Debugging of Parallel Programs using Distributed Cooperating Components

[1] Reema Mohammad Mrayyan, [2]Ahmad AbdulQadir Al Rababah.
ahd_68@yahoo.com
[1]Ministry of education, Telaa alali secondary school, Amman, Jordan.
[2]Faculty of Computing and Information Technology, King Abdulaziz University, Rabigh 21911, Saudi Arabia.

Summary
Recently, in the field of engineering and scientific and technical calculations, problems of mathematical modeling, real-time problems, there has been a tendency towards rejection of sequential solutions for single-processor computers. Almost all modern application packages created in the above areas are focused on a parallel or distributed computing environment. This is primarily due to the ever-increasing requirements for the reliability of the results obtained and the accuracy of calculations, and hence the multiply increasing volumes of processed data [2,17,41]. In addition, new methods and algorithms for solving problems appear, the implementation of which on single-processor systems would be simply impossible due to increased requirements for the performance of the computing system. The ubiquity of various types of parallel systems also plays a positive role in this process.

Simultaneously with the growing demand for parallel programs and the proliferation of multiprocessor, multicore and cluster technologies, the development of parallel programs is becoming more and more urgent, since program users want to make the most of the capabilities of their modern computing equipment[14,39]. The high complexity of the development of parallel programs, which often does not allow the efficient use of the capabilities of high-performance computers, is a generally accepted fact[23,31].

Keywords: *Debugging of Parallel Programs, Distributed Cooperating Components*

1. Introduction

At present, intensive research is being carried out in the field of automating the development of parallel programs, in particular, in the field of creating tools for debugging and researching parallel programs. These tools can be used for a variety of purposes. This is, first of all, the search for errors in the program, including such specific for parallel programs as errors in accessing procedures that ensure parallelism, errors in message transmission, synchronization errors, errors in accessing shared resources[15,19,44].

Parallel algorithms and programs, as a rule, are much more complex than sequential ones. Parallel programs are more difficult to debug, as they introduce new types of errors that are absent in sequential programs, caused by incorrect synchronization of processes or threads and incorrect use of tools that ensure parallelism[42]. Their non-deterministic behavior significantly complicates the debugging of parallel programs, since it makes it difficult to use the usual technique of gradual error localization by means of multiple program launches under the control of the debugger[1,9,17].

The complexity of creating high-quality tools for debugging and researching parallel programs is, on the one hand, a consequence of the specific problems of developing parallel programs, and, on the other hand, the developed debugger of parallel programs is also a parallel program that must interact with the debugged one[33,38], also parallel, which is even more, complicates the problem.

The problem of visualizing the results obtained is especially relevant for interactive debugging and research of parallel programs[16,28]. Unlike a simple sequential program, where there is one current point of program execution, and a variable has one value, in a parallel program there can be many, hundreds, or even thousands of execution points. The same program variable can also have many values - each process has its own[6,11,18]. The developers of tools for debugging and researching parallel programs are faced with the difficult task of presenting all the available information and providing tools for controlling the debugging process: on the one hand, in a form that is convenient, understandable for the user, and on the other hand, to give the user an exhaustive, complete picture of the behavior of the program under study. How to make this display convenient and compact, and at the same time to allow the user, if desired[32], to get access to all the details of interest to him - this is the task that a high-quality tool for debugging and researching parallel programs should solve.

Another characteristic difference between the process of debugging and researching a parallel program from similar actions with a sequential program is that, in a typical case, the parallel program is executed on a remote computer complex[5,20]. The program is executed on the computing nodes of the complex, to which the user usually does not have access. Therefore, the user can no longer influence the course of execution of individual processes for his program, in contrast to debugging a sequential program,

Manuscript received December 5, 2021
Manuscript revised December 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.12.79

which is typically executed on a computer directly at which the user sits[3,10,21]. Also, due to the fact that the investigated parallel program is executed on a remote complex, there is a problem of transferring the collected information about the program to the user's computer - in some cases; the amount of this information can be quite large[4,12,33].

2. Materials and Methods

The known approaches to debugging and researching parallel programs can be divided into three main areas: automatic control of the correctness of the program execution, comparative debugging (comparison of program execution at its various launches)[13], and interactive debugging. Automatic correctness control and comparative debugging can be carried out either by analyzing traces collected during the execution of a parallel program, or without using traces - dynamically in the real-time execution of a parallel program[22,43]. Dialogue debugging is usually carried out during the real execution of a parallel program by setting breakpoints, step through the program and inspect the values of the specified variables[46].

Debug prints

The simplest way to debug programs is to add additional print statements or output statements to a file in the program code[26,29]. These seals usually contain information on the basis of which the user determines from where in his program the print was made, and, possibly, the values of any variables at these points. Thus, you can track on which branches the program was executed and with what data. Based on this information, the user can track an operator or a group of operators leading the program to an incorrect state[5,36].

Dialogue debugging

In dialog debugging, a specially created tool is used - the debugger. When developing a dialog debugger, special attention is paid to its interface - the user should feel comfortable and convenient when working in the debugger environment[9,40,45]. All debugger functions and how to use them should be intuitive and, if possible, match those of a traditional sequential debugger.

During interactive debugging using the tools and tools provided by the debugger, the user defines one or more controlled points in the program[3,22,35]. These points can be breakpoints or watch points. The user gives the command to start or continue the program execution, and the program execution is interrupted at the first reached control point. The step-by-step mode of program execution is also possible and widely used. Further, the user analyzes the state of the program at the moment of shutdown. At this point, he can inspect the values of variables, look at the stack and parameters of subroutine calls, and use all the other tools and tools that a particular debugger provides him. If the state of the program is correct, the user continues its execution until the next control point, while it is possible to quickly adjust the further progress of debugging (set additional breakpoints, view the values of any other variables)[25,39]. Or, if the state of the program is already incorrect, the user defines additional monitored points and restarts the program in order to track the operator or a group of operators leading the program to an incorrect state.

Automatic correctness control

Automatic control of program correctness - checking additional correctness conditions in the process of its fulfillment. This check can be carried out both according to the previously obtained trace of the program execution, and in the process of real-time program execution. The conditions for the correctness of parallel programs are the correctness of calls to libraries that provide parallelism and message exchange, correct synchronization of processes and threads when using shared data, and so on. The advantages of automatic correctness control in comparison with traditional debugging methods is its complete automation, quality of debug diagnostics and the ability to detect a wide range of errors. However, the absence of diagnostics about errors when analyzing the correctness of the program does not guarantee the correctness of its operation, since the limited resources may not allow performing all the desired checks[24,30].

Comparative Debugging

In cases where the user has two versions of the program, one of which works correctly (reference), and the other does not (debuggable), a working version can be taken as a formal specification that is not working[37]. The idea of comparative debugging is precisely to compare the work of two versions of the same program, and at the same time the values of variables at certain controlled points of program execution are compared. Data for comparison can be taken both dynamically from running programs and from traces obtained during their execution. When debugging parallel programs, a sequential program is usually used as a reference program, which can often be debugged using standard tools, but a parallel program can also be taken as a reference. Comparative debugging allows you to detect differences caused by program changes, such as parallelization, as well as differences that arise when programs are porting from one platform to another or when the processor configuration is changed. The revealed differences indicate the errors in the program[9,47].

All of the above approaches to debugging and researching parallel programs have both their advantages and disadvantages. Even such a seemingly outdated approach as debug prints can be successfully applied in the absence of any other debugging tools on a particular

computer system. Dialogue debugging is good for its interactivity, the ability to quickly influence the course of debugging, and its proximity to traditional sequential program debuggers[2,37].

When using automatic trace-based debugging methods, the amount of data saved is usually very large, which can cause problems with disk space and the time required to save the data. In the process of searching for errors, due to the large volume and unstructured data, using only the tracer for debugging programs larger than test programs seems to be a rather tedious task. From this point of view, interactive debugging or dynamic debugging during program execution looks preferable. From this point of view, interactive debugging or dynamic debugging during program execution looks preferable[6,17].

However, during further debugging of the program, when questions of its efficiency, memory use, computing resources, etc. are considered, debugging along the path seems to be more preferable. First, when analyzing the temporal characteristics or the dynamic behavior of some components of the program relative to others, minimal intervention in the operation of the program under study is required. Secondly, this kind of research is usually carried out for the entire program in the entire time interval of its operation, and not for any specific fragment. Therefore, the use of a tracer for such studies seems to be the most acceptable option. But the huge volumes of the resulting traces greatly complicate the research process. Consequently, the obtained traces must be somehow automatically processed, highlighting and grouping the data of interest to the user, and also, if the user is interested in detailed information on some aspects of the program execution, show him not the entire trace, but only those events that satisfy the selected user criteria. And thus, we again come to the need for a dialogue with the user, only not in the process of executing the program, but later, in the process of examining the obtained traces[19,27].

Program instrumentation

It should be noted that, without exception, all approaches to debugging and researching programs are based on changing the program itself. Only by inserting additional statements intended to obtain debugging information can the debugger obtain any information about the program. This can be achieved by replacing the system or concurrency libraries with their own libraries, which provide the debugger with the information it needs, and then call the corresponding replaced library functions. Note that in this case, stopping and obtaining information for debugging is possible not at an arbitrary point in the program, but only at the entry and exit points of the functions of the replaced library. Or it can be program instrumentation - adding additional operators directly to the program itself. Operators can be added to the source code of the program, which will require recompilation, or to its object code. The addition and removal of operators in the object code can be done dynamically, during the execution of the program[2,31].

Dynamic instrumentation is undoubtedly preferable to static instrumentation in the case of an interactive debugger, because at the same time, only those parts of the program are changed in which breakpoints are currently set, in contrast to the static one, when it is necessary to control all possible breakpoints. Debuggers that use dynamic instrumentation also most often use all available platform-specific tools, which together provide the most efficient scheme for obtaining debug information with minimal interference with the program itself. Static instrumentation usually increases execution time significantly. But with a large number of controlled points, dynamic and static instrumentation can be comparable in efficiency. The low level of dynamic instrumentation and, as a consequence, the complexity of its implementation and support on several platforms make its development too difficult for small research teams. Therefore, the use of static instrumentation when creating prototypes of dialog debuggers and when creating automatic tools for debugging and researching programs is more than justified[13,25].

3. Experiments

The execution of parallel programs is most often performed on specialized computing facilities - clusters or multiprocessor systems with shared memory. Such computing facilities are usually accessed over a network via a host machine. The program is executed on specially designed computational nodes.

Most often, for security reasons, access to the user directly to the computing nodes is closed. Moreover, the computational nodes are not accessible from outside the computational unit, and they themselves cannot access any external machines. Thus, a running program communicates with the outside world only through the host machine, and the use of any traditional methods of dialog debugging on such computer systems is impossible.

General scheme of a distributed complex

To solve this problem, it is proposed to use an approach in which the running program itself reports all the information necessary for debugging about itself. Moreover, it reports them not directly to the user's computer, to which, as noted earlier, access from the computing node will most likely be impossible, but to the host machine, with which there should always be communication. And only then, from the host machine, the data will be transmitted to the user's computer. Thus, we come to the conclusion that it is necessary to have a special program that will be executed on the host machine and transfer data between the executing parallel program being debugged and the user's computer.

In the diagram below in Fig. 1, such a program is called Monitor. In addition to simple data transfer functions, the Monitor can perform a number of communication and logical functions: it must serve all parallel running processes of the program being debugged, perform various reduction and logical actions with many of these processes, make decisions about suspending or continuing the execution of the program, and access the user's computer only in case of any expected events. Thus, the Monitor minimizes network traffic going through the global network to the user's computer, which has a beneficial effect on minimizing time losses when debugging and examining programs.

When building a distributed software package for debugging and researching remotely executing parallel programs, it is proposed to apply the following scheme of distributed components and interactions between them as shown on Fig. 1.

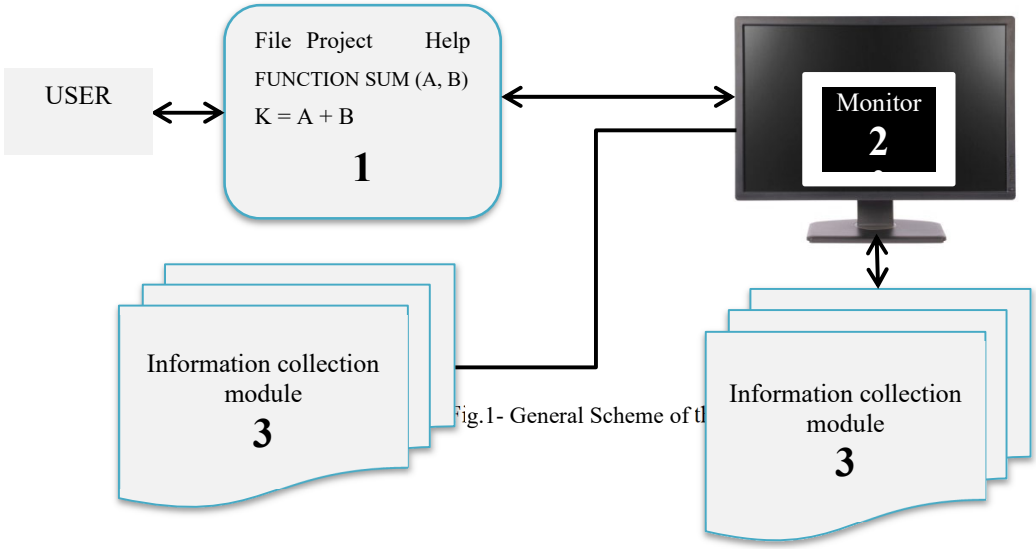

Fig.1- General Scheme of the distributed complex

User interface

The user interface is running on the user computer (1). It is a program with a user-friendly graphical interface, made in the style of Microsoft Visual Studio, which demonstrates the progress of the program being debugged or the results of the program examination, and provides a convenient interface to all functions and tools of the debugger. The user interface does not have any information about the program under study, but receives all the information for visualizing the debugging and research process from the Monitor, and sends all commands and user requests to it, without any preliminary processing.

The user interface operates with general concepts: "show the source code: file such and such a line such and such", "a breakpoint has been reached by such and such a process in such and such a place in the source code", etc. The commands, the execution of which is implemented in the User Interface, also do not depend on the type of the program being debugged, but depend only on the current state of the program (running, paused) and are divided by purpose into typical for sequential debuggers (start, pause, take a step, show the value of an expression, set a breakpoint, etc.), specific to parallel programs (show the state of processes, start or take a step for a selected group of processes, show variables in individual processes, etc.), as well as commands of the MPI program research system (start the analyzer, show the result of its work, show a fragment of the trace etc.).

Monitor

This is the main program in the complex; it is responsible for establishing communication between all components of the complex and controls the execution of the program being debugged or the launch of its research tools. The monitor also initially does not have any information about the running program, but receives it from concurrently running processes (instances) of the program being debugged or from tools for studying the program. Then, depending on the type of the program being debugged, or on the type of activity (debugging or research on traces), The monitor enters one of the predefined modes of operation, in which it receives messages received from the processes being debugged (or from the program research tool) and from the User Interface, processes them, and sends

messages in response. Thus, the Monitor is already dependent on the type of program being debugged, and even the same primitives are processed differently when debugging different types of programs. Besides, the Monitor also contains other language-dependent information, for example, about the way of data distribution in the NORM-program, taking into account which the Monitor calculates the values of expressions.

Information collection modules

These modules are responsible for the direct collection of information about the debugged process or the program under study. Depending on the type of the task at hand, the modules for collecting information can be performed in different ways. If information is collected directly during program execution, then it must be a library of functions linked to the program being executed. The functions of this library are called directly from each process of the program being debugged. In the case when the program is analyzed on the basis of pre-assembled traces, the Information Acquisition Modules represent a separate program that performs the analysis. In both cases, before starting their work, the Information Collection Modules establish a connection with the Monitor and provide it with information about themselves, their type and characteristics of the debugged process or objects of research. Then, in the case of collecting information directly during the execution of the program, upon the occurrence of various events (execution of a certain operator by the process, call of a function, etc.), the Information Collection Modules can inform the Monitor about this, and wait from him for instructions and various requests. In this case, the execution of the program under study is suspended. Then, at the direction of the monitor, execution of the suspended program is resumed.

Software package

The software package for creating tools for debugging and investigating parallel programs in the interactive mode was implemented in full accordance with the above and described scheme of distributed components.

4. Implementations and Discussions

The non-procedural language Norma is designed to automate the solution of grid problems on computing systems with a parallel architecture. This language allows you to exclude the programming phase, which is necessary in the transition from the calculation formulas given by the applied specialist to the program. Calculation formulas are written in the Norma language in a mathematical form familiar to an applied specialist, and then the NORMA language compiler generates a program in the "traditional" programming language - FORTRAN or C.

When constructing an output parallel program in the message transfer model for parallel systems with a distributed architecture, the compiler automatically determines the structure of the output program according to the Norma program, distributes data and their processing over a given number of virtual processors, generates operators for counting, calculating, transferring data between parallel running processes. The generated program can also call subroutines and functions written in Fortran or C by the user himself. When executing programs in the NORMA language, parallel programs automatically generated by the compiler in Fortran MPI or C MPI languages are actually executed. It would probably be wrong to offer an applied specialist who has compiled a program in the NORMA language to debug it by debugging a parallel generated unfamiliar program in another language. Debugging programs in the NORMA language at the source code level and in terms of the NORMA language looks much more preferable.

Thus, the main task of debugging programs in the NORMA language can be formulated as follows: to implement the possibility of interactive debugging of declarative specifications in the NORMA language in terms of the NORMA language, despite the fact that the generated parallel program in the FORTRAN MPI or C MPI language is actually executed.

When generating executable parallel programs, the translator from the NORMA language also generates for the debugger all the necessary information about the conversion of NORMA language operators into FORTRAN or C language operators, about the parallelization performed, about the distribution of data among processes, etc. All these data are transferred to the Information Acquisition Module, which is made in the form of a library of functions (this library was called the Library of Communication with the Monitor), by generating calls to these library functions in the executable program. In this case, all the necessary information is passed through the actual parameters of the function calls.

Generating start / end blocks

At the beginning of the program, after calling MPI_Init (...), a function call from the Monitor Link Library is inserted, this records the start of the program execution. The following parameters are passed to it: the name of the main section of the program; the name of the file with the source code of the program and its checksum calculated during the broadcast; line number in the file from which the main section of the program begins. Exit from the MPI_Init (...) function means that all the necessary MPI instances of this program have been launched and initialized, connections are established between them. You can start debugging. At the end of the program, before calling MPI_Finalize (...), a call to a function from the Monitor

Link Library is inserted, this records that the execution of the program ends.

Generating the start of statement execution

Before each operator in the generated program, if it is the first operator in the group of operators that implement a certain construction of the original NORM-program, a call to the function of the Library of Communication with the Monitor is inserted, this fixes the beginning of execution of the construction of the original NORM-program. In this case, information about the line number in the source file is transmitted, with which the construction of the original NORM-program begins and ends. Based on this information, the debugger determines whether a breakpoint has been reached and displays the current position of the process in the source code of the program.

Generating variable registration

In order for the debugger to have access to the values of the variables, in each function of the generated program, before the first executable statement, calls to the Monitor Link Library function are inserted, which registers one variable declared in this section. The number of calls is inserted as many as the variables declared in this section - one for each variable. The following are passed as actual parameters: the name of the variable, the address of its beginning in memory, the type of the variable, the dimension, the names of the indices, the original ranges for each index and, if the variable has been distributed among the processes, the ranges of the variable description in the generated program. Based on this information, the debugger, using direct access to an address in memory, can get the value of a variable at any point in its description.

When debugging a generated program in Fortran MPI or C MPI, it is necessary to transform objects that the user operates - objects of the NORM-program - into objects of the generated debugged program. This must be done when processing various users requests, as well as when displaying the results of queries, perform the reverse transformation. Using this transformation, it is achieved that the user operates with the objects of the source program in the NORMA language and receives information in terms of those objects that he himself created in his program.

To implement support for external modules in the FORTRAN MPI language in the debugger, the debugger scheme was used and slightly expanded, which was implemented when debugging NORM-programs: the program code of external modules written by the programmer in FORTRAN MPI is instrumented. At the same time, operators of calling the functions of the Library of communication with the Monitor are inserted into the text of the program. The monitor, as in the case of the NORM-program, collects information from all processes and sends / receives commands from the User interface. Only in this case, the Monitor no longer performs any transformations - when debugging, the user works in the context of a Fortran MPI program written by him.

Dialogue interface for the MPI program research system

To study an MPI program in a DVM system, the MPI program is built with a special tracer library. Then, during program execution, all MPI calls, their parameters, etc. are saved as trace files. After the completion of the program, the resulting traces can be examined both visually (but, as noted earlier, in real problems the volume of traces is very significant, which makes it extremely difficult to visualize the traces), and special programs - a correctness analyzer, an efficiency analyzer. As a result of the analyzer operation, a textual protocol is obtained, which lists all found and potential errors, timing characteristics, etc.

When using this system for debugging and examining MPI programs, the user works with different types of text files: files with source program texts, files with trace events, files-logs of analyzers. In this case, there is a logical connection between the contents of various files. But, since the work takes place with simple text files, the user, when he wants to look at the elements of other files related to the information of interest to him, is forced to independently open the necessary file and find the information he needs in it.

Meanwhile, as you can see, all the data for automating this process is available in the files themselves. Moreover, trace files initially have a binary format, which can allow reading and interpreting only the information that is of interest to the user, and not the entire huge trace. The analyzers also have a library interface that allows you to develop programs that receive information directly from the analyzers in a binary, structured form, bypassing its textual representation.

To create a complex that allows the user from his computer to conduct a dialogue with the System for researching MPI programs operating on a remote computer complex, it was decided to use the general scheme of organization of a distributed complex proposed in the second chapter for debugging and researching remotely executing parallel programs.

5. Results and Conclusions

The created complex, which is an addition to the Tools for debugging MPI-programs in the DVM-system, was named "Dialogue interface for the system for researching MPI-programs". The user interface and the Monitor, developed for the Parallel Programs Debugger in the NORMA language with Fortran MPI support, have been modified to solve new specific tasks of the Dialogue Interface being created for the MPI program research

system. In the User Interface, new types of windows were implemented to present information in a convenient, structured form and navigation tools were implemented both within windows and between windows of various types, means of partial request for information. In the Monitor, a scheme for establishing communication and exchange of information with a specially created program was implemented, which plays the role of an Information Gathering Module in the general scheme of a distributed complex for debugging and researching remotely executing parallel programs. This program, called the Dialogue Analyzer, was created using the existing libraries of access to the trace files and the libraries of the analyzers of the Debugging Tools for MPI programs in the DVM system, and using the component link library to communicate with the Monitor.

The dialog interface for the MPI program research system has demonstrated fast and stable operation when using all its functions and tools. Regardless of the used hardware and software architecture of the computing complex, on which the study of the obtained traces was carried out, and the characteristics of the communication channel with the remote computing unit, the operation of the complex was distinguished by stability and quick response to all user requests.

It is provided technical information about the software implementation of the created distributed software package for creating debugging tools and researching parallel programs in the interactive mode. The data on the extensibility and modifiability of the components of the complex are presented, the complexity of the development of new tools for debugging and researching remotely running parallel programs based on the created software complex is assessed.

Main results of work

- A diagram of distributed interacting components of a software package has been developed for creating debugging tools and researching parallel programs in an interactive mode.
- A software package has been created for the implementation of tools for debugging and researching parallel programs in an interactive mode.
- An interactive debugger for programs written in the declarative non-procedural NORMA language has been developed, with support for debugging external modules written in Fortran MPI.
- An interactive debugger for programs written in Fortran MPI has been developed.
- A dialogue interface for the MPI-programs research system has been developed.

REFERENCES

[1]Anduela Lile, “Analyzing E-Learning Systems Using Educational Data Mining Techniques,” Mediterr. J. Soc. Sci., vol. 2, no. 3, pp. 403-419, 2011.

[2]F. Castro, A. Vellido, À. Nebot, and F. Mugica, “Applying Data Mining Techniques to e-Learning Problems,” Studies in Computational Intelligence (SCI) vol. 62, no. 221, pp. 183–221, 2007.

[3]Romero, Cristobal, and Sebastian Ventura, eds. “Data mining in elearning,” WIT Press, Vol. 4, 2006.

[4] L.Behari, A. AlRababah. "Enhancing Educational Data Mining based ICT Competency among e-Learning Tutors using Statistical Classifier" International Journal of Advanced Computer Science and Applications (IJACSA), Volume 11 Issue 3 March 2020.

[5] A. AlRababah. "Neural Networks Precision in Technical Vision Systems" IJCSNS International Journal of Computer Science and Network Security, VOL.20 No.3, March 2020.

[6] Ahmad A. AlRababah "Assurance Quality and Efficiency in Corporate Information Systems", IJCSNS International Journal of Computer Science and Network Security, VOL.19 No.4, April 2019.

[7] B. Alrami, A. AlRababah. "Information Protection Method in Distributed Computer Networks Based on Routing Algorithms" IJCSNS International Journal of Computer Science and Network Security, VOL.19 No.2, February 2019.

[8] Ahmad AbdulQadir AlRababah, Ahmad Alzahrani. "Software Maintenance Model through the Development Distinct Stages", IJCSNS International Journal of Computer Science and Network Security, VOL.19 No.2, February 2019.

[9] L. Jiang, H. Zhang, and Z. Cai, “A Novel Bayes Model : Hidden Naïve Bayes,” IEEE Transaction on Knowledge and Data Engineering, vol. 21, no. 10, pp. 1361–1371, 2009. doi:http://dx.doi.org/10.1109/TKDE.2008.

[10] S. Taheri and M. Mammadov, “Learning the naive bayes classifier with optimization models,” Int. J. Appl. Math. Comput. Sci., vol. 23, no. 4, pp. 787–795, 2013. doi:http://dx.doi.org/10.2478/amcs-2013-0059.

[11] W. Zhang and F. Gao, “An improvement to naive bayes for text classification,” Procedia Eng., vol. 15, pp. 2160–2164, 2011.

[12]S. Banga, S. Mongia, S. Dhotre, and I. Introduction, “Regression And Augmentation Analytics on Earth ’ s Surface Temperature,” vol. 5, no. 3, pp. 17–19, 2017.

[13]X. Wu et al.,“ Top 10 algorithms in data mining,” Springer-Verlag London, vol. 14, no. 1. 2008. doi:http://dx.doi.org/doi:10.1007/s10115- -0114-007 .2

[14] AlRababah A. "Data Flows Management and Control in Computer Networks", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 9, No. 11, 2018.

[15] A.A. AlRababah. "Problems Solving of Cell Subscribers based on Expert Systems Neural Networks" International Journal of Advanced Computer Science and Applications (IJACSA), 10(12), 2019

[16] A.AbdulQadir AlRababah. "Implementations of Hybrid FPGA Microwave Format Extension as a Control Device", IJCSNS International Journal of Computer Science and Network Security, VOL.18 No.11, November 2018.

[17] Ahmad AlRababah. "Assurance Quality and Efficiency in Corporate Information Systems", IJCSNS International Journal of Computer Science and Network Security, VOL.19 No.4, April 2019.

[18] A. Choi, N. Tavabi, and A. Darwiche, "Structured features in naïve bayes classification," 30th AAAI Conf. Artif. Intell. AAAI, 2016.

[19]Zhang H., "The Optimality of Naive Bayes," 2004. American Association for Artificial Intelligence. 2004.

[20]T. Calders and S. Verwer, "Three naive Bayes approaches for discrimination-free classification," Data Min. Knowl. Discov., vol. 21, no. 2, pp. 277–292, 2010

[21] R. Y. M. Li and H. C. Y. Li, "Have housing prices gone with the smelly wind? Big data analysis on landfill in Hong Kong," Sustain., vol. 10, no , 2.pp. 1–19, 2018.

[22]Boyd, Danah and Crawford, Kate, "Six Provocations for Big Data," A Decade in Internet Time: Symposium on the Dynamics of the Internet and Society, September 2011.

[23]K. Swan and L. F. Shih, "on the Nature and Development of Social Presence in Online Course Discussions," Online Learn., vol. 9, no. 3, pp–115. .2019 ,136

[24]Segaran, Toby, and Jeff Hammerbacher, "Beautiful data: the stories behind elegant data solutions," O'Reilly Media, Inc., p. 257, 2009.

[25]S. O. Material, S. Web, H. Press, N. York, and A. Nw, "The World ' s Technological Capacity," vol. 60, no. 2011, pp. 60–66, 2014.

[26] Ahmad AbdulQadir. "DIGITAL IMAGE ENCRYPTION IMPLEMENTATIONS BASED ON AES ALGORITHM", VAWKUM Transactions on Computer Sciences, Volume 13, Number 1, May-June , 2017, Pages: 1-9.

[27] A.A. Alrababah. "Implementation of Software Systems Packages in Visual Internal Structures" , Journal of Theoretical and Applied Information Technology, Volume 95, Issue 19 (2017), Pages: 5237-5244.

[28] AlRababah A. A. "A New Model of Information Systems Efficiency based on Key Performance Indicator (KPI)" (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 3, 2017.

[29] Ahmad A. Rababah. "On the associative memory utilization in English- Arabic natural language processing", International Journal of Advanced and Applied Sciences, Volume 4, Issue 8 (August 2017), Pages: 14-18

[30] Ahmad A.Al. "Lempel - Ziv Implementation for a Compression System Model with Sliding Window Buffer", International Journal of Advanced Computer Science and Applications (IJACSA), Volume 6, Issue10, 2015.

[31] A. AlRababah, Ali AlShahrani, Basil Kasasbeh. "Efficiency Model of Information Systems as an Implementation of Key Performance Indicators", IJCSNS International Journal of Computer Science and Network Security, December 2016 Vol. 16 No. 12 pp. 139-143,

[32] P. Larrañaga, H. Karshenas, C. Bielza, and R. Santana, "A review on evolutionary algorithms in Bayesian network learning and inference tasks," Inf. Sci. (NY)., vol. 233, pp. 109–125, 2013.

[33] M. E. Maron, "Automatic Indexing: An Experimental Inquiry," J. ACM, vol. 8, no. 3, pp. 404–417, 1961. doi: http://dx.doi.org/10.1145 .321075.321084

[34] Rish, Irina. "An empirical study of the naive Bayes classifier." IJCAI 2001workshop on empirical methods in artificial intelligence. Vol. 3. No. 22. 2001.

[35] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," ACM Int. Conf. Proceeding Ser., vol. ,148pp. 161–168, 2006.

[36]Keerthi S. S., Shevade S. K., Bhattacharyya C., and Murthy K. R. K",Improvements to Platt's SMO Algorithm for SVM Classifier Design", Neural Computation, 13: 637-649, 2001.

[37]S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to the SMO algorithm for SVM regression," IEEE Trans. Neural Networks, vol. 11, no. 5, pp. 1188–1193, 2000.

[38] A.A. Rababah. "Embedded Architecture for Object Tracking using Kalman Filter", *Journal of Computer Science,* Volume 12, Issue 5, Pages 241-245, 2016. Science Publications, SCOPUS, DOI: 10.3844/jcssp.2016.241.245

[39] Ahmad AlRababah. "A new dynamic Model for software testing quality" journal of applied sciences

and engineering technology, Elsevier (SCOPUS), Vol. 7, (1): 191-197, 2014.

[40] Tagreed Altamimi, A.AlRababah, Najat Shalash. “A New Model for Software Engineering Systems Quality Improvement”. Research Journal of Applied Sciences, Engineering and Technology, Elsevier (SCOPUS), 7(13): 2724-2728, 2014.

[41] Ofeishat H. AlRababah A. "Real-time programming platforms in the mainstream environments", International Journal of Computer Science and Network Security (IJCSNS), , Vol.9 No1,pp.197-204, 2009. ISSN : 1738-7906

[42] Ranjit Biswas and Ahmad AlRababah. "Rough Vague Sets in an Approximation Space". INTERNATIONAL JOURNAL OF COMPUTATIONAL COGNITION (HTTP://WWW.IJCC.US, VOL. 6, NO. 4, DECEMBER 2008, pp.60-63.

[43] Nabeel Banihani and A.AlRababah. “Component Linked Based System”, The VI international conference- “Modern problems of radio engineering, telecommunications and computer science”, IEEE, pp: 405-407, 2004.

[44] Mohammad AlRababah and Ahmad AlRababah. "Module Management Tool in Software Development Organizations", Journal of Computer Science 3 (5): 318-322, © 2007 Science Publications,

[45] M.AlRababah and A.AlRababah. "Functional Activity IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.1, January, 2007 , pp. 153-158

[50]N. Matić, I. Guyon, L. Bottou, J. Denker, and V. Vapnik, “Computer aided cleaning of large databases for character recognition,” Proc. - Int. Conf. Pattern Recognit., vol. 2, pp. 330–333, 1992.

[46]Ali Z, Shahzad SK, and Shahzad W. Performance analysis of support vector machine based classifiers. International Journal of Advanced and Applied Sciences, 5(9): 33-38, 2018.

[47]C. Jensen, M. Kotaish, A. Chopra, K. A. Jacob, T. I. Widekar, and R. Alam, “Piloting a Methodology for Sustainability Education: Project Examples and Exploratory Action Research Highlights,” Emerg. Sci. J, vol. 3, no. 5, pp. 312–326, 2019. doi:http://dx.doi.org/10.28991/esj2019-01194.