WJCMS

Wasit Journal of Computer and Mathematics Science

Journal Homepage: https://wjcm.uowasit.edu.iq/index.php/WJCM

e-ISSN: 2788-5879 p-ISSN: 2788-5879

Features and characters for distributed system

Prof. Dr. Paul Mccullagh[1]*

[1]School of Computing, Ulster University, United Kingdom.

*Corresponding Author: Prof. Dr. Paul Mccullagh

DOI: https://doi.org/10.31185/wjcm.89
Received: September 2022; Accepted: November 2022; Available online: December 2022

ABSTRACT: The introduction of microprocessors had created a number of product options that were just not there before. These clever processors have infiltrated and dispersed themselves across every aspect of our life, whether it be in the office (fax machines, pagers, laser printers, credit card readers), living rooms (televisions, air conditioners), or kitchens (food processors, microwave ovens). The usage of an operating system has many benefits as distributed applications get more complicated. The real-time demands of the majority of distributed systems need the usage of Real Time Operating Systems (RTOS) that can fulfill the needs of distributed systems. Real-time applications can be readily built and expanded thanks to RTOS. Through the division of the application code into different files, the use of RTOS streamlines the design process.

Keywords: OS, Real time, embedded system, distributed system

1. INTRODUCTION

A distributed system is a specialized computer system that is part of a larger system or machine. Distributed systems can also be thought of as information processing subsystems integrated in a larger system [1]. As part of a larger system, it largely determines its functionality. An distributed system usually contains an distributed processor. Many appliances that have a digital interface (microwaves, cars) utilize distributed systems. Distributed systems also can be defined as computing systems with tightly coupled hardware and software that are designed to perform a dedicated function [2]. This combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device [3]. The word distributed reflects the fact that these systems are usually an integral part of a larger system. Some distributed systems include an operating system. Others are very specialized resulting in the entire logic being implemented as a single program. These systems are distributed into some device for some specific purpose other than to provide general purpose computing. A typical distributed system is shown in figure [1].

Tracing back the history, the birth of microprocessor in 1971 marked the booming of digital era. Early distributed applications included unmanned space probes, computerized traffic lights and aircraft flight control systems. In the 1980s, distributed systems brought microprocessors into every part of our personal and professional lives. Presently there are numerous gadgets coming out to make our life easier and comfortable because of advances in distributed systems [2, 4]. Distributed systems vary considerably. Some are general-purpose computers, running standard operating systems—such as UNIX—with special-purpose applications to implement the functionality. Others are hardware devices with a special-purpose distributed operating system providing just the functionality desired. Yet others are hardware devices with application-specific integrated circuits (ASICs) that perform their tasks without an operating system [5].

There are over 3 billion distributed CPUs sold each year. Distributed CPUs are growing at a faster rate than desktop processors. A large part of this growth is in smaller (4-, 8-, and 16-bit) CPUs and digital signal processors (DSPs) [1].

Distributed systems provide several functions, some of these functions are:

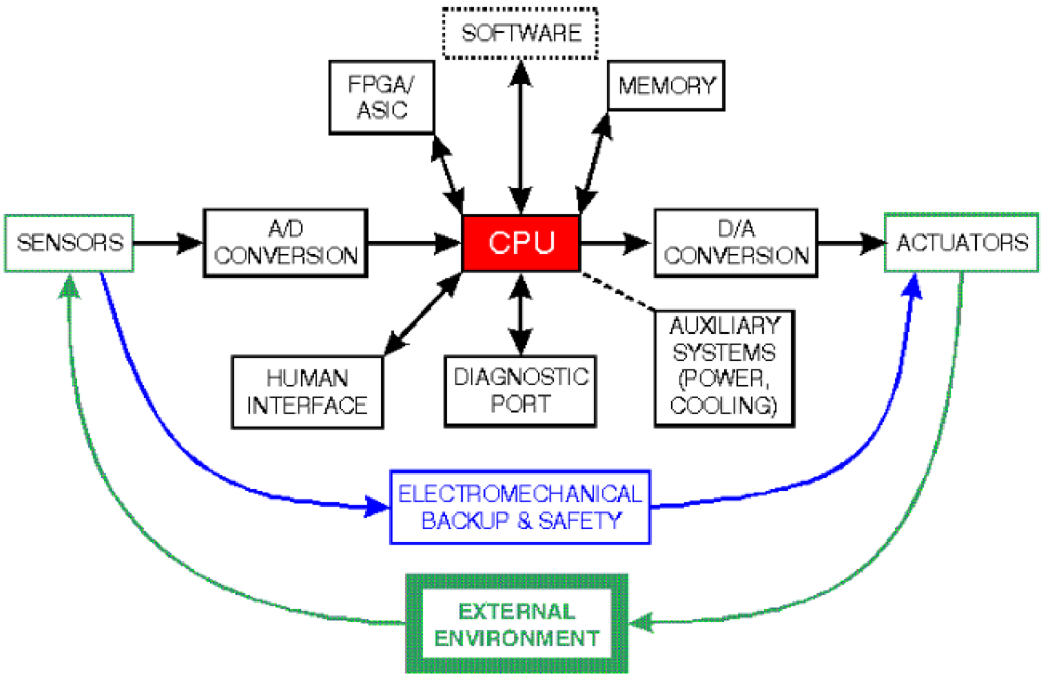

FIGURE 1. **A Typical Distributed System**

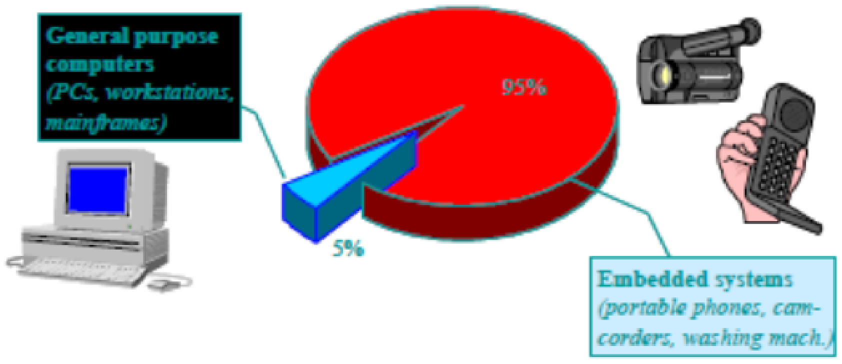

FIGURE 2. **Distributed systems dominate the microprocessor landscape**

- **Monitor the environment;** distributed systems read data from input sensors. This data is then processed and the results displayed in some format to a user or users.
- **Control the environment;** distributed systems generate and transmit commands for actuators.
- **Transform the information;** distributed systems transform the data collected in some meaningful way, such as data compression/decompression.

Although interaction with the external world via sensors and actuators is an important aspect of distributed systems, as show in Figure, these systems also provide functionality specific to their applications. Distributed systems typically execute applications such as control laws, finite state machines, and signal processing algorithms. These systems must also detect and react to faults in both the internal computing environment as well as the surrounding electromechanical systems [1].

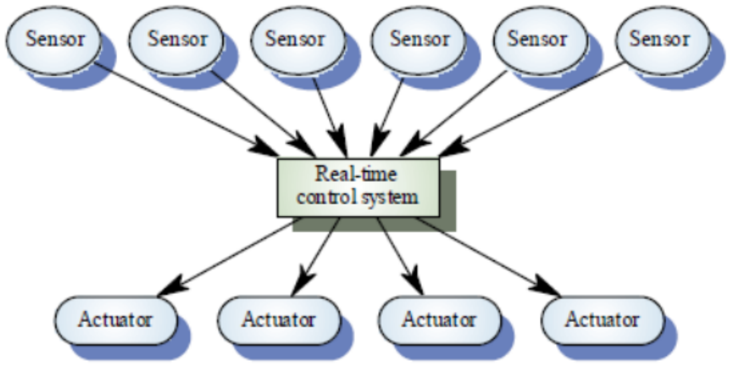

FIGURE 3. **Sensors and Actuators in an Distributed System**

There are many categories of distributed systems, from communication devices to home appliances to control systems. Examples of distributed system include;

- Communication devices (modems, cellular phones)
- Home Appliances (CD player, microwave oven)
- Control Systems (Automobile anti-lock braking systems, robotics, and satellite control).

2. CHARACTERISTICS OF DISTRIBUTED SYSTEMS

Distributed systems are characterized by a unique set of characteristics. Each of these characteristics imposed a specific set of design constraints on distributed systems designers. The challenge to designing distributed systems is to conform to the specific set of constraints for the application.

2.1 APPLICATION SPECIFIC SYSTEMS

Distributed systems are not general-purpose computers. Distributed system designs are optimized for a specific application. Many of the job characteristics are known before the hardware is designed. This allows the designer to focus on the specific design constraints of a well defined application. As such, there is limited user reprogram ability. Some distributed systems, however, require the flexibility of reprogram ability. Programmable DSPs are common for such applications.

2.2 REACTIVE SYSTEMS

As mentioned earlier, a typical distributed systems model responds to the environment via sensors and control the environment using actuators. This requires distributed systems to run at the speed of the environment. This characteristic of distributed system is called “reactive”. Reactive computation means that the system (primarily the software component) executes in response to external events. External events can be either periodic or aperiodic. Periodic events make it easier to schedule processing to guarantee performance. Aperiodic events are harder to schedule. The maximum event arrival rate must be estimated in order to accommodate worst case situations. Most distributed systems have a significant reactive component. One of the biggest challenges for distributed system designers is performing an accurate worst case design analysis on systems with statistical performance characteristics (e.g., cache memory on a DSP or other distributed processor). Real time system operation means that the correctness of a computation depends, in part, on the time at which it is delivered. Systems with this requirement must often design to worst case performance. But accurately predicting the worst case may be difficult on complicated architectures. This often leads to overly pessimistic estimates erring on the side of caution. Many distributed systems have a significant requirement for real time operation in order to meet external I/O and control stability requirements.

2.3 DISTRIBUTED SYSTEMS

A common characteristic of an distributed system is one that consists of communicating processes executing on several CPUs or Application-specific integrated circuits (ASICs) which are connected by communication links. The reason for this is economy. Economical 4 8-bit microcontrollers may be cheaper than 32 bit processors. Even after adding the cost of the communication links, this approach may be preferable. In this approach, multiple processors are usually required to handle multiple time-critical tasks. Devices under control of distributed systems may also be physically distributed. **4. Heterogeneous Architectures** Distributed systems often are composed of heterogeneous architectures, [1].

They may contain different processors in the same system solution. They may also be mixed signal systems. The combination of I/O interfaces, local and remote memories, and sensors and actuators makes distributed system design truly unique. Distributed systems also have tight design constraints, and heterogeneity provides better design flexibility.

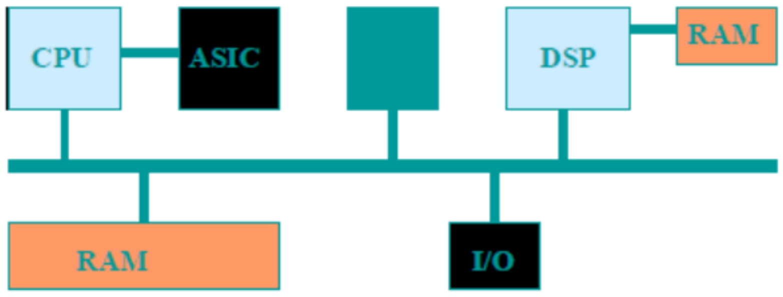

FIGURE 4. **Distributed Systems have Heterogeneous Architectures.**

2.4 HARSH ENVIRONMENT

Many distributed systems do not operate in a controlled environment. Excessive heat is often a problem, especially in applications involving combustion (e.g., many transportation applications). Additional problems can be caused for distributed computing by a need for protection from vibration, shock, lightning, power supply fluctuations, water, corrosion, fire, and general physical abuse. For example, in the Mission Critical example application the computer must function for a guaranteed, but brief, period of time even under non-survivable fire conditions. These constraints present a unique set of challenges to the distributed system designer, including accurately modeling the thermal conditions of these systems.

2.5 SYSTEM SAFETY AND RELIABILITY

As distributed system complexity and computing power continue to grow, they are starting to control more and more of the safety aspects of the overall system. These safety measures may be in the form of software as well as hardware control. Mechanical safety backups are normally activated when the computer system loses control in order to safely shut down system operation. Software safety and reliability is a bigger issue. Software doesn't normally "break" in the sense of hardware. However software may be so complex that a set of unexpected circumstances can cause software failures leading to unsafe situations. The challenges for distributed designers include designing reliable software and building cheap, available systems using unreliable components. The main challenge for distributed system designers is to obtain low-cost reliability with minimal redundancy.

2.6 CONTROL OF PHYSICAL SYSTEMS

One of the main reasons for embedding a computer is to interact with the environment. This is often done by monitoring and controlling external machinery. Distributed computers transform the analog signals from sensors into digital form for processing. Outputs must be transformed back to analog signal levels. When controlling physical equipment, large current loads may need to be switched in order to operate motors and other actuators. To meet these needs, distributed systems may need large computer circuit boards with many non-digital components. Distributed system designers must carefully balance system tradeoffs among analog components, power, mechanical, network, and digital hardware with corresponding software.

2.7 SMALL AND LOW WEIGHT

Many distributed computers are physically located within some larger system. The form factor for the distributed system may be dictated by aesthetics. For example, the form factor for a missile may have to fit inside the nose of the missile. One of the challenges for distributed systems designers is to develop non-rectangular geometries for certain solutions. Weight can also be a critical constraint. Distributed automobile control systems, for example, must be light weight for fuel economy. Portable CD players must be light weight for portability purposes.

2.8 COST SENSITIVITY

Cost is an issue in most systems, but the sensitivity to cost changes can vary dramatically in distributed systems. This is mainly due to the effect of computer costs have on profitability and is more a function of the proportion of cost changes compared to the total system cost.

2.9 POWER MANAGEMENT

Distributed systems have strict constraints on power. Given the portability requirements of many distributed systems, the need to conserve power is important to maintain battery life as long as possible. Minimization of heat production is another obvious concern for distributed systems.

3. STRUCTURE OF DISTRIBUTED SYSTEMS

All distributed systems contain a processor and software. The processor may be 8051 micro-controller or a Pentium-IV processor (having a clock speed of 2.4 GHz). Certainly, in order to have software there must be a place to store the executable code and temporary storage for run-time data manipulations. These take the form of ROM and RAM respectively. If memory requirement is small, it may be contained in the same chip as the processor. Otherwise one or both types of memory will reside in external memory chips. All distributed systems also contain some type of inputs and outputs, see Figure [2]. For example in a microwave oven the inputs are the buttons on the front panel and a temperature probe and the outputs are the human readable display and the microwave radiation. Inputs to the system generally take the form of sensors and probes, communication signals, or control knobs and buttons. Outputs are generally displays, communication signals, or changes to the physical world.

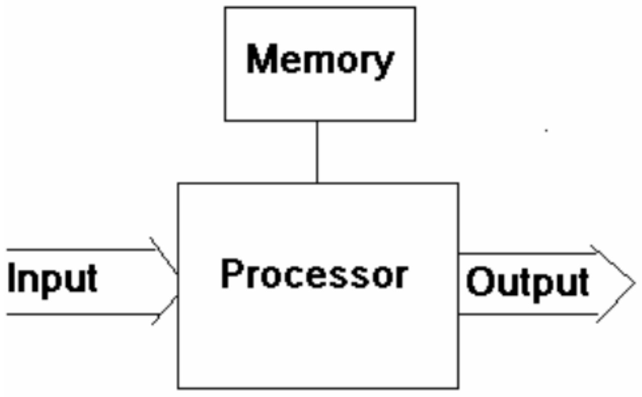

FIGURE 5. **Generic Distributed system.**

Real-time scheduling is the process of creating start and finish times for sets of tasks such that all timing, precedence, and resource constraints are met. Real-time scheduling results in recent years have been extensive. Theoretical results have identified worst-case bounds for dynamic on-line algorithms, and complexity results have been produced for various types of assumed task-set characteristics [6]. Memory management capabilities are necessary in some systems to provide memory protection and virtual memory. Special purpose interfaces are also needed to support a variety of external peripheral devices, energy consumption control, and so on. Commonly both the hardware and the software for an distributed system are developed in parallel. Constant design feedback between the two design teams should occur in this development model. The result is that each side can take advantage of what the other can do. The software component can take advantage of special hardware features to gain performance. The hardware component can simplify module design if functionality can be achieved in software that reduces overall hardware complexity and cost. As real-time distributed systems became more complex and software advanced from structured programming to object-oriented methodologies, new modeling tools were needed. Unified Modeling Language (UML) was developed in response to the need for a standardized object modeling language. UML can be adapted to design a variety of real-time systems, from small 8-bit microcontroller systems to large multi-processor networked systems. UML includes features for modeling functionality, objects, states, design patterns and extensibility features [7].

One class of distributed processors focuses on size, power consumption, and price. Therefore, some distributed processors are limited in functionality, i.e., a processor is good enough for the class of applications for which it was designed but is likely inadequate for other classes of applications. This is one reason why many distributed processors do not have fast CPU speeds. For example, the processor chosen for a personal digital assistant (PDA) device does not have a floating-point co-processor because floating-point operations are either not needed or software emulation is sufficient. The processor might have a 16-bit addressing architecture instead of 32- bit, due to its limited memory storage capacity. It might have a 200MHz CPU speed because the majority of the applications are interactive and display-intensive, rather than computation intensive. This class of PAD are small because the overall PDA device is slim and fits in the palm of your hand. The limited functionality means reduced power consumption and long-lasting battery life. The smaller size reduces the overall cost of processor fabrication. On the other hand, another class of distributed processors focuses on performance. These distributed processors are powerful and packed with advanced chip-design technologies, such as advanced pipeline and parallel processing architecture. These processors are designed to satisfy those applications with intensive computing requirements not achievable with general-purpose processors. An emerging class of highly specialized and high-performance distributed processors includes network processors developed for the network equipment and telecommunications industry. Overall, system and application speeds are the main concerns.

Yet another class of distributed processors focuses on all four requirements—performance, size, power consumption, and price. Take, for example, the distributed digital signal processor (DSP) used in cell phones. Real-time voice communication involves digital signal processing and cannot tolerate delays. A DSP has specialized arithmetic units, optimized design in the memory, and addressing and bus architectures with multiprocessing capability that allow the DSP to perform complex calculations extremely fast in real time. A DSP outperforms a general-purpose processor running at the same clock speed many times over comes to digital signal processing. These reasons are why DSPs, instead of general-purpose processors, are chosen for cell phone designs. Even though DSPs are incredibly fast and powerful distributed processors, they are reasonably priced, which keeps the overall prices of cell phones competitive. The battery from which the DSP draws power lasts for hours and hours [7].

4. DISTRIBUTED APPLICATION'S FEATURES

Distributed applications have some common features such as the following [7]: **Limited resources**: There are often strong limitations regarding available resources. Mainly due to cost and size constraints related to mass production and strong industrial competition, the system resources as CPU, memory, devices have been designed to meet these requirements. As a result of these limitations, the system has to deal with an efficient use of the computational resources. **Real-time application requirements**: Some of the applications to be run in these devices have temporal requirements. These applications are related with process control, multimedia processing, instrumentation, and so on, where the system has to act within a specified interval. **Distributed control systems:** Most of the distributed systems perform control activities involving input data acquisition (sensing) and output delivery (actuation). Deterministic communications are also another important issue. **Quality of service:** An efficient use of the system resources is a must in distributed systems. Feedback based approaches are being used to adjust the performance or quality of service of the applications as a function of the available resources.

The challenge is how to implement applications that can execute efficiently on limited resource and that meet nonfunctional requirements such as timeliness, robustness, dependability, performance, and so on. Within the exception of these few common features, rest of the distributed hardware is usually unique and varies from application to application. Each system must meet a completely different set of requirements [5, 6]. The common critical features and design requirements of an distributed hardware include [2]: **Processing power:** Selection of the processor is based on the amount of processing power to get the job done and also on the basis of register width required. **Throughput:** The system may need to handle a lot of data in a short period of time. **Response:** the system has to react to events quickly. **Memory:** Hardware designer must make his best estimate of the memory requirement and must make provision for expansion. **Power consumption:** Systems generally work on battery and design of both software and hardware must take care of power saving techniques. **Number of units:** the no. of units expected to be produced and sold will dictate the Trade-off between production cost and development cost. **Expected lifetime:** Design decisions like selection of components to system development cost will depend on how long the system is expected to run. **Program Installation:** Installation of the software on to the distributed system needs special tools.

Testability & Debug ability: setting up test conditions and equipment will be difficult and finding out what is wrong with the software will become a difficult task without a keyboard and the usual display screen.

Reliability: is critical if it is a space shuttle or a car but in case of a toy it doesn't always have to work right.

5. REQUIREMENTS FOR DISTRIBUTED SYSTEMS

Distributed systems are unique in several ways. When designing distributed systems, there are several categories of requirements that should be considered [1, 5]:

1. Functional Requirements
2. Temporal Requirements (Timeliness)
3. Dependability Requirements

1. Functional Requirements Functional requirements describe the type of processing the system will perform. This processing varies, based on the application. Functional requirements include the following; • Data Collection requirements • Sensoring requirements • Signal conditioning requirements • Alarm monitoring requirements • Direct Digital Control requirements • Actuator control requirements • Man-Machine Interaction requirements (Informing the operator of the current state of a controlled object for example. These interfaces can be as simple as a flashing LED or a very complex GUI-based system. They include the ways that distributed systems assist the operator in controlling the object/system.

2. Temporal Requirement Distributed systems have many tasks to perform, each having its own deadline. Temporal requirements define the stringency in which these time-based tasks must complete. Examples include; • Minimal latency jitter • Minimal Error-detection latency Temporal requirements can be very tight (for example control-loops) or less stringent (for example response time in a user interface).

3. Dependability Requirements Most distributed systems also have a set of dependability requirements. Examples of dependability requirements include;

• **Reliability**: this is a complex concept that should always be considered at the system rather than the individual component level. There are three dimensions to consider when specifying system reliability:

1. Hardware reliability; probability of a hardware component failing
2. Software reliability; probability that a software component will produce an incorrect result
3. Operator reliability; how likely that the operator of a system will make an error

There are several metrics used to determine system reliability;

- Probability of failure on demand; likelihood that the system will fail when a service request is made.
- Rate of failure occurrence; frequency of occurrence with which unexpected behavior is likely to occur.
- Mean Time to Failure (MTTF); the average time between observed system failures.

• **Safety**: describe the critical failure modes and what types of certification are required for the system.

• **Maintainability**: describes constraints on the system such as type of Mean Time to Repair (MTTR).

• **Availability**: the probability that the system is available for use at a given time. Availability is measured as; Availability = MTTF / (MTTF+MTTR)

• **Security**: these requirements are often specified as "shall not" requirements that define unacceptable system behavior rather than required system functionality.

6. PROGRAMMING DISTRIBUTED SYSTEMS

A large variety of applications can be found where distributed systems play an important role, from small stand-alone systems, like a network router, to complex distributed systems supporting several operating execution environments as founded in avionic applications. This variety of applications also implies that the properties, platforms, and techniques on which distributed systems are based can be very different. The hardware needs can sometimes be achieved with the use of general purpose processors, but in many systems specific processors are required, for instance, specific DSP devices to perform fast signal processing. Memory management capabilities are necessary in some systems to provide memory protection and virtual memory. Special purpose interfaces are also needed to support a variety of external peripheral devices, energy consumption control, and so on

Every distributed system had at least one LED that could be controlled by software. The following example is designed to blink LED at a rate of 1 Hz(one complete on-off cycle per second). Typically, the code required to turn an LED on and off is limited to a few lines of C or assembly, so there is very little room for programming errors to occur. And because almost all distributed systems have LEDs, the underlying concept is extremely portable [4].

The superstructure of the Blinking LED program is shown below. This part of the program is hardware-independent. However, it relies on the hardware-dependent functions toggleLed and delay to change the state of the LED and handle the timing, respectively.

```
  /****************************************************
************* * Function: main()
  *
   * Description: Blink the green LED once a second.
  *
  * Notes: This outer loop is hardware-independent. How-
ever,
  * it depends on two hardware-dependent functions.
  *
   * Returns: This routine contains an infinite loop.


   *
   void
  main(void)
  {
  while (1)
  {
  toggleLed(LED_GREEN);          /* Change the state of
the LED. */
   delay(500);        /* Pause for 500 milliseconds. */
  }
  /* main() */
  }
```

7. CONCLUSION

Real-time distributed systems vary in their functions and their component. The components of simple distributed systems are usually less than of the sophisticated ones. In addition the components themselves including the hardware and software are being more complexes for the distributed systems which perform high level tasks. The software for efficient distributed systems is optimized to reduce the execution time during work. Therefore a new software is required when the hardware is upgraded to keep the high level of performance. The failure of the hardware in many cases causes shutdown in the system and usually recoding is required to bring back the system to work. On the other hand any defects in software will not damage the hardware. The following conclusions can be drowning from the current work:

1. In this report, the real-time and distributed system characteristics and there requirements have been listed and discussed thoroughly.

2. The distributed systems comprised hardware and software. The hardware should contain the processor and memory and some peripherals devices for input and output. While the suitable software is associated with every part of the hardware in order to perform the required function that the distributed system designed to do.
3. The final step of writing distributed system software is optimizing the code to make the working program run on the lower-cost production version of the hardware.
4. Code optimization can be provided by increasing code efficiency, decreasing code size, and reducing memory usage.

FUNDING

None

ACKNOWLEDGEMENT

None

CONFLICTS OF INTEREST

The author declares no conflict of interest.

REFERENCES

[1] O. Robert, “Introduction to embedded and Real-time systems.” Texas Instruments Engineering Manager.
[2] M. Tech and Vivek, “Embedded Operating Systems for Real-Time Applications,” *Sagar P M*, 2002.
[3] http://searchenterpriselinux.techtarget.com/sDefinition/0,sid39_gci837507,00.htmlSearchEnterpriseLinux.comDefinitions.
[4] B. Michael, *Programming Embedded Systems in C and C++*. 1999.
[5] S. Abraham, P. Galvin, G. Baer, and Greg, “Operating System Concepts,” 2005. John Wiley & SONS. INC.
[6] S. John and A, *Real-Time and Embedded Systems*.
[7] “How to Design a RealTime Embedded System Using UML | eHow.” http://www.ehow.com/how_6596645_design-embedded-system-using-uml.html#ixzz1S47rv0qc.
[8] “What makes a good RTOS,” *Dedicated Systems Experts*, 2001.
[9] L. Qing and Y. Caroline, “Real-Time Concepts for Embedded Systems,” *CMP Books an imprint of CMP Media LLC*.
[10] F. Peter, H, L. Bruce, and V. Steve, “Improving Predictability in Embedded Real-Time Systems,” 2000.