization of work activities across manufacturing cells within factories and even geographically distributed factories. CyMOS system complements existing Manufacturing Execution Systems (MES) since it provides a digital counterpart for access to granular data on the machines and the processes conducted within the machine.

The API for the apps was developed in python with the intent of providing a friendly ontology to developers extending the availability of apps to interface with the machines. It is essential to ensure that the API satisfies the following requirements [29]:

1. Usability – The API should be designed with a user-centric approach. The end user in our case is taken to be an individual who is sufficiently knowledgeable in Object oriented programming and is comfortable programming in Python with some knowledge of the manufacturing process and machine involved.
2. Ease of programming – APIs should make a programmer's life easier and not make them perform tasks which are either cumbersome or repetitive or both. In our API, we have provided functions for accessing the VMM directly via the IP and the ports of the VMM. Also, we provide relevant "frequently used functions" such as moving average, standard deviation calculations etc. for direct calculations and for automated data entry to the summary layer.
3. Relevant abstraction – The API is also meant to hide irrelevant details from the end app developer. Pertinent to our case, it is irrelevant for the developer to know in advance the names of the parameters/class names for the machines' digital representations and/or the internal database names. The API provides a unified approach to calling the different layers of the VMMs directly and the API at its internal core will resolve the intent of the request.

Our API is implemented as a class *Machine* which can be initiated using the port and IP of the VMM we need to access. The VMMs for a physical machine are located at a particular server address and respective port. It does not need to be located at the machine itself but can be made available at an edge/cloud infrastructure. After initiation, some of the functionalities extended to the end user are shown in Table 3. Many such functions can be developed to abstract out details regarding how to fetch the data from the NoSQL data structure.

5. Applications of VMM in additive manufacturing

In this section, we describe two applications currently built on top of the proposed VMM middleware architecture. While we have built VMMs for other CNC machines [31], here we focus on two machines, a partially open sourced low-end fusion deposition based 3D printer (Makerbot 2X) and a high-end closed source metal based 3D printer (ARCAM series of EBM machines). Any number of applications can be built with the middleware architecture using programming language of choice, provided there is the necessary adaptor to interface with VMM generic access library. The code for the VMM architecture has been built in Python with extensive use of libraries available within the Python language ecosystem.

5.1. Apps for an open-sourced system – fused deposition modeling(FDM) 3D printers

This particular app written for the Makerbot is designed to capture machine axis movements and to rebuild the part digitally by collecting data from the controller via driver codes, the part file and any associated sensor data. We will be using this raw data to identify the parts printed on the Makerbot, then associate and visualize sensor readings with the slice layers of the part. This can be especially useful when there are fleets of FDM printers and the need for traceability on part validation and verification, particularly when process monitoring data is gathered. Moreover, by associating process information with product information, it becomes possible to identify differences at a slice level between products.

Here we have a part that was printed under a 0.1 mm resolution on a Makerbot printer (Fig. 6A). The machine was enabled to transfer the movement, temperature and sensor information to its VMM. The apps pulls this information from the database and plots the machine movement in a 3D scatter plot using the generic access library and the matplotlib library available in Python. A sample document stored in the database is shown in Fig. 6B. For clarity, we have reduced the point size of the scatter plot in Fig. 7. As seen, any external application can easily retrieve file names, information about the Makerbot motherboard status, build state, platform and tool temperatures along with the associated sensor values, all synchronized by a timestamp. This information is collected 10 times every second as the printing process runs. Using the app, an external user can easily identify that the part was built using an hexagonal fill (Fig. 7). Such reconstruction becomes extensively useful when reconstructing process steps for quality monitoring checks.

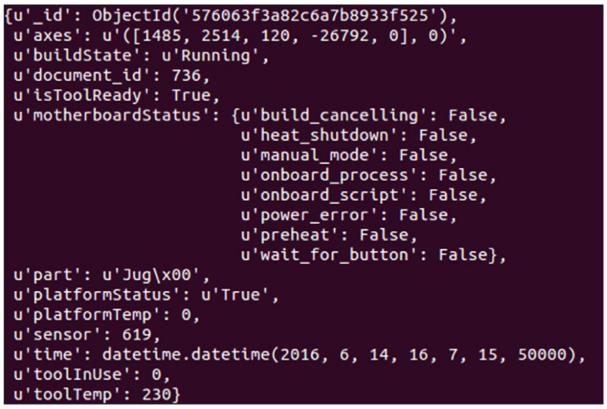
The summary layer of the machine (Fig. 1A) contains statistics for various parts including the mean, minimum and maximum values for the sensors. This is especially useful from a quality control perspective, if traceability is required to find process layer levels that exceeded a certain value. Such an analysis can be used for enhanced study of FDM printed parts and may also be used as a "playback" system for the machine. By storing the values of the encoders, it will be possible to make an app which can compare the commanded movements with the actual movement of the axes. Such a system can be used for complete quality control of 3D printed products for every single part produced. It can also be useful when security implications are considered. Advanced pattern recognition apps can be written to identify deviations of part fabrication from the norm to warn users of errors and potential malfunction. This can help mitigate cyber-threats with apps that specifically look for deviation from the accepted norm in part fabrication.

5.2. Apps for a closed system – an electron beam melting (EBM) 3D metal printers

Electron Beam Melting machines are far more complex since it is a closed system with its ability to generate data for thousands of parameters that are tracked during the part fabrication process. We demonstrate the use of our architecture for development of an app which can be used to facilitate research into development of analytical models and data visualization. In this app, we demonstrate the use of the generic access libraries to interface with custom machine specific log files for developing statistical and visualization tools. This is unique such that it allows researchers to go beyond the basic

Table 3
Sample API functions made available a VMM app developer.

Function name	Functionality
getMachineInfo(layer)	Gets all the records associated with a particular data layer required by the user
getParameters(layer)	Returns all parameter names which are being stored by a particular layer of the VMM
pushToSummary(parList)	Pushes calculated parameter values such as means, functions etc. as required by a user to the summary layer of the VMM
getParameterData(parameter,layer,associateData = "None")	Returns the data associated by a particular parameter and an optional associated data point from the same time stamp (if needed, none by default).



```
{u'_id': ObjectId('576063f3a82c6a7b8933f525'),
 u'axes': u'([1485, 2514, 120, -26792, 0], 0)',
 u'buildState': u'Running',
 u'document_id': 736,
 u'isToolReady': True,
 u'motherboardStatus': {u'build_cancelling': False,
                        u'heat_shutdown': False,
                        u'manual_mode': False,
                        u'onboard_process': False,
                        u'onboard_script': False,
                        u'power_error': False,
                        u'preheat': False,
                        u'wait_for_button': False},
 u'part': u'Jug\x00',
 u'platformStatus': u'True',
 u'platformTemp': 0,
 u'sensor': 619,
 u'time': datetime.datetime(2016, 6, 14, 16, 7, 15, 50000),
 u'toolInUse': 0,
 u'toolTemp': 230}
```

Fig. 6. Actual Part fabricated by the MakerBot and its associated sample document structure stored in the MongoDB.

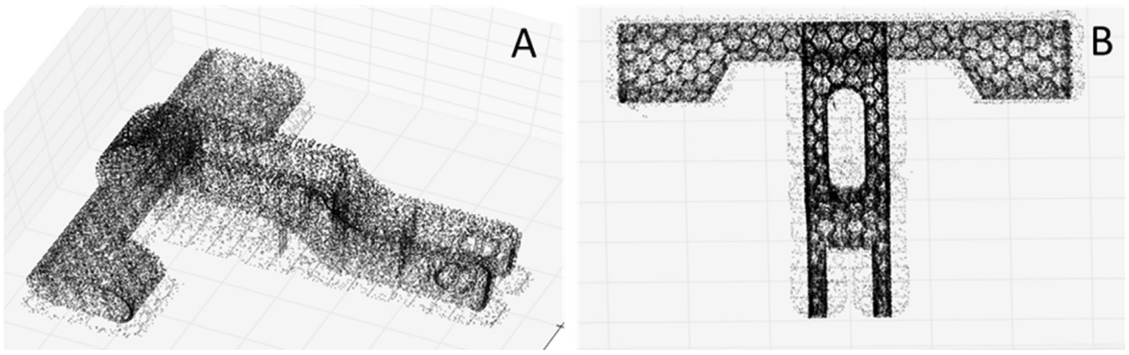


Fig. 7. ISO(A) and TOP(B) Views of Digitally Reconstructed Parts from Machine Axis Data.

tools offered by the vendor (ARCAM) in their custom machine software. A limitation of the machine software is the inability to analyze multiple parts at the same time. As such, it becomes very difficult to compare multiple parts and process features simultaneously without writing excessive code to compare data across multiple log files. This is particularly important when file sizes in excess of 300MB and even more than a TB can be easily generated during the fabrication of a single part [32].

As of this writing, the ARCAM EBM machine did not support a standardized communication protocol like MTConnect. However, the data generated during the fabrication process is made available in the form of the vendor's own format available as log files generated during the fabrication process. At the end of each build, the system makes available a log file in the form of a compressed text file (in a format called ".plg"), which can be unzipped and read by a custom driver written for the machine. The generic format of the text file is as follows:

# timestamp|Parameter|Parameter superset|Machine Internal Code|Parameter Value

An example line item within the log file would read as follows:

"# 2016-05-18 11:18:25.453|Alarms.BeamControl RecoverableError|Core|26111714|NotSet"

A driver was written for the machine which reads the log file line by line, parses the text and generates JSON documents and is stored within MongoDB and made available through the VMM for the machine. A typical log file for an 8 h build can be as large as 2 GB. This can potentially reach 100GB, depending on additional sensors or image data that is captured during the process. The power of NoSQL document structures is that the data streams can vary, yet the MongoDB initial data organization need not vary. The custom driver written using our computer server (8GB RAM, Intel i3 processor, 3.4 GHz) took about an hour to process the log file and then transmitted to the database storage cluster's using the university's 100Gb/s network speeds. These speeds can certainly be improved with multi-threading features, solid-state drives and higher processor speeds.

This example app requires the user to select a parameter and it then fits a curve to the data collected during the part fabrication (Fig. 8A). As an example, we programmed an app to report back on the time for melting of a particular layer as the build proceeds for

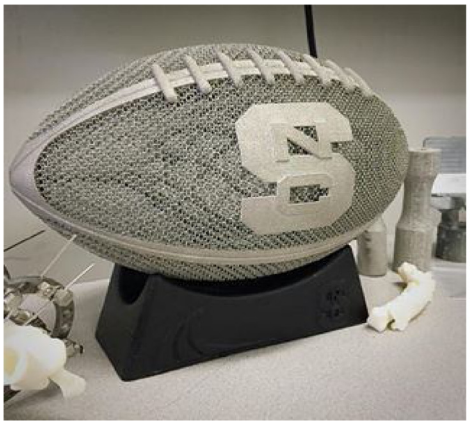
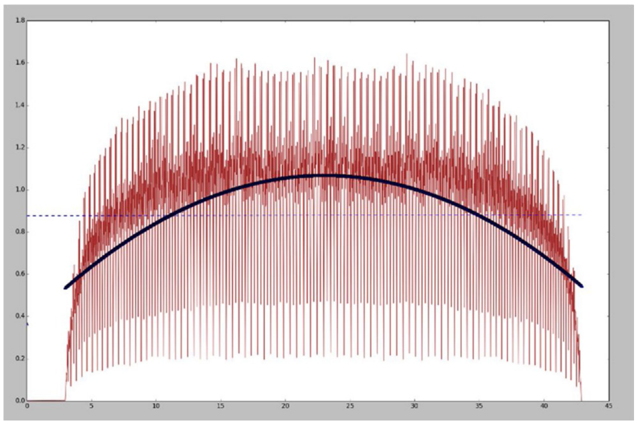



Fig. 8. (A) Porous Mesh Football built on the ARCAM EBM machine. (B) Contour Time vs Build Time. Due to the shape of the football, it is expected that melting time to create the contour is high in the center.

the fabrication of the metal football (Fig. 8B). The app fetched data from the corresponding VMM of the machine without interfacing with the machine log files. An app programmer need only interface through the GMAL to access all data with regards to the fabrication of the part.

6. Discussion

Streaming data from machines comprise an important feature that enriches CPS in manufacturing. While many machine to machine communication protocols and standards are in development across the discrete and continuous manufacturing space, the architecture must adapt to making it interoperable as much as possible. We have followed an architecture stack similar to the software architecture in smartphones (iOS and Android) and the Software Defined Networking (SDN) built to communicate over the internet [33]. Constructing stacked software layers is critical to the wide scale adoption of CPS systems in manufacturing and would make third-party software development easier.

In the first section, we have proposed a stacked architecture for the virtualization of manufacturing machines. The VMMs for a manufacturing machine record every data stream both into and out of the machine. This virtual twin allows external systems to interact with the machine without overloading the physical machine. For example, applications can be built to autonomously check if a machine is capable of fabricating a part with the right set of cutting tools or size limitations. A VMM will have near real-time status of a physical machine, thereby adding a layer of security to any external IT systems that requests or sends information to the machine. The VMM is also built to provide a layer of hardware level abstraction to higher order applications. Similar to the Android operating system, which adds a layer of hardware abstraction to the third-party software developers, the proposed VMM architecture can allow third-party developers to build software apps that transcend any machine type. The intent of the VMM is not to replace a controller for the machine but rather to complement its capabilities and have a silent system that listens and records events carried out by the machine. While we have proposed an initial architecture, it will require significant amount of work to build an entire ecosystem of hardware drivers and adaptors accompanied by an extensive generic access library to cover all the tools required. Machine vendors will need to be actively involved in writing software adaptors for their machines to help fully implement the cyber-physical manufacturing system architecture.

We have studied the use of unstructured data schemas to store data streaming in from physical machines using MongoDB, a document based flexible schema type data store. Unlike traditional data structures, unstructured data schemas allow faster adoption across the variety of machines with various data inputs and outputs. The manner in which this data is organized depends on how this data needs to be served. Unlike financial transactions, manufacturing applications are heavily write dependent and less on frequent read operations. Hence, NoSQL unstructured schemas must be optimized for best write performance of data that is scalable across various machines types. We have studied timestamp based collection of data to organize the content with options available for embedded documents. Our analysis of storing log files indicate that timestamp based embedded document organization provides the best performance. However, the optimal size of each document must still be studied. We expect this size to be dependent on the machine type. A natural extension to this work is the necessary information modeling that must be done once the data has been stored and captured within the data store. Adequately defining the requirements, constraints, rules and relationships between various components of the machine would be critical to offers levels of abstraction to third-party software developers. The library of API level functions implemented in GMAL would be further extended to offer the level of usability depending on the user type (such as an app developer/end-user) and manufacturing context.

Further, questions arise on where this data should be stored, whether at the machine itself, at the factory level or at the cloud level. Latest fog computing protocols that bridge both edge and cloud computing infrastructure make it extremely attractive to store portions of this data across the cyberinfrastructure [34]. Cloud computing in manufacturing is seen as a vital technology to expand the use of computational resources beyond the capacity of individual manufacturers [35]. VMMs can be a critical piece of technology to enable the data analytics that can be performed on large sets of historic data critical for machine health monitoring and maintenance applications. More work needs to be done to explore manufacturing data organization and how best it must be served to client applications that request this data with value of this data to be of top concern. Exploring technologies that couple MongoDB with latest data mining infrastructures such as Apache Spark or Hadoop [36] also increases the potential of VMM in the cybermanufacturing space.

With the networking of manufacturing machines and the addition of sensors to monitor the process becoming more prevalent, we argue that simply storing the data generated during the process in the form of flat log files or rudimentary databases will make the retrieval of this data limited to single part files. If further production scale analysis is desired, then custom software will have to be written to parse through multiple log files to extract needed information for statistical analysis or part validation exercises. More importantly, to assess the health of the machine, custom software will be required to analyze the process data locked into the multiple log files generated between scheduled maintenance interval times. The data to be analyzed will be in the multiple terabyte data block sizes, requiring facilities to manage computing hardware to crunch and analyze through the data. In some instances, production scale AM machines archive the data or portions of it are deleted after a production run is completed. For example, when metal additive manufacturing machines are coupled with x-ray image capture technology, data can easily span terabytes. This image based data is deleted at the end of a fabrication resulting in valuable data being lost. Therefore, we do require efficient methods of storing unstructured data to help analyze and compute across the data generated during the part fabrication process.

If each machine on a shop-floor has its virtual counterpart, multiply this across multiple factories for an enterprise, then this will necessitate an operating system for all of the virtual manufacturing machines. An operating system for such virtualized manufacturing machines becomes necessary to manage the various virtual resources pertaining to the manufacturing operation. Such an operating system would direct flow of instructions from the cybermanufacturing layer down to the right machine. This operating system complements existing MES/ERP systems in that it only manages data flow between the CPS architecture levels. Our architecture proposes a high-level four-layer architecture that merge the cyber layers on top down to the physical machines at the bottom layer. The interfaces across each of these levels is critical. Much of the work in the scientific community is focused on activity between Levels 1 (physical layer) and Level-2 (Data layer). We see the need to develop interface protocols between VMMs at Level 2 and Level-3 (Control layer), particularly when machines of various types need to interact. The control layer must encapsulate verification/validation algorithms and communication handling protocols. This work did not utilize the Control layer since we have not connected multiple VMMs together for any coordinated control action. More work is yet to be done in this space and will require significant community effort to build an operating system that spans various factory systems.

7. Conclusion

The VMM paradigm presented here focuses on a stacked architecture for ease of implementation on the shop-floor while maintaining interoperability and scalability. Rather than forcing third-party software developers to deal directly with the intricacies of physical machine infrastructure, the stacked VMM platform handles lower level functionality and allows developers to build app instances using API libraries. By doing so, VMM enables broader ease of implementation into a distributed manufacturing problem, with two foci area – at the highest level, developers can focus on building applications for the digital thread, while at the lower level, machine developers can focus on building drivers that interface with the VMM middleware.

We focused this paper on two aspects. First, we studied the use of a NoSQL schema-less architecture to implement VMM since its data store can accommodate any type of machine data with very little modification to the document structure. Addition and removal of data is seamless and does not require significant changes to the database structure, therefore making it easier to implement across the variety of machine types on a factory floor. Second, we make the case for the necessity for a control platform, which we term as CyMOS, to help translate and mediate between higher level applications down to the various VMM for each machine. We have showcased two such examples of building apps on 3D printer type machines. This concept of building apps through a middleware can be extended to any type of machine. There is still much work to be done both in building extensive API libraries at the VMM and CyMOS end. This enables the broader involvement of the hardware and software community in building manufacturing apps.

Acknowledgments

We thank the funding agencies who have supported the initial idea and related work, NSF #1547105 and DMDII DMC 16-08.

References

[1] Vijayaraghavan A, Huet L, Dornfeld DA, Sobel W, Blomquist B, Conley M. Addressing process planning and verification issues with MTConnect. Trans N Am Manuf Res Instit SME 2009:557–64.
[2] Hannelius T, Salmenpera M, Kuikka S. Roadmap to adopting OPC UA. 6th IEEE international conference on industrial informatics 2008:756–61.
[3] Hunkeler U, Truong HL, Stanford-Clark A. MQTT-S—a publish/subscribe protocol for wireless sensor networks. 3rd international conference on communication systems software and middleware 2008:791–8.
[4] Kim DB, Witherell P, Lipman R, Feng SC. Streamlining the additive manufacturing digital spectrum: a systems approach. Addit Manuf 2015;5:20–30.
[5] Gao R, Wang L, Teti R, Dornfeld D, Kumara S, Mori M, et al. Cloud-enabled prognosis for manufacturing. CIRP Ann Manuf Technol 2015;64(2):749–72.
[6] Moyne J, Iskandar J. Big data analytics for smart manufacturing: case studies in semiconductor manufacturing. Processes 2017;5:39, http://dx.doi.org/10.3390/pr5030039.
[7] Chen H, Chiang RHL, Storey V. Business intelligence and analytics: from big data to big impact. MIS Q 2012;4(36):1165–88.
[8] Wang G, Gunasekaran A, Ngai E, Papadopoulos T. Big data analytics in logistics and supply chain management: certain investigations for research and applications. Int J Prod Econ 2016;176:98–110.
[9] Lee J, Bagheri B, Kao HA. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manuf Lett 2015;3:18–23.
[10] Wang L, Törngren M, Onori M. Current status and advancement of cyber-physical systems in manufacturing. J Manuf Syst 2015;37(2):517–27.
[11] Wong TN, Leung CW, Mak KL, Fung RY. Dynamic shop-floor scheduling in multi-agent manufacturing systems. Expert Syst Appl 2016;31(3):486–94.
[12] Sikora R, Shaw MJ. Coordination mechanisms for multi-agent manufacturing systems: applications to integrated manufacturing scheduling. IEEE Trans Eng Manag 1997;44(2):175–87.
[13] Maffei A, Onori M. Evolvable production systems: environment for new business models. Key Eng Mater 2011;467:1592–7.
[14] Ferreira P, Lohse N, Ratchev S. Multi-agent architecture for reconfiguration of precision modular assembly systems. Precision assembly technologies and systems, international precision assembly seminar 2010;315:247–54. S. Ratchev (eds.).
[15] Nassehi A, Newman ST, Allen RD. STEP-NC compliant process planning as an enabler for adaptive global manufacturing. Robot Comput-Integr Manuf 2006;22(5):456–67.
[16] Hu H, Yu L, Wo Tsui P, Zhou Q. Internet-based robotic systems for teleoperation. Assem Autom 2001;21(2):143–52.
[17] Xu X. From cloud computing to cloud manufacturing. Robot Comput-Integr Manuf 2012;28(1):75–86.
[18] Wang L. An overview of internet-enabled cloud-based cyber manufacturing. Trans Instit Meas Control 2017;39(4):388–97.
[19] Kang YS, Park IH, Youm S. Performance prediction of a MongoDB-based traceability system in smart factory supply chains. Sensors 2016;16(12):2126.
[20] Li T, Liu Y, Tian Y, Shen S, Mao W. A storage solution for massive iot data based on nosql. IEEE Int Conf Green Comput Commun 2012;5:0–57.
[21] Van der Veen JS, Van der Waaij B, Meijer RJ. Sensor data storage performance: SQL or NoSQL, physical or virtual. Cloud computing (CLOUD), IEEE 5th international conference on cloud computing 2012:431–8.
[22] Le TD, Kim SH, Nguyen MH, Kim D, Shin SY, Lee KE, et al. EPC information services with No-SQL datastore for the internet of things. IEEE Int Conf RFID 2014;4:47–54.
[23] Boicea A, Radulescu F, Agapin LI. MongoDB vs oracle-database comparison. EIDWT 2012;33:330–5.
[24] Liu Y, Wang Y, Jin Y. Research on the improvement of MongoDB auto-sharding in cloud environment. IEEE in computer science & education (ICCSE) 2012:851–4.
[25] Dima A, Bhaskarla S, Becker C, Brady M, Campbell C, Dessauw P, et al. Informatics infrastructure for the materials genome initiative. JOM 2016;68(8):2053–64.
[26] Volatile Data Stream (VDS), https.//smstestbed.nist.gov/vds/current http://www.dx.doi.org/10.18434/.
[27] Helu Moneer, Hedberg Thomas. Enabling smart manufacturing research and development using a product lifecycle test bed. Procedia Manuf 2015;1:86–97.

[28] Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, 2002. ACM SIGACT News 2001;33(2):51–9.
[29] Gilbert S, Lynch N. Perspectives on the CAP theorem. Computer 2012;45(2):30–6.
[30] Scheller T, Kuhn E. Automated measurement of API usability. J Inform Software Technol Arch 2015;61(C):145–62.
[31] Singh S, Angrish A, Starly B, Barkley J, Lee YS, Cohen P. Streaming machine generated data to enable a third-party ecosystem of digital manufacturing apps. Procedia Manuf 2017;10:1020–30.
[32] Dunbar AJ, Nassar AR, Reutzel EW, Blecher J. A real-time communication architecture for metal power bed fusion additive manufacturing. In: Proceedings of the solid freeform fabrication symposium. 2016. p. 67–80.
[33] Haleplidis E, Pentikousis K, Denazis S, Salim JH, Meyer D, Koufopavlou O. Software-defined networking (SDN): layers and architecture terminology, RFC 7426, IETF 2015. ISSN 2070-1721, January 2015.
[34] Wu D, Liu S, Zhang L, Terpenny J, Gao RX, Kurfess T, et al. A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing. J Manuf Syst 2017;43(1):25–34.
[35] Wu D, Jennings C, Terpenny J, Kumara S. Cloud-based machine learning for predictive analytics: prediction of tool wear. In: IEEE international conference on big data. 2016.
[36] Yang C, Yen C, Tan C, Madden SR. Osprey: implementing mapreduce-style fault tolerance in a shared-nothing distributed database. Data engineering (ICDE), 26th international conference on data engineering 2010:657–68.