A Distributed Method for Multiplication of Large Matrices using MSMQ Middleware

Hassan Ziafat, Sayyed Morteza Babamir
University of Kashan, Kashan, Iran
ziafat@grad.kashanu.ac.ir

***Abstract*—Multiplication of large matrices is time consuming. Although parallel algorithms have been presented to reduce the multiplication time, distributed computing and algorithm mechanism is also able to help us in reducing the time. In this paper, we aim to present a new distributed method for multiplication large matrices using the MSMSQ middleware. The multiplication of the large matrices is used in various engineering fields. We came to a conclusion that the proposed method reduces the multiplication time of large matrices to an adequate level.**

***Index Terms*—Multiplication; Large Matrices; Distributed Computing; Middleware; MSMSQ.**

I. INTRODUCTION

A distributed system consists of independent computers which is connected each other through a network and a middleware interface. The computers, which are viewed as an integrated system for its users, communicate through the exchange of message and provide various services for its users. Figure 1 shows the structure of a distributed system [1].

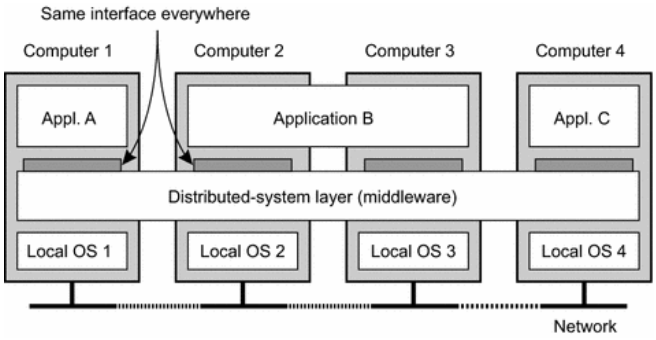

Figure 1:Distributed systems architecture [1]

One of the important features of distributed systems is that their users and the methods of communication are hidden. A distributed system is a faulty tolerant, which means that if one of its stations fails, it is still accessible for its user. Moreover, users are not aware of the station failure and recovery.

A distributed system speeds up the execution of tasks as it uses several computers simultaneously. The objectives of distributed systems are providing the following matters to their users: (1) high resource accessibility, (2) access transparency in using resources, (3) communication intricacies hiding, (4) openness (i.e. the acceptance of adding new standards by the system, and (5) scalability and extensibility. Distributed system aims that users can easily have access to the remote resources and control the resources when sharing with others. Such accessibility facilitates information exchange and communication and makes the system more reliable.

A distributed system, which is capable of presenting itself as a single system to the users and its applications is called a *transparent* system. The transparency feature of a distributed system indicates hiding the dispersal of the system components (computer) for users. There are some types of transparency [1]:

- Location: Hiding physical location of resources from the users using Domain Name Server (DNS);
- Relocation: Hiding relocation of resources from the users; and
- Concurrency: Hiding concurrent use of data and resources from users.

There are three types of scalability:

- Size: adding new resources to the system easily; and
- Geography: distributing users and resources geographically.

Hereafter, "resources" refers to the data that contain the matrices elements; we distribute them between clients (the system workstations) in order to calculate matrix multiplication.

A. Distributed Systems

There are different types of distributed systems:

- Computing Systems: a distributed computing system is one that is highly efficient in computing distributed applications: *clusters* and *grids* are such systems[1];
- Information Systems: These systems are connected directly to a database with a client – server structure; and
- Pervasive Systems: connected mobile devices with limited memory and processing power.

In this paper, we applied the distributed computing system, where a middleware is used to communication between users.

B. Middleware

A networked system without the middleware is not recognised as a distributed system, single or coherent system for their users. To convert a networked system into a distributed system, an additional software layer is required to be installed and executed on all network machines. This software layer hides the hardware heterogeneity, causing

transparency or invisibility of distributed computers. This layer is called middleware, which enjoys expandability (scalability), openness, and transparency. These enable easy communication of stations of the network (see Section 1). In this paper, a message-oriented middleware called MSMQ is used. Various kinds of transparencies can be seen in this architecture so that matrix elements can be moved from one station to another, and the calculations are done concurrently. Subsection A of this section provides the location, relocation and concurrency transparencies.

II. MESSAGE ORIENTED MIDDLEWARE (MSMQ)

MSMQ is a queue-based inter-process communication system; it was implemented by Microsoft in the Windows 95 operating system[2] and was supported in the next generation of Windows. Its last version is included in Windows 8.

MSMQ is known as a MOM (Message Oriented Middleware). It establishes a mechanism to integrate communicating process into a *loose* (message oriented) connection. In this method, processes make asynchronous communication with each other. Figure 2 shows how MSMQ works. As the figure shows, it consists of some public/ private queues. Message structure consists of three parts: header, body and labels [3].

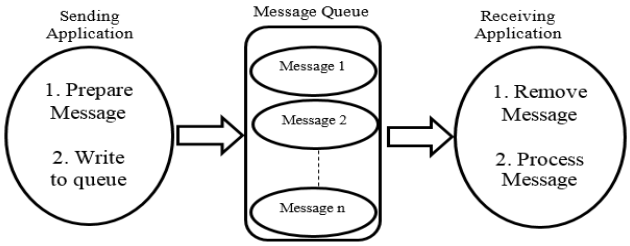

Figure 2: Interprocess communication in MSMQ

A. *MSMQ queue*

There are three types of queues in MSMQ:

a. *Private*

Using this queue, applications put/read their private messages on/from the queue. Figure 3 shows a private queue.

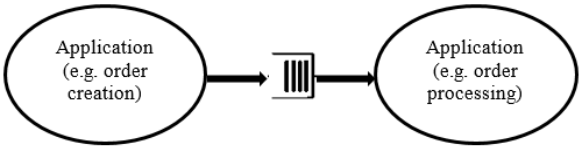

Figure 3: View of Private Queue

b. *Public*

This type of queue is considered for exchanging messages between the system server and clients. In this model, messages remain in the queue after reading; however, in the private model, messages are removed from the queue after being read by clients. Figure 4 shows a general view of the public queue.

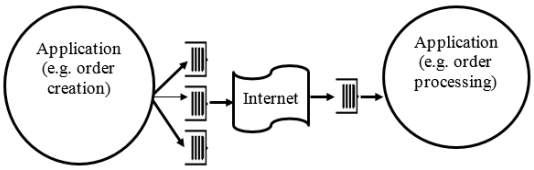

Figure 4: Public Queue

c. *Systematic*

This queuing model is used for specific purposes.

i. *Journal* message queue: This queue is used to save all sent messages. This option is enabled by setting the parameter *UseJournalQueue*.

ii. *Dead-letter* message queues: This queue is used to save the messages not delivered or their timespan has come to an end.

iii. *Transactional dead-letter* message queues: This queue is similar to the Dead-letter message queue, but it is used for transactional messages [3]. Figure 5 shows the view of these queues in the MSMQ menu.

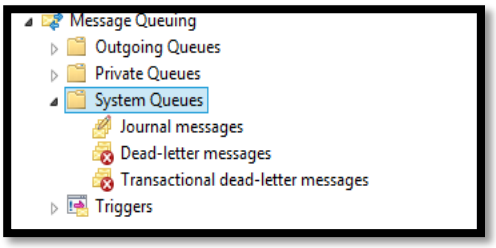

Figure 5: Queues in MSMQ

B. *Communications in MSMQ*

a. *Point-to-Point*

Using this method, a message is sent to a queue and read by a recipient only: It is a one-to-one communication.

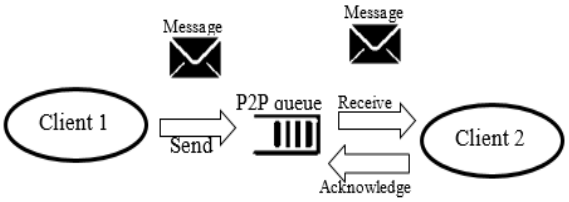

Figure 6: The point-to-point communication

As shown in Figure 6, Client1 sends a message to the queue and Client2 takes it and puts an ACK in the queue. This method represents a point-to-point connection where sender and the message receiver can remove message from the queue at any time.

b. *One-to-many*

In this method, each message is put in queue under a Topic so that clients can access the messages according to topics. The message sender and receiver are called *publisher* and *subscriber* respectively (Figure 7). As shown in Figure 7, the creator locates a creative message in the Topic, and Client2 and Client3 refer to the topic client and receive them in order

to use them. In this method, publishers and subscribers are interdependent in terms of time. A client can consume the message if it has been located on the topic after being subscribed and has been on the Topic at the time of creating active message and the subscriber.

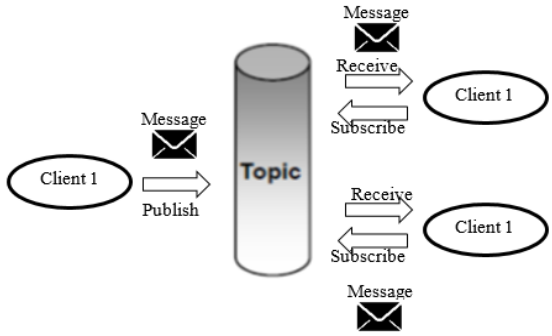

Figure 7: One-to-many communication

III. Related Works

In 1988, Walter and Tichy [4] considered six types of algorithms for matrix multiplication using a network environment. To make a link between the computers, a particular model and pattern was used so that calculation and distribution of results were done faster. The results showed better performance and higher speed in matrix multiplication over a single computer. The problem with their pattern was that the high complexity of sending and receiving messages. In 2000, Bymant and his colleagues [5] tested the operation of matrix multiplication using heterogeneous environments. They aimed to carry out matrix multiplication using network and individual system. Using a particular combination, they divided the matrix into different parts and then performed the matrix multiplication using a local computer and the network environment. The proposed method had some favourable performances and reduced the time of multiplication by 16%. Following this work, in 2009, Cotton et al. [6] used a distributed system for matrix multiplication and applied Feed Forward Operation and Back Propagation using Neural Networks. They produced the primary neurons in the local system and computed the input, middle, and final layers using a distributed system. Due to the performance of their method, the speed of Neural Network outputs was increased.

There is a big disadvantage when making communication between computers, using a network system without a middleware. Assume that one of stations: (1) faces a problem in receiving a message, (2) crashes when a message reaches it crashes, or (3) the message is lost for the network disconnection. Accordingly, the elements of the matrix multiplication are missed and the entire multiplication would be suspended. However, the distributed systems using queue-based middleware, such as the MSMQ has the ability to resolve this big disadvantage as follows:

If any of the system stations encounters a problem, the message, including data matrix will be in the destination or departure queue and is delivered to the destination after resolving the problem. Another option is using a distributed system whose stations enjoy multi-core CPU.

In 2014, Ismail et al. [7] used a simple and efficient algorithm called SPC3 PM using multi-core systems to matrix multiplication; it took shorter running time over standard parallel processing. They considered the multiplication of matrices in different sizes from 100 * 100 to 10,000 to 10,000 using 24 cores. All the multiplications showed improvement in the performance. The multiplication time of a 10000*10000 matrix was 6.80, 13.22, 9.19 and 23.75 using 4, 8, 12 and 24 cores respectively.

In a survey conducted by Yang Hong-Yan [8], the performance of using the cache memory in matrix multiplication on multi-core processors has been investigated. The matrix multiplication has been studied in two ways: using (1) a multi-core computer with shared cache memory, and (2) a set of multicore computers with non-shared memory, which are connected to each other as a cluster. Experiments were conducted using 10 computers with matrices of variable lengths. The research findings by [8] showed the performance of the network computers with non-shared cache memory is better than multi-core computers with shared memory as the RAM and CPU usage were reduced by 93% percent.

Compared to the first method, the second method enjoys speed and efficiency improvement; however, because all computations were done using a single computer, it is possible that the computer runs low on available memory. Although a network environment with multi-core systems has been used, similar to [4,5,6], the entire multiplication will be suspended if a portion of the system encounters a problem. As a whole, due to the nature of a network, it may loss in delivering messages. These problems motivate us to propose a method with less cost and higher security and efficiency.

IV. Proposed Method

Multiplication of big matrices is one of the commonly used methods in mathematics, quantum physics, statistics, accounting and many others. However, multiplication of big matrices is a concern because it takes too much time as solutions are using a network structure or parallel algorithms. As mentioned in Section III, losing messages is a matter of concern; accordingly, we try to deal with it. Compared with the parallel architecture and the network, the increased efficiency and less time consumption is an advantage of using MSMQ technology to distribute messages. This technology is available by Microsoft Windows and it is simple to implement. The method presented in this article includes six stages as follows:

- Breaking matrices to smaller ones;
- Sending Matrices to MSMQ;
- Sending matrices to clients through MSMQ;
- Performing matrix multiplication by each client;
- Sending the Matrix obtained from MSMQ to clients; and
- Getting Results from MSMQ and integrating them to a matrix to produce the final matrix.

A. Splitting Matrix

For multiplication of 2 matrices, say A and B, matrix A is split (horizontally or vertically) into n matrices, named A_1 to

A_n (Figure 8).

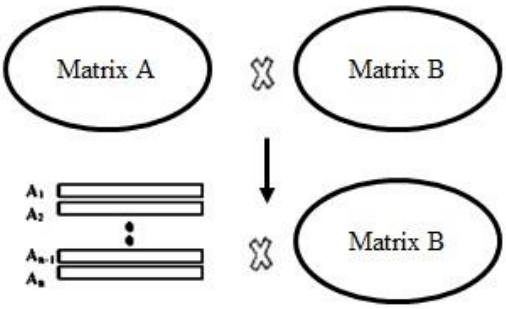

Figure 8: Matrix division

In case of horizontal split, the multiplication is carried out by multiplying each row of the matrix by a column of matrix *B* and in case of vertical split, the multiplication is done by multiplying each column of the matrix by a row of *B*. In this paper, we use the horizontal division.

B. Sending Matrix to MSMQ

Each of the matrices A_1 to A_n along with matrix *B* is sent to the MSMQ management system as a portion of the entire multiplication operation.

```
For i = 1 to n
   Send row i of A to MSMQ Manager
Send Matrix B to MSMQ Manager
```

MSMQ management includes 2 public queues called Result and Matrix. The matrix queue holds A_1, *B* to A_n, *B* and the result queue holds the received results from existing computers in the distributed system.

C. Sending matrices to the clients

At this stage, A_1, *B* to A_n, *B* are sent from the matrix queues to the current computers. In each computer, there is a *MAT* for holding the sent data (Figure 9).

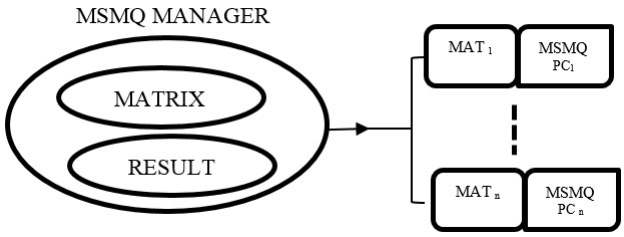

Figure 9: The quality of sending information to the client.

We must create a message to send the information to MSMQ. The message sent from the server to client and vice versa should be in such a way that the matrix multiplication operation is broken into fragments, and the final summary is simplified by each client or a server. For this reason, we used a *XML* structure to create the message including a head and a body: Using such structure to send messages is a part of our innovation. To build the structure, a matrix is converted to a string, and a separator character is used between the matrix elements. Similarly, a separator character is used between the matrices. The created string constitutes the message body. In the message head, there are parameters *m* (the number of rows), *n* (the number of columns of the matrix A_1), *x* (the number of rows) and *y* (the number of columns of the matrix *B*) Figure 10 shows the message structure.

Head	m . n . x . y
Body	$A1_{11}$; $A1_{12}$; …; $A1_{mn}$ @ B_{11};B_{12};..; B_{xy}

Figure 10:The sent structure in the proposed method.

The code for the generation of the message structure in the form of a string is:

```
Message.Head = m concat. n concat. x concat. y
For i = 1 to m {
    For j = 1 to n {
      Message.Bodyconcat with A[i,j]
      Message.Bodyconcat with ";"
      }
 }
  Message.Bodyconcat with "@"
  For i = 1 to x {
    For j = 1 to y {
       Message.Bodyconcat with B[i,j]
      Message.Bodyconcat with ";"
     }
 }
Send Message to Client [i]
```

Variables *m*, *n*, *x* and *y* are converted to string and are separated by a dot. They are joined together as a string and stored in the message header. In the message body, elements of matrix A_i are arranged in tandem with character ';' between them and character '@' at the end. Similar to matrix A_i, matrix *B* is constructed. Finally, the string is stored in the message body and sent to the client.

D. Multiplication in client side

There is a queue in each system station named *MAT* for storing messages from MSMQ. In each station, there is a time-setting, which controls the queue contents every 1 millisecond. And if it is compatible with the data set, the message is read from the queue and matrices *A* and *B* are constructed. Multiplication of matrices *A* and *B* is calculated sequentially as follows, and the result is stored in matrix *C*.

```
For  i=1  to  m-1
  For j=1 to n-1
    For k=1 to y-1
        C[ i, j]=C[ i, j] + A[ i, k ]*B[ k, j]
```

E. Sending the resultant matrix

Matrix *C* is sent to MSMQ by the client. This matrix is constructed in the form of string (like the previous stage) and the matrix elements are separated by separating characters. In the message header, the client number is placed so that it will be identified by the server. The matrix elements are sent in the

XML structure and are placed in the result Queue. Figure 11 shows how respond message is sent to the MSMQ management.

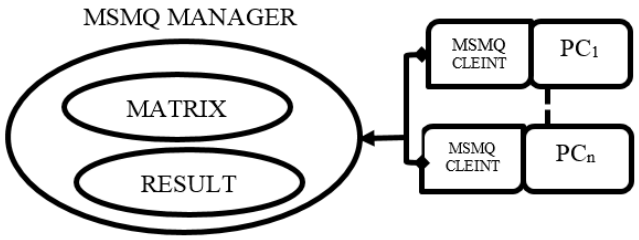

Figure 11: Sending result matrix to the MSMQ Manger

F. Obtaining and Integrating Results

The MSMQ system consists of a time scheduler server for obtaining messages from the result Queue of MSMQ Manager. It identifies the corresponding client response (i.e. the production element) based on the client number in the message header. Then, the production element is laid to the corresponding element in the production matrix. Integration of the matrices will be carried out leading to product matrix, when all clients send their responses. The time order of multiplication of two matrices on local computers is $O(n^2)$ where n is the number of rows/columns of the first/second matrix. Now, we deal with the time order of multiplication of two matrices using the proposed method:

- Time of splitting matrix (T_m),
- Send time of the split section to a client (T_s),
- Computing time used by a client T_c,
- Send time of response from a client to the server (T_r),
- Integration of responses (product elements) in by the MSMSQ server and generation of the final response (result product matrix), T_f.

The time order of sending and receiving messages in the form of strings, is $O(n)$. The integration and split operations are accomplished in a form of string and the time order is $O(n)$. Because matrices are split into N parts, the time order of multiplication in the client side is $O(\frac{n^3}{N})$ where n and N are the number of matrix rows and clients respectively. According to this method, the time order is T_W, as shown in Equation (1):

$$\begin{aligned} Tm = Ts = Tr = T_f = O(n) \\ Tc = O(\frac{n^3}{N}) \\ Tw = O(\frac{n^3}{N}) + O(n) \end{aligned} \quad (1)$$

V. EVALUATION OF THE PROPOSED METHOD

The proposed method was implemented in the Microsoft Visual Studio environment in the C# language, consisting of 4 clients. By installing a server on a system workstation, the public queue was created on the workstation so that the distribution process of matrices elements could be carried out. Then, clients were linked to each other through switches and hubs. Afterwards, we added the clients to the system domain and the IPs were set up. In the next step, middleware MSMQ was enabled in the server and clients.

We first obtained the time needed for matrices multiplication using just one workstation; then the multiplication operations were distributed using four workstations (clients). We considered the proposed method on a heterogeneous distributed environment using four workstations with different platforms. Table 1 shows the features of our distributed system workstations.

Table 1
Workstations of distributed system

Client	CPU	RAM	OS
Pc_1	2.4 GHz	2 GB	Windows 8
Pc_2	2.8 GHz	2 GB	Windows 7
Pc_3	3.2 GHz	4 GB	Windows 8
Pc_4	2.4 GHz	1 GB	Windows XP

Table 2 shows the time of multiplication of two matrices with different dimensions using (1) a single system and (2) the distributed system. Having obtained the results on four clients, we expanded the system; this led to a decrease of multiplication. As Table 2 shows, when the multiplication is carried out for two matrices with low size, the single system performance is better than the distributed one. This is because of the expense imposed by the data transmission in the network. However, the distributed system performance is getting better than a single system when matrices become larger. Moreover, a single system does not have the ability of multiplication of very large matrices; in this case, we must distribute operations of multiplication among workstations.

Figure 12 shows the performance of distribution system for matrix multiplication with different dimensions rather than a single computer.

Table 2
The time of computing matrices multiplication on a computer and on a distributed system

Matrices size	Multiplication time (milliseconds)					
		Distributed System				
	Single	W1	W2	W3	W4	Total
100	50	339	433	318	610	722
200	423	820	783	712	998	1058
300	1402	2200	2199	2185	2369	2590
400	3141	2531	2439	2398	2673	2945
500	7693	4119	4374	4021	4030	4833
600	13219	8543	9405	8195	8201	10122
700	21548	12569	14802	11986	12007	16121
800	34053	15634	14143	13690	16033	17180
900	48911	19654	18303	18452	23705	25303
1000	66789	30221	28413	27870	33129	35006
1100	107587	36112	35897	34998	44924	47585

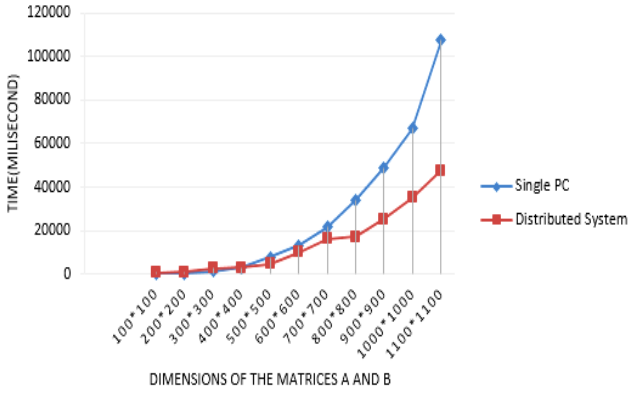

Figure 12: Distributed system performance for multiplication of matricies with different dimensions

Figure 13 shows performance timeline of each existing computer in the distributed system for matrix multiplication.

VI. PERFORMANCE WITH AN INCREASE OF CLIENTS

Table 3 shows the calculation time of matrix multiplication when the number of clients increases. As a whole, when a distributed system with the MSMQ middleware was used to multiply large matrices, the results indicated that using the proposed method reduces the time of multiplying large matrices suitably. We observed that our method enjoys more performance over the single workstation when we use matrices with high dimension. Equation (1) shows how time complexity reduced with the increase in the number of clients.

$$Tw = O\left(\frac{n^3}{N}\right) + O(n), N \approx n \\ \downarrow \\ Tw = O(n^2) + O(n) \quad (2)$$

Table 3
The multiplication time in terms of the number of clients of a distributed system

Matrix size	Multiplication time (in milliseconds)			
	4Ws	8Ws	16Ws	20Ws
100	722	810	892	903
200	1058	1050	1104	1223
300	2590	2358	2720	2915
400	2945	2703	3068	3214
500	4833	4171	4823	5148
600	10122	9123	10231	11531
700	16121	14983	16003	16841
800	17180	15200	15340	17039
900	25303	22391	21830	21966
1000	35006	32142	30963	30847
1100	47585	44189	42667	40952

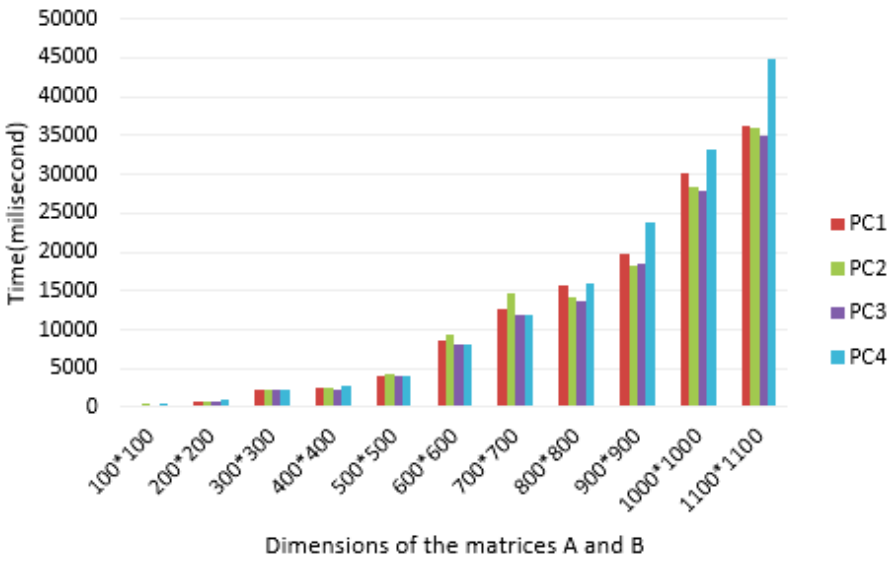

Figure 13: The timeline diagram of the performance of each computer in a distributed system for matrix multiplication.

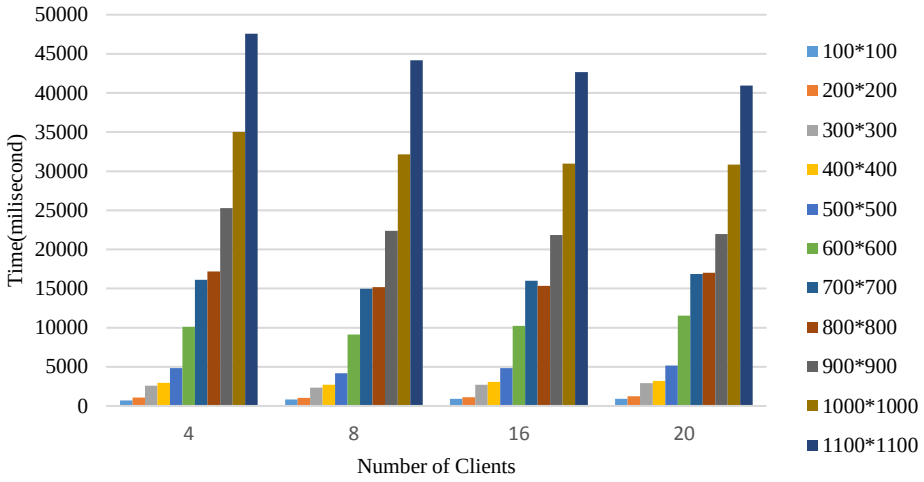

Figure 14: Charts for matrix multiplication distributed system with varying client displays.

VII. CONCLUSIONS AND FUTURE WORK

We used a distributed system with the MSMQ middleware to multiply large matrices. The results indicated that using the proposed method reduces time of multiplying large matrices. By increasing the dimensions of matrices, performance of our proposed method increases over a single workstation. This is because when the matrix size is getting higher, the time order $O(n^3)$ increases. By increasing the number of clients, the multiplication time is reduced. As the value of N (the number of clients) closes to the value of n (the number of matrix rows/columns), the time complexity of multiplication is reduced. Therefore, as the matrix gets larger, an increase in the number of clients will typically cause a better performance of our method. For future work, it is suggested that the various algorithms of matrix multiplication are implemented using our method and their performance is compared to a single workstation.

REFERENCES

[1] van Steen M, Tanenbaum AS, A brief introduction to distributed systems, Computing, 2016, 98, pp. 967-1009.

[2] Microsoft, Message Queuing Overview, https://msdn.microsoft.com/en-us/library/ms703216(v=vs.85).aspx, Access date: November, 12, 2016.

[3] Redkar A, Rabold K, Costall R, Boyd S, Walzer C, Pro MSMQ: Microsoft Message Queue Programming: Apress, 2004.

[4] Tichy WF, Parallel matrix multiplication on the connection machine, International Journal of High Speed Computing, 1989, 1, pp. 247-262.

[5] Beaumont O, Boudet V, Rastello F, Robert Y. Matrix-matrix multiplication on heterogeneous platforms. *International Conference on Parallel Processing*, 2000, pp. 289-298.

[6] Kattan A, Abdullah R, Salam RA. Reducing Feed-Forward Neural Network Processing Time Utilizing Matrix Multiplication Algorithms on Heterogeneous Distributed Systems. *First International Conference on Computational Intelligence, Communication Systems and Networks*, 2009, pp. 431-435.

[7] Ismail MA, Mirza S, Altaf T, Concurrent matrix multiplication on multi-core processors, International Journal of Computer Science and Security (IJCSS), 2011, 5, pp. 208.

[8] Yan Y, Kemp J, Tian X, Malik AM, Chapman B. Performance and Power Characteristics of Matrix Multiplication Algorithms on Multicore and Shared Memory Machines. *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 626-632.

APPENDIX

This is a view of our application environment for splitting matrices and sending split sections to MSMQ management in the server side. To consider the source code, refer to the http://www.conf.natanziau.ir/file/upload/Appendix.pdf.