EOE-DRTSA: End-to-End Distributed Real-time System Scheduling Algorithm

Dhuha Basheer Abdullah[1], Amira Bibo Sallow[2]

[1]Computer Science Dept., Mosul University, Computer Sciences and Mathematics College
Mosul, Iraq

[2]Computer Science Dept., Mosul University, Computer Sciences and Mathematics College
Mosul, Iraq

Abstract

In this paper, scheduling dependent threads in distributed real-time system where considered. We present a distributed real-time scheduling algorithm called (EOE-DRTSA (end-to-end distributed real time system Scheduling algorithm)). Now a day completed real-time systems are distributed. One of least developed areas of real-time scheduling is distributed scheduling where in Distributed systems action and information timeliness is often end-to-end. Designers and users of distributed systems often need to dependably reason about end-to-end timeliness. Our scheduling model includes threads and their time constraints depend on developed DTUF value and maintaining end-to-end prosperities of distributed real-time system.

Keywords: Collaborative scheduling, end-to-end constraints, RMI java, Real-time Distributed system, Real-time scheduling algorithm, Time/utility function, DTUF.

1. INTRODUCTION

Distributed real-time systems such as those found in industrial automation, and military surveillance must support for timely, end-to-end activities. Timeliness includes application-specific acceptability of end-to-end time constraint satisfaction, and of the predictability of that satisfaction. These activities may include computational, sensor, and actuator steps which levy a causal ordering of operations, contingent on interactions with physical systems. Such end-to-end tasks may be represented in a concrete distributed system as: chains of
(a) nested remote method invocations;
(b) publish, receive steps in a publish-subscribe framework;
(c) event occurrence and event handlers **[5].**

Reasoning about end-to-end timeliness is a difficult and unsolved problem in such systems. A distinguishing feature of such systems is their relatively long activity execution time scales (e.g., milliseconds to minutes), which permits more time-costlier real-time resource management. Maintaining end-to-end properties (e.g., timeliness, connectivity) of a control or information flow requires a model of the flow's locus in space and time that can be reasoned about. Such a model facilitates reasoning about the contention for resources that occur along the flow's locus and resolving those contentions to optimize system-wide end-to-end timeliness. The distributable thread programming abstraction which first appeared in the Alpha OS, and later the Real-Time CORBA 1.2 standard directly provides such a model as their first-class programming and scheduling abstraction. A distributable thread is a single thread of execution with a globally unique identity that transparently extends and retracts through local and remote objects **[12].**

When resource overloads occur, meeting deadlines of all application activities is impossible as the demand exceeds the supply. The urgency of an activity is typically orthogonal to the relative importance of the activity-e.g., the most urgent activity can be the least important, and vice versa; the most urgent can be the most important, and vice versa. Hence when overloads occur, completing the most important activities irrespective of activity urgency is often desirable. Thus, a clear distinction has to be made between the urgency and the importance of activities, during overloads **[3]**.

Deadlines by themselves cannot express both urgency and importance. Each thread's time constraint is specified using a time/utility function (or TUF) .. They were introduced by E. Douglas Jensen in 1977 as a way to overcome the limited expressiveness in classic deadline constraints in real-time systems. In a graphical interpretation, the utility (positive for reward, negative for penalty) is plotted over the time. A deadline then represents the point where the utility changes from positive to negative. In computer science and programming, this is when a task must be terminated. If not, an exception occurs, which usually leads to an abortion. As such, a TUF is a generalization of deadline constraints in everyday life. With TUF time constraints, timeliness optimality criteria can be specified **[6]**.

Past efforts on thread scheduling can be broadly categorized into two classes: independent node scheduling and collaborative scheduling. In the independent scheduling approach, threads are scheduled at nodes using propagated thread scheduling parameters and without any interaction with other nodes. Thread faults are managed by integrity protocols that run concurrent to thread execution. Integrity protocols employ failure detectors (or FDs), and use them to detect thread failures. In the collaborative scheduling approach, nodes explicitly cooperate to construct system-wide thread schedules, detecting node failures using FDs while doing so**[5]**.

2. Contributions

In this paper, we consider the problem of scheduling dependent threads. So we design a collaborative thread scheduling algorithm called EOE-DRTSA that has the following prosperities:

1. Using RMI Java technique to build distributed system model.
2. It makes decisions in each local scheduling dependently from the global scheduler.
3. Each thread has DTUF value that has been calculated depending on Importance and urgency of the thread.
4. The DTUF value makes the decision of which thread will be executed next.

2. Related work

Past works on developing scheduling algorithm for distributed real time system by Sherif F. Fahmy, Binoy Ravindran, and E. D. Jensen they considered the distributable threads abstraction for programming and scheduling such systems, and presented a collaborative thread scheduling algorithm called the Quorum-Based Utility Accrual scheduling (or QBUA). they showed that QBUA satisfies (end-to-end) thread time constraints in the presence of crash failures and message losses, has efficient message and time complexities, and lower overhead and superior timeliness properties than past thread scheduling algorithms **[12].**

Sherif Fahmy, Binoy Ravindran, and E. D. Jensen in year 2010. They considered scheduling distributable real-time threads with dependencies in partially synchronous systems in the presence of node failure. they present a collaborative distributed real-time scheduling algorithm called DQBUA. The algorithm uses quorum systems to coordinate nodes' activities when constructing a global schedule. DBQUA detects and resolves distributed deadlock in a timely manner and allows threads to access resources in order of their potential utility to the system. Their main contribution is handling resource dependencies using a distributed scheduling algorithm **[9].**

Binoy Ravindran, Edward Curley, Jonathan Anderson, and E. Douglas Jensenz, They considered the problem of recovering from failures of distributable threads in distributed real-time systems that operate under run-time uncertainties including those on thread execution times, thread arrivals, and node failure occurrences. They presented a scheduling algorithm called HUA and a thread integrity protocol called TPR **[4].**

Jonathan S. Anderson, Binoy Ravindran, and E. Douglas Jensen, they demonstrated a consensus utility accrual scheduling algorithm for distributable threads with run-time uncertainties in execution time, arrival models, and node crash failures algorithm called DUA-CLA algorithm. The DUA-CLA algorithm represents a unique approach to distributable thread scheduling in two respects. First, it unifies scheduling with a fault-tolerance strategy. Second, DUA-CLA takes a collaborative approach to the scheduling problem, rather than requiring nodes independently to schedule tasks without knowledge of other nodes' states. Global scheduling approaches where in a single, centralized scheduler makes all scheduling decisions have been proposed and implemented. DUA-CLA takes a via media, improving independent node scheduler decision making with partial knowledge of global system state **[5].**

Piyush Garyali, Matthew Dellinger, and Binoy Ravindran considered the problem of scheduling dependent real-time tasks for overloads on a multiprocessor system, yielding best-effort timing assurance. they developed a class of polynomial-time heuristic algorithms, called the Global Utility Accrual (GUA) class of algorithms and they developed a Linux-based real-time kernel called ChronOS **[9].**

Haisang Wu, Binoy Ravindran, and E. Douglas Jensen they extended Jensen's time/utility functions and utility accrual model with the concept of joint utility functions (or JUFs) that allow an activity's utility to be described as a function of the completion times of other activities and their progress. they also specified the concept of progressive utility that generalizes the previously studied imprecise computational model, by describing an activity's utility as a function of its progress. Given such an extended utility accrual model, they considered the scheduling criterion of maximizing the weighted sum of completion time, progressive, and joint utilities. they presented an algorithm called the Combined Utility Accrual algorithm (or CUA) for this criterion. Experimental measurements with an implementation of CUA on a POSIX RTOS illustrate the effectiveness of JUFs in a class of applications of interest to them **[2].**

3. Distributed real-time systems

The vast majority of deployed distributed real-time computing systems employ at least one of the following programming models:

control flow: movement of an execution point, with or without parameters, among application entities – e.g., remote procedure call (RPC) and remote method invocation (RMI)

data flow: movement of data, without an execution point, among application entities – e.g., publish/ subscribe and bulk data transfers

networked: asynchronous or synchronous movement of messages, without an execution point, among application entities – e.g., message passing IPC

Other distributed system programming models – e.g., mobile objects, autonomous agents, web services, etc.

Control flow models are usually designed for multi-node – usually trans-node (linearly sequential) – behaviors that are synchronous, that is they request a remote execution and then

wait for a response. When a synchronous response is not needed, a one-way invocation can be spawned asynchronously (as in CORBA). An example of the distributed control approach is the distributed thread model. A distributed thread is a single thread, with a system-wide ID, that extends and retracts itself sequentially through an arbitrary number of local and remote objects. A distributed real-time thread transparently propagates its timeliness properties (and perhaps also resource ownership, transactional context, security attributes, etc.) when its execution point transits object (and perhaps node) boundaries.
Almost all data flow models, including those for “real-time” publish/subscribe, are more oriented toward maximizing throughput than maintaining end-to-end timeliness properties (cf. OMG’s RFP for a Data Distribution Service for Real-Time Systems).

Java already includes a model for distributed systems – Remote Method Invocation (RMI) – and JSR-50 proposed to enhance it for real-time systems. RMI provides the familiar distributed object system control flow model of method invocations using abstract interfaces to define local stubs for remote objects. As in any distributed object system, a programming abstraction similar to asynchronous message passing can be provided by allowing asynchronous (one-way with no return parameters) method invocations. RMI also can transfer object instances by value. This allows messages to be passed as objects. It also supports a simple form of data flow – for point-to-point flow of modestly sized data, but not for effective publish/subscribe models.

A distributed object system (as opposed to, for instance, web services) requires a well-defined architecture with stable interfaces between components and some level of system-wide agreement on infrastructure technology. In a real-time distributed object system that agreement includes the semantics of timeliness and sufficiently synchronized local clocks. Typically, such systems are found inside an enterprise – it is difficult to create the technical agreement and coordination needed to build a distributed object system between enterprises. This is consistent with the current normal deployment of distributed real-time computing systems. The scalability requirement for the DRTSJ is based on this presumption of intra-enterprise environments **[1]**.

4. RMI and JAVA

The main components of RMI can be considered as follows:
•**the programming model,** where objects that can be accessed remotely are identified via a remote interface,
•**the implementation model,** which provides the transport mechanisms whereby one Java platform can talk to another in order to request access to its objects, and
•**the development tools** (e.g., rmic or its dynamic counterpart), which take server objects and generate the proxies required to facilitate the communication.
Key to developing RMI-based systems is defining the interfaces to remote objects. RMI requires that all objects that are to provide a remote interface must indicate so by extending the pre-defined interface Remote. Each method defined in an interface extending Remote must declare that it "throws RemoteException". Thus one of the key design decisions of RMI is that distribution is not completely transparent to the programmer. The location of the remote objects may be transparent, but the fact that remote access may occur is not transparent **[7].**

5. RMI applications

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application.

Distributed object applications need to do the following:
Locate remote objects. Applications can use various mechanisms to obtain references to remote objects. For example, an application can register its remote objects with RMI's simple naming facility, the RMI registry. Alternatively, an application can pass and return remote object references as part of other remote invocations.
Communicate with remote objects. Details of communication between remote objects are handled by RMI. To the programmer, remote communication looks similar to regular Java method invocations.
Load class definitions for objects that are passed around. Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's data.

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed **[12].**

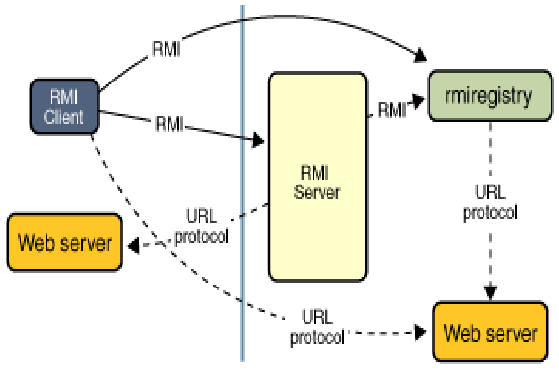

Fig.1: RMI Applications

6. Models and Objectives

6.1. The System Model

The model of our system architecture is shown in Figure 2. We consider a distributed system architecture model consisting of heterogeneous processors; a set of client nodes; and a set of server nodes. These nodes were interconnected via a communication network. There is a single Global Scheduler for the system, responsible for computing the initial priorities for the tasks and its utility is checked. based on information provided by the application programmer. As new tasks were introduced to the system, the Global Scheduler distributes the Tasks to the client processors. A Local Scheduler on each processor was responsible for specifying The order of executing threads on a node. Thus, scheduling decisions made by a node scheduler were independent from that made by other node schedulers.

Client Node schedulers make scheduling decisions using thread scheduling attributes, which typically include threads' time constraints (e.g., importance, urgency). When a new job arrives at Client Node, If the new job has maximum utility than the currently executing, the current job will be preempted and the new job will be scheduled immediately.

6.2 Task Model

We define a thread as a basic unit of CPU utilization. It comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals. Our Real-Time distributed application is comprised of a set of threads, denoted as T ={ T_1; T_2; T_3; . ..}. All threads are assumed to be:

1. Aperiodic and their arrival time is not known a priori.
2. Preemptable and, therefore, can be preempted at any time during their execution.
3. Do not share any non-CPU resources or have precedence relations with other threads and, are independent of each other.

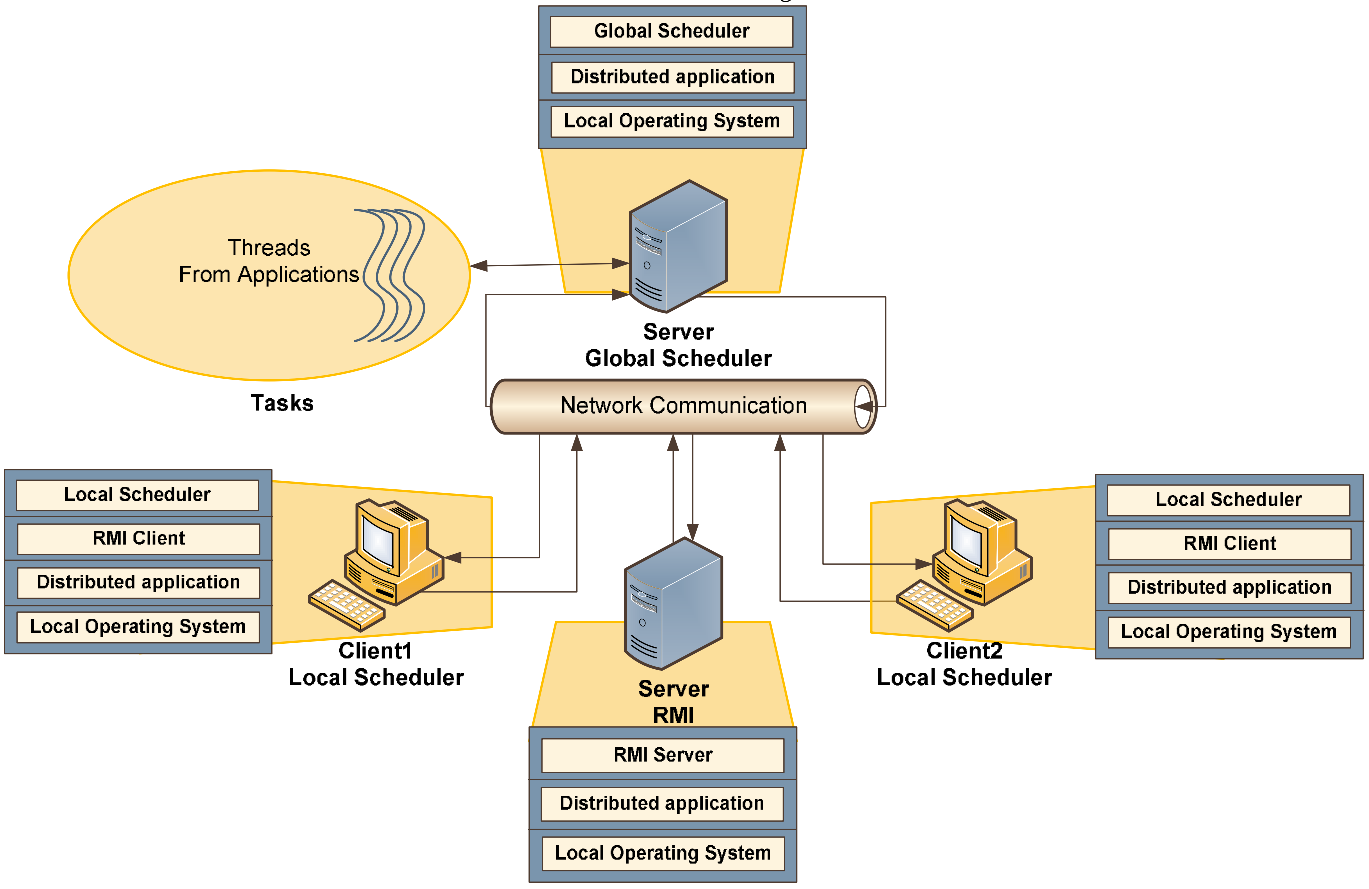

Fig. 2: Our System Model

We pointed to the thread by task in paper sections, and Each thread has the following parameters:

1. **Deadline (DL):** the time interval which the thread should be completed, the Developer will specify it's value.
2. **Importance (I):** a metric that represents the relative importance of the thread. As the Developer will specify

it's value we consider that the value of the importance be between (1-9).

3. **Computation Time (C):** the amount of time required for the task to complete the execution.
4. **Urgency (U):** the difference between Deadline(DL) and Computation Time(C).

$$Urgency = Deadline - Computation\ time$$

6.3 Timeliness Model

We develop the TUF function depending on the concept of a Time Management Matrix for prioritizing. It is a simple tool which helps to priorities' tasks, based on whether they are Urgent or Important, or both, or neither, introduced by Stephen Covey **[11]**.

The Time Management Matrix advocates the use of four quadrants to determine the tasks need to do immediately and the other which need to eliminated, the four quadrants are shown in figure(3):

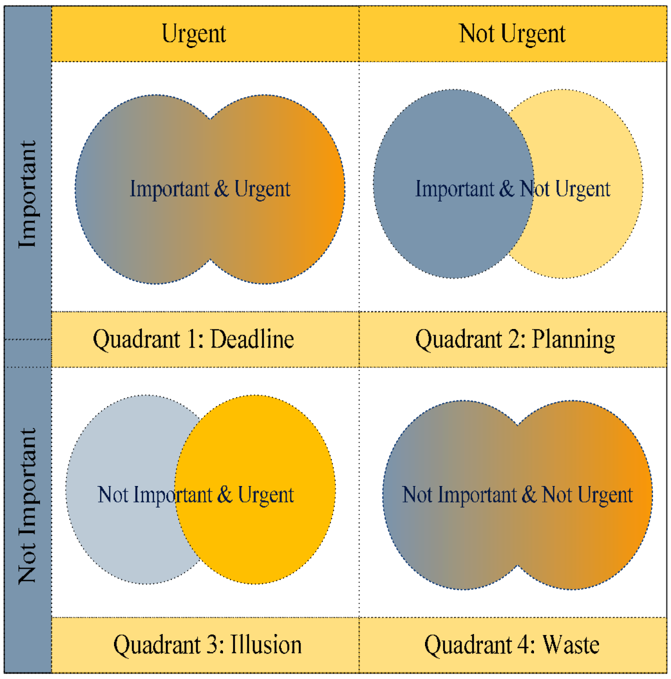

Fig. 3: Time Management Matrix

1. ***Quadrant 1 (Deadline):*** There are important, urgent tasks need to be delt with immediately.
2. ***Quadrant 2 (Planning):*** There are important, but not urgent tasks. They do not require immediate attention, and need to be planned for.
3. ***Quadrant 3 (Illusion):*** There are urgent, but unimportant tasks which should be minimized or eliminated.
4. ***Quadrant 4 (Waste):*** There are unimportant and also not urgent tasks that don't have to be done anytime soon, and also should be minimized or eliminated. These are often wasting time.

We consider the DTUF (Developed TUF) function to be three dimensional function that decouples importance and urgency of a thread, urgency is measured on the X-axis, and importance is measured on the Y-axis. The Benefit is denoted by DTUF and is measured on the Z-axis figure(4) . The equation of our DTUF function is:

$$DTUF = \begin{cases} e^{(2*Importance)} + e^{(Urgency)} & urgency > 0 \\ e^{(2*Importance)} & urgency = 0 \\ 0 & urgency < 0 \end{cases}$$

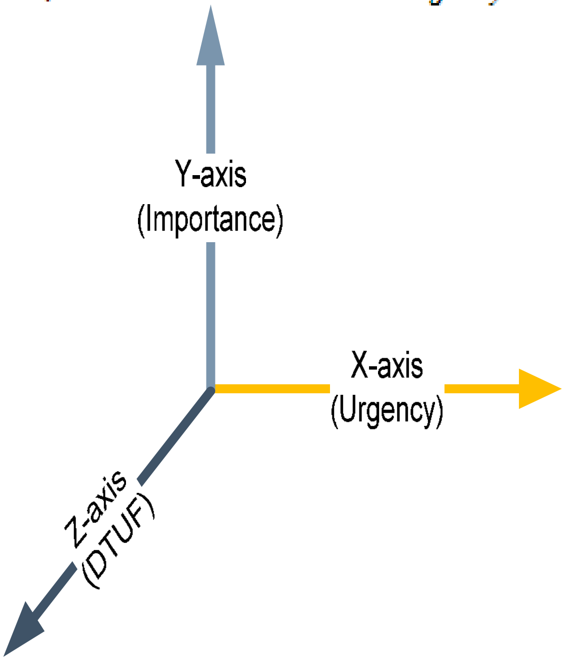

Fig. 4: 3D Time/Utility Functions

7. Scheduling Objectives

Our primary objective is to design a thread scheduling algorithm to maximize the total utility accrued by all threads as much as possible in the presence of dependencies. Further, the algorithm must provide assurances on the satisfaction of thread termination times in the presence of crash failures. Moreover, the algorithm must bounds the time threads remain in a deadlock.

8.Algorithm Description

EOE-DRTSA algorithm was designed to overcome the shortcomings of independent scheduling algorithms by taking into account global information while constructing schedule. In EOE-DRTSA scheduling when a thread arrives, it first arrives to the Global Scheduler server. The Global Scheduler will compute the initial Urgency and DTUF value. After the computation was done the Global Scheduler will pass the threads to the target node to be executed there.

When the thread reaches the Local Scheduler node. The node will schedule the thread depending on it's local scheduling algorithm. EOE_integrity protocol will send report to the Global Scheduler server to analyze the report and monitor the scheduling work on the node. If the server detects any failure on any node, it will run the EOE_Failure protocol.

Algorithm 1: EOE-DRTSA Distributed scheduling algorithm On Global scheduling Server side:

```
1: Input: Thread set to be distributed to client.
2: Output: Feasible Scheduling produced from the Distributed Real-time system.
3: Thread (in the task queue) was ordered depending on their arrival time.
4: If (not empty thread queue)
    Repeat
    Calculate Urgency of each thread by using:
    $Urgency = Deadline - Computation\ time$
    Calculate the DTUF value for each thread by suing:
    $DTUF = \begin{cases} e^{(2\cdot Importance)} + e^{(Urgency)} & urgency > 0 \\ e^{(2\cdot Importance)} & urgency = 0 \\ 0 & urgency < 0 \end{cases}$
    Send task to its related client to be executed.
     Until (no thread in the task queue)
EndIf
5: Repeat
    Read monitoring report for each client
    If any Fault on the client
    Start EOE_Failure protocol
    Until (all clients in the system finished execution)
6:End
```

Algorithm 2: EOE-DRTSA Distributed scheduling algorithm On local scheduling Client side:

```
1: Input: Thread Set to be executed.
2: Output: Execute Thread Set and meet their deadline.
3: Thread (in the task queue) was ordered depending on their DTUF value.
4: If (not empty task queue)
    Repeat
    Execute task with high DTUF value.
    If (any Task arrived with highest DTUF value)
    Preempt the current Task.
    Reorder the Task queue.
    EndIf
    Until (no task in the task queue)
5:End
```

9. Simulation

We performed a series of simulation experiments on a set of threads (Table (1)) to measure the performance of our algorithms. The results are summarized in Section IX.

Table 1: Threads set example.

Task	Deadline	Importance	Urgency	Cost	DTUF
p1	7	8	3	4	8886111
p2	9	6	4	5	162809.4
p3	5	6	3	2	162754.8
p4	4	4	1	3	2980.958
p5	3	3	0	3	403.4288
p6	6	2	2	4	54.59815
p7	8	1	2	6	7.389056
p8	2	2	-1	3	0

We compare the utilization of all threads in the system with the utilization of all threads depending on DTUF. Figure (4) shows that the utilization of threads without using DTUF was equal until (no. task=5) where the DTUF of p5=0 which effects the total utilization of the system using DTUF which be smaller than the utilization of system without using DTUF.

10. Conclusion

We have implemented a dynamic scheduling algorithm that examines the computation times, real time requirements of the tasks to produce a feasible schedule for Distributed Real-Time system. The schedule was driven by using DTUF function of the urgency (laxity), and importance of the tasks. The decisions are made on a system-wide basis because tasks were represented as distributed threads that have end-to-end timing requirements. We conclude that the TUF Function has significantly advanced and the scheduling algorithm is generic and can be used in other distributed soft real-time systems.

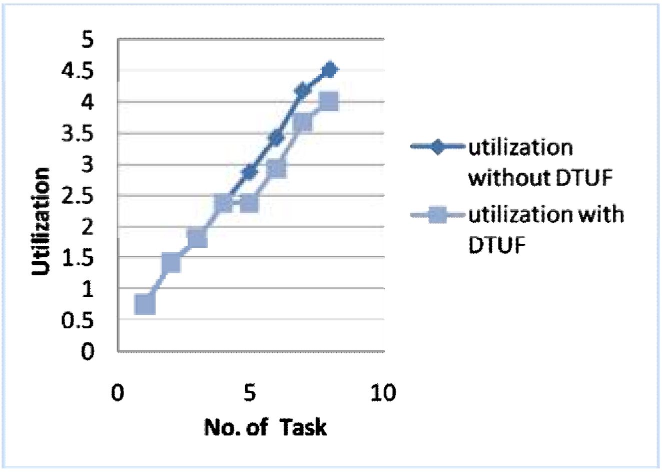

Fig. 4: Simulation Results

References

[1] Andy Wellings , Ray Clark and Doug Jensen, “A Framework for Integrating the Real-Time Specification for Java and Java's Remote Method Invocation”, University of York, 2001.

[2] Haisang Wu, Binoy Ravindran, and E. Douglas Jensen, “On the Joint Utility Accrual Model”, ECE Dept., Virginia Tech,USA, 2004.

[3] Hyeonjoong Cho, Binoy Ravindran, E. Douglas Jensen, ,” On Lock-Free Synchronization for Dynamic Embedded Real-Time Software”, ECE Dept., Virginia Tech, 2004.

[4] Jensen Douglas, ” Application QoS-Based Time-Critical Automated Resource Management in Battle Management Systems”, The MITRE Corporation, 2003.

[5] Jonathan S. Anderson, Binoy Ravindran, and E. Douglas Jensen,“Consensus-Driven Distributable Thread Scheduling in Networked Embedded Systems”, Department of Electrical and Computer Engineering,Virginia Tech, USA, 2007.

[6] Matthew A. Dellinger, ,”An Experimental Evaluation of the Scalability of Real-Time Scheduling Algorithms on Large-Scale Multicore Platforms”, Blacksburg, Virginia, 2011.

[7] Mohamed M. Saad , Binoy Ravindran ,“Supporting STM in Distributed Systems: Mechanisms and a Java Framework”, ECE Dept., Virginia Tech, Blacksburg, VA 24060, USA, 2010.

[8] Piyush Garyali, Matthew Dellinger, and Binoy Ravindran, ,“On Best-Effort Utility Accrual Real-Time Scheduling on Multiprocessors”., Virgina Tech, USA, 2010.

[9] Sherif Fahmy, Binoy Ravindran, and E. D. Jensen,, “Scheduling Dependent Distributable Real-TimeThreads in Dynamic Networked Embedded Systems” ”,ECE Dept., Virginia Tech,USA, 2007.

[10] Sherif Fahmy, Binoy Ravindran, and E. D. Jensen, “Fast Scheduling of Distributable Real-Time Threads with Assured End-to-End Timeliness”, ECE Dept., Virginia Tech,USA, 2008.

[11] Stephen R. Covey,” THE SEVEN HABITS OF HIGHLY EFFECTIVE PEOPLE ”, FranklinCovey, 1989.

[12] William Grosso, “Java RMI “, O'Reilly O'Reilly and Associates, Inc. 1005 Gravenstein Highway North Sebastopol, 2001.

Dhuha Albazaz is the head of Computer Sciences Department, College of Computers and Mathematics, University of Mosul. She received her PhD degree in computer sciences in 2004 in the speciality of computer architecture and operating system. She supervised many Master degree students in operating system, computer architecture, dataflow machines, mobile computing, real time, and distributed databases. She has three PhD students in FPGA field, distributed real time systems, and Linux clustering. She also leads and teaches modules at both BSc, MSc, and PhD levels in computer science. Also, she teaches many subjects for PhD and master students.

Amira Bibo Sallow is a Phd. student in Computer Sciences Department, College of Computers and Mathematics, University of Mosul. She interest with networks, Databases, and operating system subjects.