An Application of MongoDB to Enterprise System Manipulating Enormous Data

Tsukasa Kudo†, Yuki Ito†, and Yuki Serizawa†

†Faculty of Comprehensive Informatics, Shizuoka Institute of Science and Technology, Japan
kudo@cs.sist.ac.jp

Abstract - With the spread of the IoT, a variety of sensor data have been widely used, such as the image data of surveillance cameras and so on. And, such a system operations are spreading, in which the image data is saved in the database directly. Here, the relational database management system has a problem about the efficiency to manipulate the enormous unstructured data such as images. So, it is spreading that the system manipulating such a data is implemented by using the NoSQL databases such as the MongoDB with GridFS interface. However, as for the enterprise system, since the ACID properties of the transaction cannot be maintained in MongoDB, there has been the problem about its application to such a system. On the other hand, in our previous study, we had implemented the transaction feature for MongoDB to maintain all the ACID properties. Therefore, it is expected that MongoDB can be applied to such a system by using this transaction feature. That is, the advantage of MongoDB can also be used in the enterprise systems. In this paper, we show the application case of MongoDB to the production management system, which is a kind of enterprise system; also, we show the image management for the stocktaking as the case of the efficiency improvement by using MongoDB.

Keywords: database, transaction processing, ACID properties, MongoDB, GridFS, production management system

1 INTRODUCTION

With the spread of the IoT, networking sensors are collecting various types of data including the semi-structured and unstructured data such as images, audio and videos（hereinafter, “image and video”) [5], [8], [13]. Since these data is stored into the database and shared, it has become necessary that the database management systems adopt the feature called 3V, that is, Volume (huge amount), Velocity (speed) and Variety (wide diversity) [11]. Here, the relational database management systems (RDBMS) was designed for handling structured data, so it has limitations when it comes to manage the enormous and numerous unstructured data like files [19]. Thus, the various database management systems called NoSQL database, which is different from the conventional RDBMS, have been proposed and put to practical use [20]. MongoDB is a kind of NoSQL database and equips GridFS interface [1], by which various types of enormous data are stored efficiently by dividing into units called chunk, and it has been proposed and evaluated to apply to the above-mentioned fields [19], [26].

By the way, our laboratory is supporting the introduction of the production management system for the actual company. At its factory, the products are manufactured by the order-made, and their parts are replenished in lot unit only when they are insufficient. However, since the type of parts are so many in this factory, their accurate inventories often cannot be grasped. As a result, it has become the factor causing the shortage of the parts.

For this problem, we converted the idea from the inventory management by counting the actual quantity. That is, the human vision can grasp the approximate number of the actual inventory efficiently. So, we conceived that if the necessary inventory quantity is designated, the worker can find that the inventory is insufficient by looking at the inventory shelf. And, MongoDB can be applied to treat the image and video data that is the evidence of his judgment. Here, in this production management system, it is also necessary to manage the theoretical inventory, which is the quantity of the parts inventory managed in the system. So, in the case of moving 10 parts from the parts shelf to the assembly field, 10 is subtract from the former *collections* (corresponding the table in RDBMS) of the database; the same number is added to the latter *collections*. And, to maintain the consistency, these two data operations must be performed as a single transaction.

However, as for MongoDB, the ACID properties are maintained only in the case of updating a single data. So, in the case of updating the plural data, the data is updated one after another, and finally they become to be updated. That is, there is the anomaly in the midst of this updating, such as one data has been updated and another has not been updated; and, it can be queried by the other transactions. So, we could not find the application case of GridFS interface of MongoDB to the enterprise systems, though it provides the useful feature for the enormous data manipulations.

On the other hand, as for this problem, we showed the method of transaction feature for MongoDB for the centralized database environment [10]. By this method, all the ACID properties, which is consist of the atomicity, isolation, consistency, and durability, can be maintained throughout in each transaction. Moreover, this production management system can be built with the centralized database environment.

Therefore, we decided to apply MongoDB equipping this transaction feature to this production management system, and implemented its prototype. As a result, we confirmed that MongoDB could be applied to the enterprise system by using our transaction feature. Moreover, we confirmed the image and video data management by using MongoDB could improve the efficiency of inventory management works.

The remainder of this paper is organized as follows. Section 2 shows our transaction feature and the problem of the target production management system. And, we show the application method of MongoDB to the enterprise system in Section 3. Section 4 shows the implementation of this method and its evaluation results. We discuss on the results in Section

5, and Section 6 concludes this paper.

2 RELATED WORKS AND PROBLEMS

2.1 Transaction Feature for MongoDB

MongoDB is a document-oriented NoSQL database, and its data is stored as a *document* of the JSON (JavaScript Object Notation) format as shown in Fig. 1 [14], [25]. Its *document* is composed of the fields, and {"_id": Id1 } expresses the field having the identifier "_id " and value "Id1". Here, "_id" indicates the ObjectID that corresponds to the primary key of the relational databases. In Fig. 1, the other fields of the *document* are "name" and "address". Here, field "name" has a nested structure, which is composed of field "first" name and "last" name. Since MongoDB has such a structure, it is not necessary to define the scheme of the database beforehand like RDBMS. That is, the fields of each *document* can be added or removed at any time. Incidentally, the set of *documents* composes the *collection*, and each of them corresponds respectively to the records and table in the relational database although not strictly. Thus, each *collection* can have *documents* of various structures.

In addition, similar to SQL of the RDBMS, CRUD (Create, Read, Update, Delete) data manipulation is provided. However, since its transaction feature is based on the eventual consistency of the BASE properties [3], [18], the ACID properties can be maintained as for only the single *document*. Therefore, there is the problem that the ACID properties of transactions cannot be maintained in the case of updating multiple *documents* simultaneously. Incidentally, the ACID properties are defined as the following 4 properties: Atomicity means the transaction updates completely or not at all; Consistency means the consistency of database is maintained after it is updated; Isolation means each transaction is executed without effect on the other transactions executing concurrently; Durability means the update results survive the failure [7].

For example, in the case of the bank account transfer from account A to account B, the total amount of the both accounts does not change. However, as shown in (a) of Fig. 2, since the ACID properties are not maintained on the entire updating in MongoDB, there is the problem of the anomaly. That is, the halfway state during the updating is queried by the other transactions: one data has been already updated; another data has not been updated yet. In this case, when the account A was updated, the anomaly that the sum of the query result was reduced to 2,000 temporarily happened. Also, since the rollback must be executed by the compensation transaction in the case of failure [22], it has been shown that the isolation on these documents cannot be maintained by this method [12]. That is, the isolation cannot be maintained in the both cases of the commit and rollback. On the other hand, in the case where the same procedure was performed in the RDBMS, this halfway state can be concealed until the commit as shown in (b) of Fig. 2.

Furthermore, as for SQL of the RDBMS, the isolation levels of the transactions are defined. That is, corresponding with the business requirement, the suitable isolation level can be selected: in the case where it is needs the efficient execution,

```
{ "_id" : Id1, "name" : { "first" : "Tsukasa", "last" : "Kudo" },
                "address" : "Hukuroi-shi"}
```

Figure 1: An example of MongoDB document.

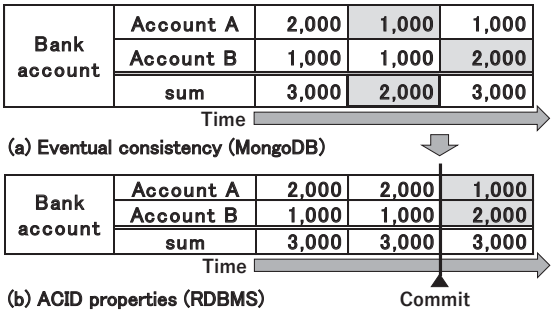


Figure 2: Problem of transaction processing in MongoDB

or the strict concurrency control [7]. So, we had implemented and evaluated the method to perform each transaction with the designated isolation level in MongoDB in the previous study, as well as the RDBMS as shown in Table 1. As a result, we confirmed the following: the isolation levels of Table 1 were achieved, and the query transaction performance at READ UNCOMMITTED is same as MongoDB [10].

We show the overview of this method below. First, it performs the lock operation during the access to the *document* as well as the RDBMS as shown in Table 1. In Table 1, "2PL" shows the two phase locking protocol [7]. Incidentally, in this method, to prevent the cascade abort of the transaction, the rigorous 2PL is adopted as well as the RDBMS. That is, the lock is held until the commit or rollback.

Second, as shown in Fig. 3, the *document* of the *Data collection* saves the business data into two fields, "Data before update" and "Data after update". While the *document* is not being updated, the business data is saved in the former; and, there is not the field of the latter. On the contrary, while the *document* is being updated, the business data of two state is saved: the state of before update is saved in the former; and, the state of after update is saved in the latter. In addition, it has the fields to save the information of the transactions locking it: for each of the shared lock and the exclusive lock. And, in order to manage the transactions that are locking the *document* of *Data collection*, we implemented *TP (transaction processing management) collection*. And, its *document* saves the corresponding transaction state: before or after the commit. Also, it saves the isolation level of the transaction.

For the case of Fig. 2, we show the procedure to query the *document* of *Data collection* while it is being updated with the isolation level READ COMMITTED or REPEATABLE READ. If the bank account transfer from the account A to ac-

Table 1: Locking protocol of each isolation level

Isolation level	Exclusive lock	Shared lock
READ UNCOMMITTED	2PL	(none)
READ COMMITTED	2PL	While query
REPEATABLE READ	2PL	2PL

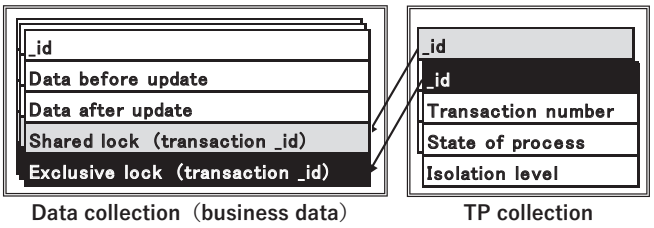


Figure 3: Transaction processing method for MongoDB

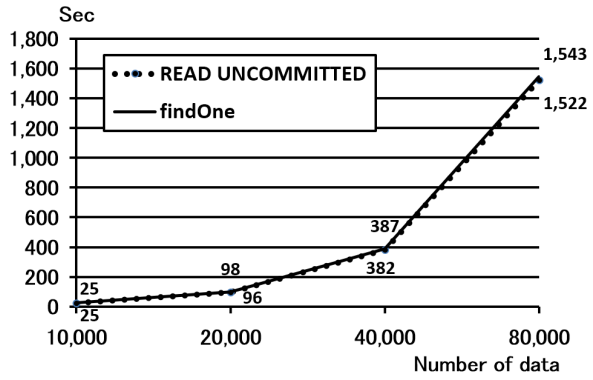


Figure 4: Comparative evaluation with findOne of MongoDB

count B shown in (a) of Fig. 2 is executed by this method, then both of the data before and after update are retained. And, the data before update is queried until the commit of the transaction; the data after update is queried after the commit. Therefore, in (a), both update results of the account A and B are not queried until the commit; both update results are queried only after the commit. That is, the query results are similar to the RDBMS shown in (b). Thus, the halfway state during the updating is concealed from the other transactions, and the transaction processing maintaining the isolation can be provided.

Similarly, other properties of the ACID propertied can be also maintained by this method. Basically, as for the update of individual *document*, the ACID properties are maintained by the transaction feature of MongoDB itself. And, as for the atomicity, the data before update is saved in "Data before update" field. Therefore, when the failure occurs while updating plural data, the rollback of all the *documents* can be performed by deleting their "Data after update" fields. As a result, since the rollback can be performed without using the compensation transactions, the isolation property can be maintained. Incidentally, though the compensation transactions is usually used for the rollback in MongoDB [22], it cannot maintain the isolation. On the contrary, in the case where this update completes normally, their commit can be performed by changing "Data after update" to "Data before update". Then, the former is deleted. As for the consistency, it also can be maintained by this rollback when the consistency is not maintained during updating plural *documents*.

As for the durability in the event of a crash, it is provided by the original transaction feature of MongoDB, which uses write ahead logging to an on-disk journal [16]. And, since this method uses this transaction feature, the durability can be maintained in this method, too.

On the other hand, the Velocity (speed) of the 3V feature, that is, high efficiency for the query of the NoSQL databases is generally required. So, we performed the comparative evaluation on the query processing between this method and MongoDB. As for the former, the query transaction was performed with the isolation level READ UNCOMMITTED, in which the query was performed efficiently without the shared lock as shown in Table 1. As for the latter, we used "findOne" method, which was the standard query method of MongoDB. As a result, we found that the performance of the both are almost the same as shown in Fig. 4 [10].

Here, it has been shown that if all the transactions are performed with any of the isolation level shown in Table 1, then any transaction can be performed with the designated transaction level [7]. In other words, by using this method, the usual query transactions can be performed efficiently with the isolation level READ UNCOMMITTED as in the conventional data manipulation of MongoDB; only the query and update transactions, which need the strict data manipulations as shown in Fig. 2, can be performed with the isolation level READ COMMITTED or REPEATABLE READ.

2.2 Enormous Data Manipulation Feature in Databases

As for the enormous data manipulation, the RDBMS has some problem: there is the limitation of the data size, and it must be read sequentially. We show this in detail as follows.

As for the relational databases, there had been the request to treat the various type of enormous data including the image and video. So, in SQL:1999, known as SQL3, the data type LOB (LARGE OBJECT) have been defined. It is composed of the data type CLOB (CHARACTER LARGE OBJECT) and BLOB (BINARY LARGE OBJECT): the former treats the character strings; the latter treats the binary data including the image, video and so on [6]. However, it has been shown: since RDBMS was designed for handling structured data, it has limitation to manage the enormous unstructured data [26], [19]. That is, the enormous unstructured data in binary-based column increases the demand for hardware resources, and the distributed systems to reduce this problem tend to be rigid and hard to administrator. And, it has been shown that using RDBMS for managing the enormous unstructured data is inefficient, because it is due to the architecture for the concurrency control by using locking, logging and so on [27], [28].

In addition, for example, MySQL (MySQL 5.7) also supports the BLOB type, and the data up to 4GB can be stored. However, as for the BLOB type, some restrictions have been shown [17]. For example, VARBINARY type is recommended in the case of the small size of data (up to 64KB); the column of BLOB type should be separated to another table for the sake of query processing. And, the data size transferred between the server and client, or among the servers, is limited up to 1GB. So, in the case of using the replication feature, the data size must be up to 1GB; in the case of storing the data of more than 1GB to the database, the data must be divided and stored sequentially. Furthermore, even in the case of querying a portion of the enormous BLOB type data, it is necessary to read sequentially from the beginning.

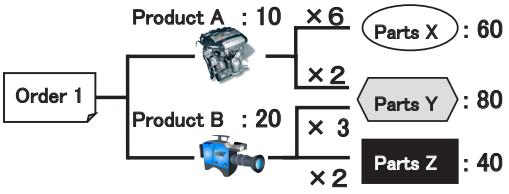


Figure 5: Composition of MRP system

Besides this, several methods are provided to save the unstructured enormous data such as images and videos. For example, Sears et al. showed that the file system has the clear advantage in the case where the data amount was so large [21]. However, in the case where the business system is constructed by the file system, it is pointed out that there are problems: they lack the efficient mechanism for data integration, security, backup and recovery, and so on; the configuration of application software becomes complicated [26], [23].

On the other hand, as for MongoDB, GridFS interface was prescribed, which is the convention to store the enormous data, and the official drivers support this [14]. In this convention, the enormous *document* is divided into chunks as a separated *document*. By using the GridFS, the data which size exceeds the file system of the OS can be manipulated, and the replication is also supported. In addition, as for even the binary data, a portion of the data can be queried. Because of this advantage, MongoDB has been proposed and evaluated in several systems dealing with enormous unstructured data such as images and videos: medical images for health care system, documents for e-learning system and so on [26], [19].

2.3 Target Production Management Business

Our laboratory has been supporting the production management system of a manufacturing company: the implementation, introduction and support of the system operations. Previously, we had introduced the MRP (Material Requirement Planning) system to automate the calculation of the quantity and cost of the parts, which is necessary to assemble the ordered products. We show the over view of the MRP system in Fig. 5. In this case, 2 parts Y is used for product A; 3 for Product B. So, in the case of order 1, which is composed of 10 products A and 20 products B, 80 parts Y is necessary. The cost of the parts, which calculated by this system, is used as the master data of the system in conjunction with the order company by EDI (Electronic Data Interchange) [9].

Now, we have been asked to introduce the inventory management system. The inventory control is the important function of the production management, and it aims to maintain the inventory quantity at the proper levels. In other words, the inventory levels of all the parts should be controlled such that the following can be achieved: the quantities of each parts are always more than the safety inventory, by which some problems can be dealt with to prevent the parts shortage; on the other hand, there should not be too much excess inventory, which causes the increase of the production cost. Here, this company manufactures the products by the order made, and the parts are replenished in lot unit only when they are short.

We show the inventory and flow of parts i in the factory

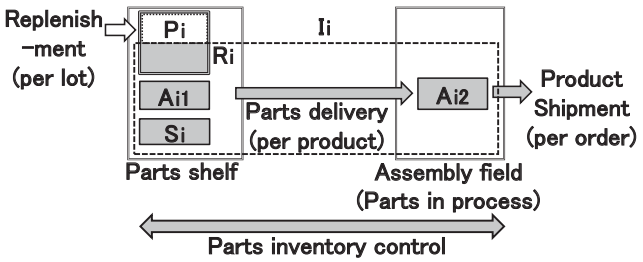


Figure 6: Product manufacturing process

in Fig. 6. Unused parts are kept in "Parts shelf", then they are delivered to "Assembly field" by "Parts delivery" to manufacture the products. And the finished products are shipped for each order. Here, I_i is the inventory quantity, which is indicated by the dashed box. And, S_i is the safety inventory quantity included in it. Similarly, A_{i1} is the already assigned inventory quantity to the other ordered products in "Parts shelf"; A_{i2} is the one in "Assembly field", that is, the parts are in process. When the factory receives the new order, the necessary parts quantity R_i is calculated by the MRP system. Then, insufficient quantity P_i is replenished in lot unit. P_i and the assigned total quantity A_i are expressed as follows.

$$P_i = R_i - I_i + A_i + S_i \quad (1)$$

$$A_i = A_{i1} + A_{i2} \quad (2)$$

For example, in the case of Fig. 5, as for parts Y, if R_Y is 80, I_Y is 50, A_Y is 30 and S_Y is 10, then the production quantity P_Y becomes 70. In the factory, replenishment is done in lot units as above-mentioned. So, there is often surplus, and $P_i = 0$ in this case.

Here, it is necessary to grasp the accurate inventory quantity to determine P_i. However, it is not easy in the actual factory. The types of the parts are several hundreds, and the parts shelf are dispersed in various places of the factory to adapt to the individual work process. Figure 7 shows the parts shelf examples as for the long parts and small parts. The long parts have to be counted from a particular direction. And, the small parts are stored in containers. So, it is necessary to take out them in order to count the exact quantities. In this way, it takes time to move among the parts shelf and to investigate the quantities. Actually, it was estimated to take a few man-days for the stocktaking of all the parts. Moreover, the parts are always moving from the parts shelves to the assembly fields, so the actual inventory quantities of parts shelves are always changing, too.

Incidentally, in the field of the production management in the large companies, the large scale production management system is introduced, such as the SAP [4], and the production information is managed as the integrated system including the inventory, accounting, order and so on. Also, the inventory quantity is sometimes measured by using the RFID (radio frequency identifier) tags in the various field to reduce the inventory investigation workload [2].

However, the target factory is the small or medium-sized company like most companies in Japan, of which proportion is said 99.7% [24], and it is pointed out that the introduction of such a management system is so less than the large company.

(a) Examples of storage of long parts

(b) Examples of storage of small parts

Figure 7: Parts shelf in target factory

As for this cause, two factors can be pointed out from the view point of their production scale. First, it is difficult to obtain the effect commensurate with the system investment such as the RFID and so on. Second, it is difficult to reserve the full-time personnel for the system operations, grasping the field data and entering it into the system. However, with the development of the e-commerce and supply chain management (SCM), it is becoming necessary to introduce the EDI with the large companies. Therefore, it is also becoming necessary to introduce the production management system to manage the data for the EDI. And, the inexpensive packages of the production management systems are distributed. However, they need to enter accurate inventory quantities.

As a result, to grasp the actual inventory efficiently and to determine the part production quantity P_i became the requirement of the target inventory management system. And, there are also following supplemental requirements. First, from the viewpoint of the cost performance, the target system must be implemented without using expensive equipments and devices. Second, the system operations must be performed without increasing the workload of personnel.

3 APPLICATION METHOD OF MONGODB

3.1 Novel Method to Grasp Actual Inventory

For the problem mentioned in Section 2.3, we changed our idea about the judge method of parts sufficiency from counting the actual inventory. And, we proposed a method to judge whether the necessary quantity is satisfied or not by the human vision based on the following facts. Firstly, as for the parts delivery or the product shipment, the quantity of each necessary parts can be grasped easily by using the MRP system. That is, it can be calculated automatically by the order ID and necessary quantity of each product, and these data is received via EDI as the electronic data and can be used efficiently. Secondly, the human vision can grasp the approximate number of parts efficiently in the various situations.

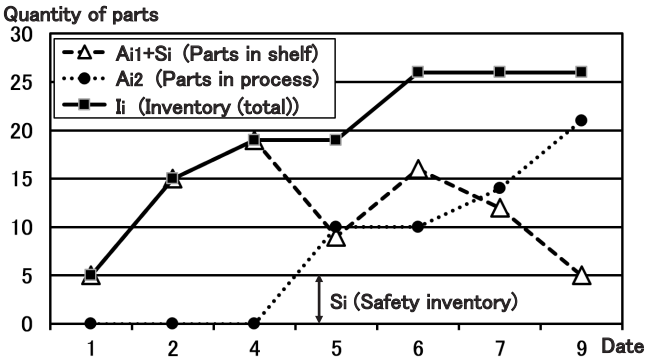


Figure 8: Change of quantity of parts inventory and delivery

Figure 8 shows the change of the theoretical inventory of a parts, which is the necessary quantities and corresponds to equation (1). Incidentally, the product shipments are omitted for the sake of simplicity in this figure. In this case, the parts are prepared in the parts shelf prior to assembly start 3 days, and the safety inventory quantity is 5. For example, 10 parts are prepared (15 including the safety inventory; "$A_{i1} + S_i$") on second, and they are moved to the assembly field on fifth ("A_{i2}"). Similarly, the parts are prepared 4 on fourth; 7 on sixth, and they are delivered on seventh and ninth respectively. So, on second, R_i is 4; I_i is 15; A_{i1} is 10; S_i is 5. Then, the production quantity P_i is 4 on fourth.

However, in the actual field, since there are manufacturing loss and the process delay, they do not always equal to the actual inventory. Therefore, as above-mentioned, to perform the inventory control, the actual inventory must be grasped, too. And, only the judgement that there is the necessary quantity of parts in the parts shelf on the designated date is performed in our proposed method, so it can be done efficiently. For example, assuming that Fig. 8 shows the transition of parts stocked in the rightmost container of Fig. 7 (b), and it is sufficient if more than 15 parts are present on the second day. If we see in Fig. 7 (b), then we find it is easy to judge it by the human vision, even if we do not know the exact inventory quantity. Therefore, by this method, the inventory manager can perform his business efficiently in the office by using the image and video, and he needs no field work. And, in the case where some actual parts inventory may be insufficient, the parts replenishment in lot unit is ordered by the manager.

We show the composition of the proposal system in Fig. 9. The worker takes out the parts from the parts shelf, then take the image and video of this shelf by his hand-held camera. And, he enters it into the database with the parts data: the order ID and product ID. Then, the system calculates the both theoretical inventory as shown in Fig. 8: the one was remained in the parts shelf; the other was delivered to the as-

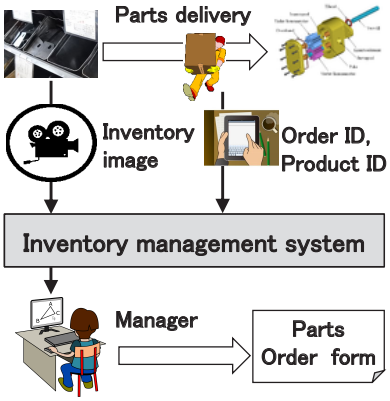


Figure 9: Composition of proposal system

sembly field. In this operation, for example, as shown on the fifth day of Fig. 8, the delivered quantity (here, 10) is reduced from the *document* ($A_{i1} + S_i$) of the parts shelf and the same quantity is added to the *document* (A_{i2}) of the assembly field. And, the total quantity (I_i) that is the sum of the both must not change during this operation.

Here, in order to manage the actual inventory by this method, it is necessary to grasp the exact theoretical inventory shown in Fig. 8. And, since the parts are stocked separately in both the parts shelf and the assembly field, the theoretical inventory is managed by each corresponding *document*. This indicates that the updating of these two *collections* must be processed as a transaction maintaining the ACID properties.

3.2 Requirements for Database Application

As shown in Section 3.1, the database of the proposal system must satisfy the requirements in two sides. The one is the enormous data manipulation to grasp the actual inventory, which is provided by MongoDB as shown in Section 2.2; the other is the transaction management to calculate the theoretical inventory, which is provided by the RDBMS for the usual enterprise system.

That is, the database of the proposal system has to treat not only the character and numerical data of the inventory information, but also the image and video data in the factory. So, if this system was implemented by using the RDBMS, the significant restrictions occurs on the data manipulation as shown in Section 2.2.

For this reason, we used MongoDB for the database of this system. On the other hand, if this system was implemented by using MongoDB, the following restrictions are considered: the ACID properties is not maintained as the whole transaction; the join operation to connect plural *collections* each other is not supported.

In summary, the requirements for the application of MongoDB to the proposal system is as the following. The first requirement is that its transaction can manipulate the plural *collections* with maintaining the ACID properties. The second requirement is its *collections* can be connected each other using only the reference among them. That is, it is composed without the feature not provided in MongoDB: the join operation and so on. The third requirement is that its transaction

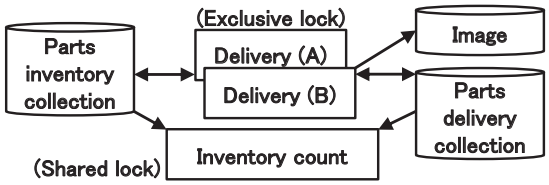


Figure 10: Inventory management model for MongoDB

can efficiently manipulate the enormous data such as images and videos. That is, its manipulation time of such a data must be shorter than the time of MySQL.

3.3 Inventory Management Model for MongoDB

As for the first requirement, it could be satisfied by the transaction feature of MongoDB which was the result of our previous study shown in Section 2.1; as for the second requirement, we composed the system with a few *collections*, and made them to be correlated by the same key field or ObjectID. Based on these policy, we constructed the transaction model of the inventory management shown in Fig. 10.

In the following, we show only the necessary data fields extracted from the actual data fields for the sake of simplification. In Fig. 10, "Parts inventory" *collection* (below, $Parts\ inventory$) saves each parts quantity in the parts shelf; "Parts Delivery" *collection* (below, $Parts\ delivery$) saves the delivered parts quantities, and it has the following fields {order_ID, product_ID, parts_ID, necessary_quantity, shortage_quantity, imageID}; "Image" *collection* (below, $Image$) saves {image_name, image_data} of the parts shelf. Here, imageID is the ObjectID of the $Image$. As for $Parts\ delivery$ at the planning time, {order_ID, product_ID, parts_ID, shortage_quantity} is saved, and the value of {necessary_quantity} is also set to {shortage_quantity}.

$Delivery$ is the transaction which executes the delivery processing of the parts from the parts shelf to the assembly field. And, in Fig. 10, there are transactions $Delivery(A)$ and $Delivery(B)$. They correspond to the product A and B in Fig. 5 respectively. That is, they reduce the parts quantity from $Parts\ inventory$ on the basis of the necessary amount for product A or B, and save the image and video after parts delivery into $Image$. Also, they update {shortage_quantity} of $Parts\ delivery$ according to the delivery of parts. Here, in order to process as a transaction, it performs the delivery processing by each product unit of each order. For example, in the case of product A in Fig. 5, 60 parts X and 20 parts Y is delivered. Then, if any parts is insufficient for its delivery, then no delivery is executed. That is, the $shortage\_quantity$ value is calculated by the following equation.

$$shortage\_quantity = \begin{cases} 0 & (All\ parts\ supplied) \\ at\ plan & (otherwise) \end{cases}$$

Transaction $Inventory\ count$ calculates the total quantity of parts X, Y, Z in two *collections*: $Parts\ inventory$ and $Parts\ delivery$. This process is executed as a single transaction for each parts.

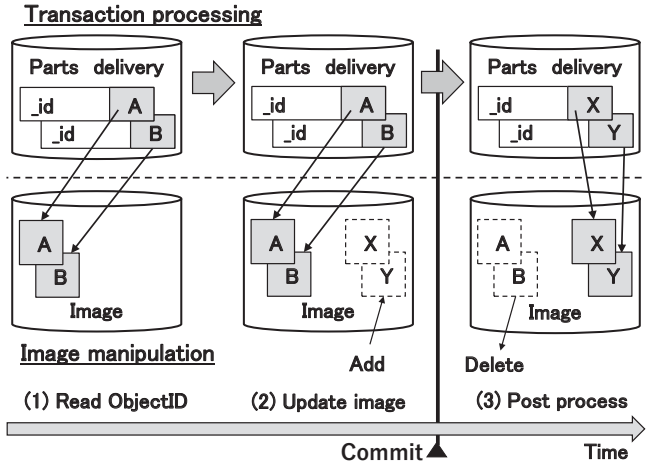


Figure 11: Access method for the enormous data

The concrete requirements to MongoDB in order to implement this model are as follows. First, the concurrency control of the transactions on two *collections* should be performed. That is, $Parts\ inventory$ and $Parts\ delivery$ are updated simultaneously, and queried. Concretely, while the transaction $Delivery$ is updating these *collections*, $Inventory\ count$ is querying these *collections* to calculate the total quantity of the parts as shown in Fig. 8. Here, both of $Delivery$ and $Inventory\ count$ should be executed as a single transaction respectively. That is, if the latter queries the anomaly state of the parts such that one *collection* has already updated and another *collection* has not updated yet, the incorrect inventory quantity is calculated.

3.4 Transaction Processing Method for Enormous Data

As for the third requirement, the images and videos are manipulated to confirm the inventory shelf. In particular, the image and video size becomes very large in the following cases where the long time video data is necessary: the status of the parts shelf shown in (a) of Fig. 7 must be confirmed from various side; various kinds of parts are delivered simultaneously from the parts shelf shown in (b). Therefore, the elapsed time of transaction to manipulate the enormous image and video becomes so long. As a results, since the lock is used in our transaction feature, the extreme latency is expected in the case where the plural transactions executed concurrently.

For this problem, we used an optimistic locking utilizing the ObjectID of $Image$ for the implementation of this method as shown in Fig. 11, in order to reduce the lock time to save the enormous data. Here, the ObjectID is the identifier of the *document* in MongoDB as shown in Section 2.1. So, it can be used instead of the time stamp, and it is used as the link that specifies the reference *document* in MongoDB. Therefore, its procedure is as follows. Firstly, at (1) in Fig. 11, {imageID} of the *document* in *Parts delivery* that is the ObjectID A and B is queried, by which the corresponding *documents* of *Image* are referred. Next, at (2), the new image or image X and Y are added to *Image*, then the transaction to update *Parts delivery* is begun. So, the target documents of *Parts delivery* are locked, and above-mentioned ObjectIDs are queried again: if these ObjectIDs have not been changed, these ObjectIDs are changed to X and Y and its commit is performed; if the ObjectIDs are changed, that is, these *documents* of $image$ have been changed by the other transactions, its rollback is performed. Then, the post process in (3) is performed: in the case of the former, the images and videos having ObjectID A and B are deleted; in the case of the latter, the images and videos having ObjectID X and Y are deleted.

By this method, above-mentioned enormous data manipulation can be separated from the lock period of the transaction, and the lock is performed only while the transaction to manipulate *Parts delivery*. As a result, the lock period can be shortened, and the manipulation of several images and videos can also be performed as a single transaction. By the way, though only the case of update is shown in Fig. 11, the case of addition and deletion of images and videos can be performed similarly: as for the former, firstly the images and videos are added, then the transaction to update *Parts delivery* is performed; as for the latter, firstly the transaction is performed, then the images and videos are deleted.

4 IMPLEMENTATIONS AND EVALUATIONS

4.1 Implementation of Inventory Management Model

We implemented the prototype of the inventory management model shown in Fig. 10 on a stand-alone PC. Its implementation environment is as follows: CPU is i7-6700 (3.41 GHz); memory is 16GB; disk is SSD memory of 512GB; OS is Windows 10. We adopted MongoDB (Ver. 3.3.6) for the database; Java (Ver. 1.8.0_73) for programming; MongoDB Java driver (Ver. 2.14.2) to access MongoDB from Java program. The above-mentioned three transaction programs are performed simultaneously using Thread class of Java, and the transaction feature shown in Section 2.1 was used for their concurrency control, which had been implemented by our previous study. And, we used GridFS class of MongoDB Java driver to store the image and video data to MongoDB [15].

Also, we implemented MongoDB update transaction by the following two methods to evaluate the deterioration of conflicts associated with saving the image and video data. The first method is shown in (2) of Fig. 12, and the image and video data is saved to MongoDB as a part of the transaction, that is, it does not use the method for the enormous data shown in Section 3.4.

Its procedure is as following. Firstly, to confirm whether there is sufficient stock in the inventory shelf, the transaction $Delivery$ query $Parts\ inventory$ query it by using the exclusive lock. So, the conflict between other transactions may occur henceforth. Then, it update {shortage_quantity} of the corresponding *document* in $Parts\ delivery$ to 0. Next, it reduces {storage_quantity} of $Parts\ inventory$, and save the image and video data to $Image$. Finally, it executes the commit. Incidentally, we excluded the case of shortage of inventory in this experiment. In addition, we set the delay between

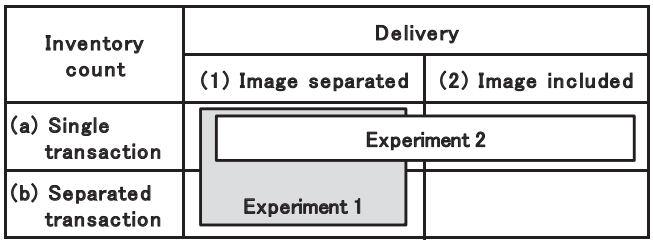


Figure 12: Program structure and experiment

the above-mentioned access of the two *collections* in order to confirm the occurrence of the conflicts. Also, in the case where the conflict occurs, the transaction performs the retry after a certain waiting time.

The second method is shown in (1) of Fig. 12 which is mentioned in Section 3.4. And, it is similar to the first method, except that it saves the image and video data before updating the *document* in $Parts\ inventory$ and $Parts\ delivery$. In other words, since the image and video data is saved prior to the start of the transaction, any *document* is not locked by this image and video data manipulation.

Then, as for the query transaction $Inventory\ count$, we also implemented it by the following two methods, in order to evaluate the difference between the execution as the single transaction and as the multiple transactions ("separated transaction" in Fig. 12 (b)). Here, the latter corresponds to the MongoDB's method such as "findOne".

In the first method, in order to prevent the *collections* to be updated by other transactions during its query, it queries each *collection* by using the shared lock. And, based on the query results, it calculates the sum of the parts. In this way, after it completes the processing, it executes the commit. Then, after waiting for a certain time, the next transaction is started to query another parts.

The second method is similar to the first method, except that it executes the commit after querying $Parts\ inventory$; then it queries $Parts\ delivery$. That is, it separates the query processes into two transactions. We show these two methods in (a) and (b) in Fig. 12 respectively.

4.2 Experiments and Evaluations

We conducted the experiments by the implementation program of the inventory management model, and evaluated the methods. The setting and procedure of the experiments are as follows. We set the sufficient inventory quantity of each parts as the initial value of $Parts\ inventory$. Also, we saved enough order data into $Parts\ delivery$.

Then, we started the three transactions in Fig. 10 at the same time: two update and one query transactions. As for the update transactions, we set 100 msec for the delay time between update the two *collections*. We also set 100 msec for the delay of next transaction start. As for the query transaction, we also set 250 msec for the delay of next transaction start. And, for both transactions, we set 50 msec for the delay before the retry when the conflict occurs. In the experiment, we executed each update transaction 12 times, and the query transaction 14 times. We used the same image data for every

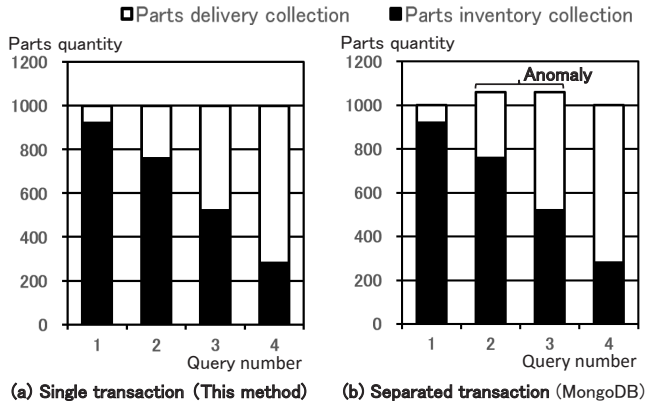


Figure 13: Result of experiment 1

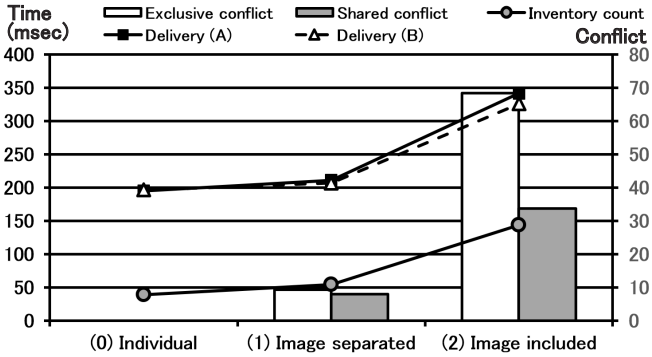


Figure 14: Result of experiment 2

transaction in order to simplify the evaluation. And, its size is 3.3MB.

For the evaluations about the first requirement, we conducted the following experiment 1 and 2. In experiment 1, we conducted the comparative evaluation between the methods (a) and (b) in (1) of Fig. 12, and Fig. 13 shows its results. Here, "Parts quantity" shows sum of the following quantities queried by *Inventory count* transaction: one was the quantity in *Parts delivery* which was as of after the delivery; the other was in *Parts inventory* which was as of before the delivery. And, Fig. 13 shows the query results at 4 check point when the quantity in *Parts inventory* of the both graph was equal. Here, the total parts quantity of the both must be always constant. And, as for (a), the query result of the total is always constant. However, as for (b), the query result is increased depending on the query timing. That is, since the query process was separated into two transactions, the anomaly occurred by querying the halfway state. As a result, it was confirmed that the isolation of the transactions is maintained also in this application field, by our transaction feature.

Figure 14 shows the results of experiment 2, in which we conducted the comparative evaluation between the methods (1) and (2), in (a) of Fig. 12: as for (1), the image data was saved before the transaction start as shown in Fig. 11, so the time period of transaction was shortened; as for (2), the image data was saved inside the transaction. Incidentally, this experiment was conducted in the case where image data was added. Here, this experiment was performed three times for each case, and Fig. 14 shows the average of these results.

```
-- Add (insert) of image data
insert into image values
    (1,1,LOAD_FILE('Uploads/img.MTS'));

-- Query (select) of image data
select image into dumpfile '/Uploads/img_1.MTS'
    from image where d_id=1 and sub_id=1;
```

Figure 15: Video manipulation statement of MySQL

```
echo %date% %time%
REM Add (put) of image data
mongofiles -v -d test put img_1.MTS -l img.MTS
echo %date% %time%

REM Query (get) of image data
mongofiles -v -d test get img_1.MTS
echo %date% %time%
```

Figure 16: Video manipulation command of MongoDB

Prior to this experiment, we measure the individual elapsed time of $Delivery$ and $Inventory\ count$ transaction. In this case, only one transaction is executed at the same time, and there is no conflict. We show this result in (0) of Fig. 14.

In Fig. 14, the line graph shows the change of the elapsed time for each transaction; the bar graph shows the number of the conflicts occurred for exclusive lock and shared lock respectively. (1) of Fig. 14 shows the experimental result of the method of (1) in Fig. 12. As the result in this case, the number of each conflict was about 10; the increase of elapsed time from (0) was about 10%. On the other hand, as shown in (2), in the case where the image data is stored as a part of the transaction, the number of the exclusive and shared conflict was about 70 and 30 respectively; the elapsed time of $Delivery$ transaction became 1.7 times longer than (0); $Inventory\ count$ transaction became 2.7 times longer. Therefore, as for the enormous data such as image and video data, the conflicts could be reduced by the method shown in Fig. 11.

4.3 Comparison Evaluations of Image and Video Data Access Performance

For the evaluations of access performance of image and video, we conducted the following experiment. That is, it is the performance comparison about the enormous video data between MySQL and MongoDB. Here, as shown in Section 2.2, though there are some restrictions as for MySQL, it can save the video as the binary data. Therefore, this experiment was performed using video data up to 1 GB, which is within the restriction of MySQL. We used BLOB type in MySQL, and GridFS interface of MongoDB.

The videos were shot by the digital camera, and its data size was about 121 MB per minute. In this evaluations, we used the data of 1, 2, 4 and 8 minutes. And, as for each data we measured the following elapsed time, respectively: firstly, we saved the data into the database, then queried the data. We performed these data manipulations outside of the transaction processing as shown in Fig. 11. As for MySQL, we performed the SQL statement by the MySQL monitor, and

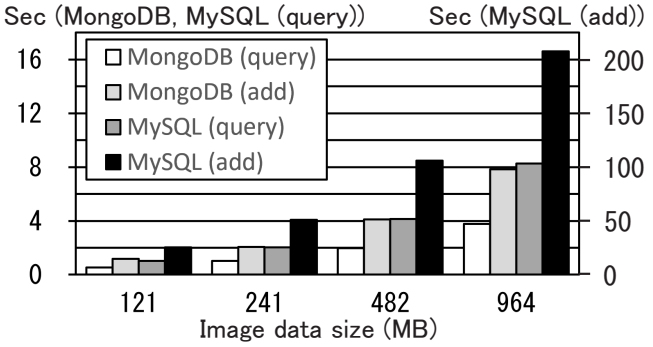


Figure 17: Evaluation result of performance comparison

Table 2: Comparison of man-hours per year

No	Classification	CapEx *	OpEx	Total
(1)	No systematization		1,200	1,200
(2)	Inventory planning	72	60	132
(3)	Image management	72	4	76

*: Man-hours per year when the life cycle is 3 years.

grasped their elapsed times by the displayed execution time on the monitor. We show the statements in Fig. 15: we added the data into the database by the “insert” statement, then loaded the data into the file by the “select into dumpfile” statement. Here, the folder name is simplified. And, as for MongoDB, we performed the batch file as shown in Fig. 16: we added the data into the database by the “put” command, then query the data by the “get” command; their elapsed times were grasped by displaying the system time using the “echo” command.

We show the experimental result in Fig. 17. It shows the average time of three executions as for each processing. The elapsed time increased in proportion to the amount of data in all the cases. In Fig. 17, the elapsed time of additional case in MySQL is shown by the right axis of the graph. It was about 25 times the additional case in MongoDB: in the case where the additional data size was 964 MB, the time in MySQL was more than 200 seconds; whereas the time in MongoDB was 8 seconds. In addition, the query time in MySQL was substantially the same as the additional time in MongoDB; the query time of MongoDB was about half of it.

4.4 Evaluations of Cost Performance of Proposal System

We evaluate the cost performance to introduce and operate the proposal inventory management system shown in Fig. 9. Currently, as the first step, it is planned in the target factory to gradually introduce an inventory management system for 100 common parts which are most frequently used. We show this systemization plan in Table 2, and we are advancing the systemization of inventory planning shown at (2) now. We show the detail of each systemization classification as the following.

(1) **No systematization:** This is the current situation, that is, no inventory management system has been introduced. In

this case, in order to prevent the parts shortage, it is essentially necessary to perform the stocktaking every day.

(2) **Inventory planning:** This is the systemization of the function to calculate the transition of the necessary inventory quantity due to the order information received via the EDI. That is, the theoretical inventory shown in Fig. 8 is calculated automatically. So, since it is necessary to perform the stocktaking only to grasp the difference between the actual inventory and theoretical inventory, it becomes enough to perform once per month.

(3) **Image and video management:** In addition to (2), the proposal inventory management of this study is systematized to improve the stocktaking efficiency, which utilizes the image and video of parts shelves. Incidentally, in actual operation, we are planning to take the images and videos at the time of delivery of inventory. However, we evaluate it as the monthly stocktaking for the sake of comparative evaluation with (2).

For each of these systemization, we calculated the annual man-hour from the following two perspectives: the one is capital expenditure (hereinafter, "CapEx"), which contributes to the fixed infrastructure of the company and they are depreciated over time; the other is the operational expenditures (hereinafter, "OpEx"), which do not contribute to the infrastructure itself and consequently are not subject to depreciation [29]. Here, the former is the man-hour to develop and introduce the system, and we assume that the system is depreciated in 3 years. That is, we assume the life cycle of the system is 3 years, so the man hours of CapEx shown in Table 2 are the results divided by 3. And, each man-hour of Table 2 was calculated as follows.

(a) **CapEx:** This is the man-hour to develop and introduce the system, and we calculated it based on the MRP system that has been already introduced. In (2) and (3), the man-hour for the development including the function addition and improvement after the trial use is 160 man-hours; the man-hour for introduction is mainly the data entry work, it was calculated 54 man-hours due to the ratio of the number of types of the target parts based on the introduction man-hour of the MRP system. We divided these total 214 man-hours by 3 years, which is the life cycle of the system, to calculate the annual man-hour.

(b) **OpEx:** This is the annual man-hour for the stocktaking. As for (1), we used the estimated stocktaking time of all parts obtained by the preliminary investigation. And, due to the ratio of the target parts, we calculated it as 5 man-hours for one time, then multiplied by the number of annual stocktaking regarding it as the daily work. As for (2), we calculated it as of monthly stocktaking, so it is 1/20 of (1). As for (3), we used 12 man-seconds per one type of parts, which was obtained by the experiment, and calculated in the same way as (2).

(c) **Total:** This is the total of (a) and (b).

Now, since OpEx is too large to perform as the daily regular work as shown in (1) of Table 2, it cannot be performed. On the other hand, by the systemization of the inventory planning, OpEx it is expected to be 1/20 as monthly work. So, the stocktaking is expected to be able to perform as the regular work. And, we estimate that the total man-hour including the system development and introduction can be reduced by about 90% per year. Furthermore, by adding the systemization of the image and video management (3), the man-hour of the stocktaking can be reduced, and we estimate the total man-hour can be reduced by 45% from (2).

Incidentally, the man-hour of the system development and introduction is smaller comparing to the usual commercial system. In addition, since this system can be composed on the existing PC by using only the free or existing software, the capital investment is not needed. This reason is because this system is introduced as a prototype system and we do not regard the operability such as the user interface. Instead, our students assist the system operation at the factory if necessary, and this is included the man-hour in Table 2. Incidentally, the work performed as research is not included in this man-hour, such as the method study and evaluation, technical investigation.

Then, this company evaluates the effectiveness of the system, and decides the introduction of the commercial system or using this system. For example, in the case of the preceding MRP system, they introduced a commercial system for the part related to EDI; they are using our prototype system to calculate the material cost for their estimations. The reason is because the former is related to the business connection with the other companies, and the high quality and operability are necessary; the latter needs to change the calculation flexibly according to various estimation conditions.

5 DISCUSSIONS

With the spread of the IoT, various types of enormous data are used widely. And, to store these data efficiently, GridFS interface of MongoDB is provided. So, it is expected that the enterprise system also can be more useful by using such a feature. However, as for this, there were the problems to be satisfied the following requirements as shown in Section 3.2: first, the transaction must maintain the ACID properties; second, the data manipulation must be executed without using the join operation; third, the enormous image and video manipulation must be performed as a transaction. So, there has been no its application case to the enterprise systems.

On the other hand, the inventory management work of our assisting factory was expected to be more efficiently by using the image and video data for the stocktaking. So, we conceived to apply GridFS interface of MongoDB to its production management system. And, through this application, we confirmed that the above-mentioned feature can be applied to the inventory management system. Concretely, we satisfied the requirement as following: first, we used the transaction feature which we had developed as the previous study; second, we connected *collections* by the reference using ObjectId or same key field; third, we composed the optimistic locking feature by utilizing the source *document* linked to the image

and video *document*.

As a result, we confirmed MongoDB could be applied even to the enterprise system through the experiments, that is, it satisfied the above-mentioned requirements. First, as shown in Fig. 13, the anomaly of the transaction to update the plural data could be avoided by our transaction feature. Second, the collections could be composed to refer each other including the inventory data *collection* and the image and video *collection*, as shown in Fig. 11. That is, the reference using ObjectID and so on could be used instead of the join operation. Third, as shown in Fig. 14, the enormous image and video manipulation could be performed as a transaction by the optimistic locking shown in Fig. 11. In addition, its performance is better than MySQL as shown in Fig. 17.

Now, with the development of the IoT, the databases, which can handle the wide diversity of data, is expected to spread to the enterprise systems. Along with this, it is expected that many devices access the NoSQL database concurrently like this system, too. Therefore, we consider that the needs of transaction feature for the NoSQL databases would grow more. And, by this feature, we consider that the application field of the databases can be expanded to the field where the problems has occurred by using not only the RDBMS but also the conventional MongoDB.

Next, as for the efficiency by using the image and video data, we showed the case study of the stocktaking in the inventory management. As shown in OpEx of Table 2 (2) and (3), it is estimated that its man-hour can be drastically reduced. This reason is because the stocktaking work could be changed from counting the actual parts to judging that the actual inventory was sufficient by the flexible human vision.

And, there was the requirements to introduce this system into the target company, that is, the small and medium-sized company: the workload of the personnel should not increase; the system could be implemented at a low cost. As a result, as shown in Section 4.4, we could satisfy these requirement, though it was a prototype. Therefore, we consider that the system that utilizes the various type of data such as images and videos is useful for such companies. Furthermore, we consider it is useful for the other various fields.

6 CONCLUSIONS

Currently, various type of data can be used by the spread of the IoT and the development of the NoSQL database. However, to apply the NoSQL database to the enterprise systems, there were some problems such as the transaction feature. In this paper, we showed the application case of MongoDB which is a kind of NoSQL database to the production management system. We implemented the function mainly for the stocktaking as the prototype system, and we are advancing to introduce this system to the target factory. As a result, we confirmed MongoDB could be applied to the enterprise system by equipping the transaction feature maintaining the ACID properties; the function of NoSQL database such as the manipulation of the enormous data is useful even in the enterprise systems.

The feature challenge will focus on the improvement of the efficiency of actual production management system operations such as the numerical, image and video data entry at the field. In addition, we will confirm that this transaction feature can be implemented in the distributed database environment; the enormous data manipulation can be performed more efficiently in this environment.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 15K00161.

REFERENCES

[1] K. Banker, “MongoDB in Action,” Manning Pubns Co. (2011).

[2] M. Bertolini, et al., “Reducing out of stock, shrinkage and overstock through RFID in the fresh food supply chain: Evidence from an Italian retail pilot,” International Journal of RF Technologies, Vol. 4, No. 2, pp. 107–125 (2013).

[3] R. Cattell, “Scalable SQL and NoSQL data stores,” ACM SIGMOD Record, Vol. 39, No. 4, pp. 12–27 (2011).

[4] I.J. Chen, “Planning for ERP systems: analysis and future trend,” Business process management journal, Vol. 7, No. 5, pp. 374–386 (2001).

[5] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” Mobile Networks and Applications, Vol. 19, No. 2, pp. 171–209 (2014).

[6] A. Eisenberg, and J. Melton, “SQL:1999, formerly known as SQL3,” ACM Sigmod record, Vol. 28, Issue 1, pp. 131–138 (1999).

[7] J. Gray, and A. Reuter, “Transaction Processing: Concept and Techniques,” San Francisco: Morgan Kaufmann (1992).

[8] S. Hiremath, G. Yang, and K. Mankodiya, “Wearable Internet of Things: Concept, architectural components and promises for person-centered healthcare,” EAI 4th International Conference on Wireless Mobile Communication and Healthcare (2014).

[9] C.L. Iacovou, I. Benbasat, and A.S. Dexter, “Electronic data interchange and small organizations: adoption and impact of technology,” MIS quarterly, Vol. 19, No. 4, pp. 465–485 (1995).

[10] T. Kudo, M. Ishino, K. Saotome, and N. Kataoka, “A Proposal of Transaction Processing Method for MongoDB,” Procedia Computer Science, Vol 96, pp. 801–810 (2016).

[11] D. Laney, “3D Data Management: Controlling Data Volume, Velocity and Variety,” META Group (2012), http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf (reffered May 5, 2017).

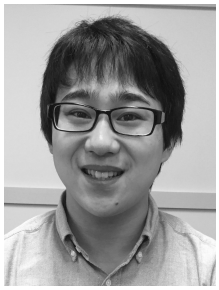
[12] H. Garcia-Molina, and K. Salem, “SAGAS,” Proc. of the 1987 ACM SIGMOD Int. Conf. on Management of data, pp. 249–259 (1987).

[13] K. Hong-yan, “Design and realization of internet of things based on embedded system used in intelligent campus,” International Journal of Advancements in

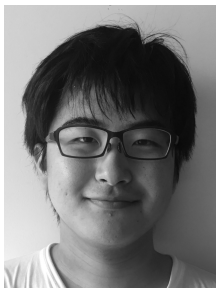
Computing Technology, Vol. 3, No. 11, pp. 291–298 (2011).
[14] MongoDB Inc., “The MongoDB 3.4 Manual,” http://docs.mongodb.org/manual/ (reffered May 5, 2017).
[15] MongoDB Inc., “MongoDB API Documentation for Java,” http://api.mongodb.org/java/ (reffered May 5, 2017).
[16] MongoDB Inc., “Write Operation Performance,” https://docs.mongodb.com/v3.4/core/write-performance/ (reffered May 5, 2017).
[17] Oracle Corp., “MySQL 5.7 Reference Manual,” http://dev.mysql.com/doc/refman/5.7/en/ (reffered May 5, 2017).
[18] J. Pokorny, “NoSQL databases: a step to database scalability in web environment,” International Journal of Web Information Systems, Vol. 9, No. 1, pp. 69–82 (2013).
[19] D.R. Rebecca, and I. E. Shanthi, “A NoSQL Solution to efficient storage and retrieval of Medical Images,” International Journal of Scientific & Engineering Research, Vol. 7, No. 2, pp. 545–549 (2016).
[20] E. Redmond, and J.R. Wilson, “Seven Databases in Seven Weeks: A guide to Modern Databases and the NoSQL Movement,” Pragmatic Bookshelf (2012).
[21] R. Sears, C. Van Ingen, and J. Gray, “To blob or not to blob: Large object storage in a database or a filesystem?,” arXiv preprint cs/0701168 (2007).
[22] K. Seguin, “The Little MongoDB Book” (2011), http://openmymind.net/mongodb.pdf (reffered May 5, 2017).
[23] L.A.B. Silva, et al., “Medical imaging archiving: A comparison between several NoSQL solutions,” Int. Conf. on Biomedical and Health Informatics, pp. 65–68 (2014).
[24] The Small and Medium Enterprise Agency, “2015 White Paper on Small and Medium Enterprises in Japan and White Paper on Small Enterprises in Japan (outline),” (2015), http://www.chusho.meti.go.jp/pamflet/hakusyo/H27/download/2015hakushogaiyou_eng.pdf (reffered May 5, 2017).
[25] S.S. Sriparasa, “JavaScript and JESON Essentials,” Packt Pub. Ltd. (2013).
[26] M.P. Stević, B. Milosavljević, and B. R. Perišić, “Enhancing the management of unstructured data in e-learning systems using MongoDB,” Program, Vol. 49, No. 1, pp. 91–114 (2015).
[27] M. Stonebraker, and C. Ugur, “One size fits all”: an idea whose time has come and gone,” Proc. of 21st Int. Conf. on Data Engineering, pp. 2–11 (2005).
[28] M. Stonebraker, et al., “The end of an architectural era:(it’s time for a complete rewrite),” Proc. of 33rd Int. conf. on Very large data bases. VLDB Endowment, pp. 1150-1160 (2007).
[29] S. Verbrugge, et al., “Modeling operational expenditures for telecom operators,” Proc. of Conf. on Optical Network Design and Modeling, pp. 455–466 (2005).

(Received October 9, 2016)
(Revised May 9, 2017)

Tsukasa Kudo received the M.E. from Hokkaido University in 1980 and the Dr.Eng. in industrial science and engineering from Shizuoka University, Japan in 2008. In 1980, he joined Mitsubishi Electric Corp. He was a researcher of parallel computer architecture, an engineer of application packaged software and business information systems. Since 2010, he is a professor of Shizuoka Institute of Science and Technology. Now, his research interests include database application and software engineering. He is a member of IEIEC and Information Processing Society of Japan.



Yuki Ito is currently working toward a B.I. degree at Shizuoka Institute of Science and Technology. His research interests include IoT, web system and SEO.



Yuki Serizawa is currently working toward a B.I. degree at Shizuoka Institute of Science and Technology. His research interests include database application and production management system.