International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-2 Number-4 Issue-7 December-2012

Real Time Scheduling Services for Distributed RT-CORBA Applications

Bineta Tresa Mathew
Assistant Professor, Department of Computer Science and Engineering Birla Institute of Technology,
Off shore Campus Ras Al Khaimah, UAE

Abstract

Distributed computing environment is flexible to control in complex embedded systems and their software components gain complexity when these systems are equipped with many microcontrollers and software object which covers diverse platforms, this system is called as DRE system. These DRE systems need new inter-object communication solution thus QoS-enabled middleware services and mechanisms have begun to emerge. Real-time application domain benefit from flexible and open distributed architectures, such as those defined by the CORBA specification. CORBA is well-suited to conventional request/response applications, but not suited to real-time applications due to the lack of QoS features and performance optimizations. The paper shows the design and implementation of the high performance scheduling technique for the real time applications domain with CORBA systems. Four different algorithms are compared by using attributes of real time tasks constraints based on CORBA specification such as RMS, MLF, MUF and EDF. The experimental outcome demonstrates the better performance of MLF by analyzing the time taken for the execution of several numbers of tasks and further it can be compared with the combination of RMS and MLF to reach the best performance strategy.

Keywords

DRE, QoS, CORBA, RMS, MLF, MUF, EDF

1. Introduction

Distributed real-time embedded (DRE) systems are becoming increasingly widespread and important. Common DRE systems include Telecommunication networks, tele-medicine, manufacturing process automation and defence applications. DRE systems should be capable of communicating in a distributed environment, be efficient and predictable and have less memory foot-print. DRE applications are tedious and error-prone because they are developed using low-level languages. These systems are hard to debug due to the limited availability of the debugging tools. Because of these challenges, application developers shifted towards software models that are reusable [5]. This paved way for Real-Time CORBA (RT CORBA).

CORBA is distribution middleware that provides run-time support to automate many distributed computing tasks, such as connection management, object marshalling / demarshaling, object demultiplexing, language and platform independence, load balancing, fault-tolerance, and security [1]. Real-time CORBA adds QoS control capabilities to regular CORBA which include improving application predictability by bounding priority inversions and managing system resources end-to-end [2]. Real-time CORBA also facilitates the configuration and control of the system resources such as processor resources, communication resources and memory resources.

Most current implementations of Real-time CORBA are available only in C++ or Ada [5]. The Java programming language is an attractive alternative because it is widely used, powerful and portable. Java also offloads many tedious and error-prone programming details from developers into the language run-time system. It has desirable language features, such as strong typing, dynamic class loading, and reflection/introspection. Java defines portable support for concurrency and synchronization. The Real-time Java Experts Group has defined the Real-time Specification for Java (RTSJ) [5], which provides capabilities without modifying the Java programming language itself. Some of the capabilities of RT Java suitable for real-time embedded systems include efficient memory management models, access to raw physical memory and stronger guarantees on thread semantics.

ZEN [3] is a Real-time CORBA ORB implemented using Real-time Java, thereby combining the benefits of these two standard technologies. Zen's ORB architecture is based on the concept of layered plugMLbility. Zen employs the Micro-kernel architecture. It has eight core ORB services namely object adapters, message buffer allocators, GIOP message handling, CDR Stream readers/writers,

protocol transports, object resolvers, IOR parsers, and any handlers. These are removed out of the ORB to reduce its memory footprint and increase its flexibility. The remaining portion of code is called the ZEN kernel.

TAO, a real-time Object Request Bus (ORB) developed by Object Computing Research group at the Washington University and Vanderbilt University aims at optimizing collocation, ORB protocol overhead, Portable Object Adapter (POA) demultiplexing and strategy patterns for scheduling to make CORBA suitable for real time applications [3]. It uses Rate Monotonic Scheduling (RMS), Maximum Laxity First (MLF) in and Maximum Urgency First (MUF) algorithms for scheduling. It does not consider multiprocessor scheduling strategies [4]. Our scheduling framework considers multiprocessor scheduling in a distributed environment. The framework is capable of scheduling the tasks intelligently based on the load of the processors and the schedulability of the tasks [1]. Since the proposed scheduling framework aims at using RTCORBA and RTJS, it can alleviate the shortcomings in the non-CORBA based implementations of scheduling strategies by bringing about heterogeneity, and platform, hardware, location transparency. Further, as it uses distributed object technology, it provides facility for extensibility.

2. Importance

Real-time applications like aircraft control systems, military command systems, industrial automation systems, transportation and telephone switches should execute their tasks within certain deadlines. Common Object Request Broker Architecture (CORBA) provides flexible middleware capable of integrating complex applications in heterogeneous environments. CORBA with its Minimum CORBA, Real-time CORBA and messaging standards makes it a highly suitable Distributed Object Computing middleware satisfying memory and timing constraints of real-time embedded applications [1]. The proposed framework aims at extending the CORBA scheduling service to bring about efficient and intelligent task scheduling in a distributed environment. It can be dynamically configured and uses hybrid and multiprocessor scheduling algorithms. Since it is extensible, it lowers the software evolution lifecycle cost and time. The framework developed from the proposed concept can be applied in various real time systems including the following:

- Air Force Multi-Platform Radar systems
- Naval / Air-line systems
- Aircraft control systems
- Firm Real-time systems
- Transportation control and Guidance systems
- Mixed Systems
- Telecom Networks

3. Tools and Methods

Models for unique paradigm for scheduler to execute without missing the deadline in the real time environment. The aim is to make middleware to real time suitable using paradigm approach. The characteristic of scheduling service model is real time response and adaptability with any kind of real-time applications. The paradigm approach in [4] has divided into two categories they are single paradigm approach and multi-paradigm approach. The two techniques are described below.

- Single paradigm strategy
    - Fixed priority algorithm (Static – RMS)
    - Dynamic priority algorithm (EDF and MLF)
- Multi-paradigm strategy
    - Hybrid algorithm (MUF)

Single Paradigm Strategy

Scheduling algorithms themselves can be characterized as being either static or dynamic. A ***static approach*** calculates (or pre-determines) schedules for the system. It requires prior knowledge of a process's characteristics but requires little run-time overhead. By comparison, a ***dynamic approach*** determines schedules at run-time thereby furnishing a more flexible system that can deal with non-predicted events. It has higher run-time cost but can give greater processor utilization. Certainly in safety critical systems it is reasonable to argue that no event should be unpredicted and that schedulability should be guaranteed before execution [4]. This implies the use of a static scheduling algorithm. Dynamic approaches do, nevertheless, have an important role.

- They are particularly appropriate to soft systems;
- They could form part of an error recovery procedure for missed hard deadlines;
- They may have to be used if the application's requirements fail to provide a worst case upper limit (for example the number of planes in air traffic controls area).

A scheduler is static and offline if all scheduling decisions are made prior to the running of the system. A table is generated that contains all the scheduling decisions for use during run-time. This relies completely upon a priori knowledge of process behaviour. Hence this scheme is workable only if all the processes are effectively periodic.
An online scheduler makes scheduling decisions during the run-time of the system. It can be either static or dynamic. The decisions are based on both process characteristics and the current state of the system. This is difficult for systems which have non-periodic processes [5]. Schedulers may be pre-emptive or non-pre-emptive. The former can arbitrarily suspend a process's execution and restart it later without affecting the behaviour of that process (except by increasing its elapse time). Pre-emption typically occurs when a higher priority process becomes runnable. The effect of pre-emption is that a process may be suspended involuntarily. Non-pre-emptive schedulers do not suspend processes in this way. This is sometimes used as a mechanism for concurrency control for processes executing inside a resource whose access is controlled by mutual exclusion. Hybrid systems are also possible. A scheduler may, in essence, be pre-emptive but allow a process to continue executing for a short period after it should be suspended. This property can be exploited by a process in defining a non-preemptable section of code. The code might, for example, read a system clock, calculate a delay value and then execute a delay of the desired length. Such code is impossible to write reliably if the process could be suspended between reading the clock and executing the delay. These deferred pre-emption primitives must be used with care. The resulting blocking must be bounded and small typically of the same magnitude as the overhead of context switching.

The **rate monotonic scheduling algorithm (RMS)** is a fixed priority scheduling algorithm which consists of assigning the highest priority to the highest frequency tasks in the system. At any time, the scheduler chooses to execute the task with the highest priority [4]. By specifying the period and computational time required by the task, the behaviour of the system can be categorized apriori. One **problem** with the rate monotonic algorithm is that the schedulable bound is less than 100%. The schedulable bound of a task set is defined as the maximum CPU utilization for which the set of tasks can be guaranteed to meet their deadlines [4]. This means that as the task graph is designed with number of processes, the system should be able to meet the entire task by meeting their deadline with the complete usage of CPU resource.
Conditions for Rate Monotonic scheduling

- Priority assignment based on rates of tasks.
- Higher rate task assigned higher priority.
- Schedulable utilization = 0.693 (Liu and Leyland).
- If $U < 0.693$, schedulability is guaranteed.
- Tasks may be schedulable even if $U > 0.693$

The **Earliest-Deadline-First Scheduling Algorithm** (EDF) uses the deadline of a task as its priority. The task with the earliest deadline has the highest priority, while the task with the latest deadline has the lowest priority. One advantage of this algorithm is that the schedulable bound is 100% for all task sets. Secondly, because priorities are dynamic, the periods of tasks can be changed at any time [4]. A major **problem** with the EDF algorithm is that there is no way to guarantee which tasks will fail in a transient overload situation. In many systems, although the average case utilization is less than 100%, it is possible that the worst-case utilization is above 100%, leaving the possibility of one or more tasks failing. In such cases, it is desirable to control which tasks fail and which succeed during such a transient overload. In the RM algorithm, low priority tasks will always be the first to fail. However, no such priority assignment exists with EDF, and thus there is no control of which task fails during a transient overload. As a result, it is possible that a very critical task may fail at the expense of a lesser important task.

Conditions for EDF scheduling

- Priority assignment based on absolute deadlines of tasks
- Shorter the absolute deadline, higher the priority
- Schedulable utilization = 1

The **Minimum-Laxity-First Scheduling Algorithm** (MLF) assigns a laxity to each task in a system, and then selects the task with the minimum laxity to execute next. Laxity is defined as follows:
Laxity = deadline – current time – CPU_time_needed
Laxity is a measure of the flexibility available for scheduling a task. A laxity of T_l means that even if the task is delayed by l time units, it will still meet its deadline. A laxity of zero means that the task must begin to execute now or it will risk failing to meet its deadline. The main difference between MLF and EDF is that MLF takes into consideration the execution time of a task, which EDF does not do. Like EDF, MLF has a 100% schedulable bound, but there is no way to control which are guaranteed to execute during a

transient overload [4]. The next section, present the MUF algorithm, which allows the control of task failures during transient overload, while maintaining the flexibility of a dynamic scheduler, and 100% schedulable bound for the critical set.

Multi-Paradigm Strategy

The **maximum-urgency-first scheduling algorithm (MUF)** is a combination of fixed and dynamic priority scheduling, also called mixed priority scheduling [2]. With this algorithm, each task is given urgency. The urgency of a task is defined as a combination of two fixed priorities, and a dynamic priority. One of the fixed priorities, called the criticality, has higher precedence over the dynamic priority. The other fixed priority, which we call user priority, has lower precedence than the dynamic priority. The dynamic priority is inversely proportional to the laxity of a task.

The *MUF algorithm* consists of two parts. The first part is the assignment of the criticality and user priority, which is done apriori. The second part involves the actions of the MUF scheduler during run-time. The steps in assigning the criticality and user priority are the following [3]:

1. As with RM, order the tasks from shortest period to longest period.
2. Define the critical set as the first N tasks such that the total worst-case CPU utilization does not ex-ceed 100%. These will be the tasks that do not fail, even during a transient overload of the system. If a critical task does not fall within the critical set, then period transformation, as used with RM can also be used here.
3. Assign high criticality to all tasks in the critical set, and low criticality to all other tasks.
4. Optionally assign a unique user priority to every task in the system.

Before its cycle, each task must specify its desired start time, deadline time, and worst-case execution time. Later it is showed that step 1 can be relaxed, but at the increased risk of a low-criticality task failing to meet its deadline. Whenever a task is added to the ready queue, a reschedule operation is performed [1].

The *MUF scheduler* is used to determine which task is to be selected for execution, using the following algorithm:

1. Select the task with the highest criticalness.
2. If two or more tasks share highest criticalness, then select the task with the highest dynamic priority (i.e. minimum laxity). Only tasks with pending deadlines have a non-zero dynamic priority. Tasks with no deadlines have a dynamic priority of zero.
3. If two or more tasks share highest criticalness, and have equal dynamic priority, then the task among them with the highest user priority is selected.
4. If there are still two or more tasks that share highest criticalness, dynamic priority, and highest user priority, then they are serviced in a first-come-first-serve manner.
5. The optional assignment of unique user priorities for each task ensures that the scheduler never reaches step 4, thus providing a deterministic scheduling algorithm.

4. Proposed Work

The Strategy pattern in Zen, a RTORB, proposes a static uniprocessor scheduling strategies [5]. The proposed framework aims at extending the CORBA scheduling service on top of Zen ORB, to bring about efficient and intelligent task scheduling in a distributed environment considering schedulability, load balancing and security constraints. The main components of the scheduling service include the global scheduler, local scheduler, profiler and the local dispatcher. The operations of the scheduling service for static and dynamic set of tasks are explained.

Operation of the scheduler for a static set of tasks

Figure 1 shows the operation of the scheduling service for a static set of tasks. The various components of this service are to be implemented as CORBA objects. The functionality of these components is explained below.

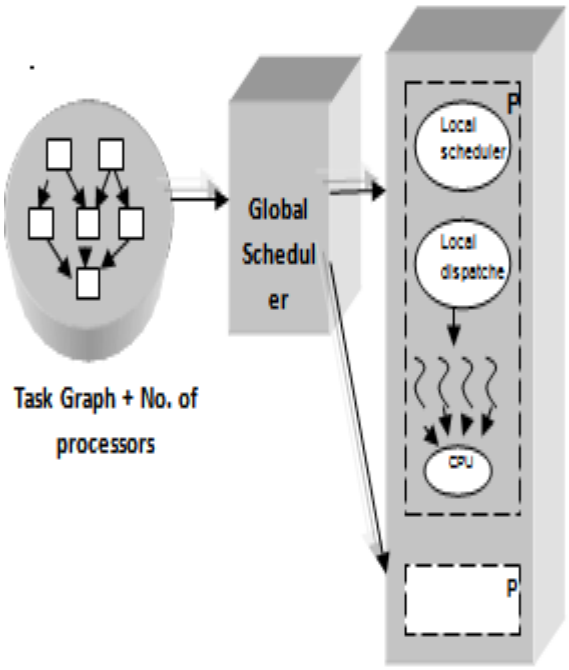

Figure 1: Operation of the static scheduling service

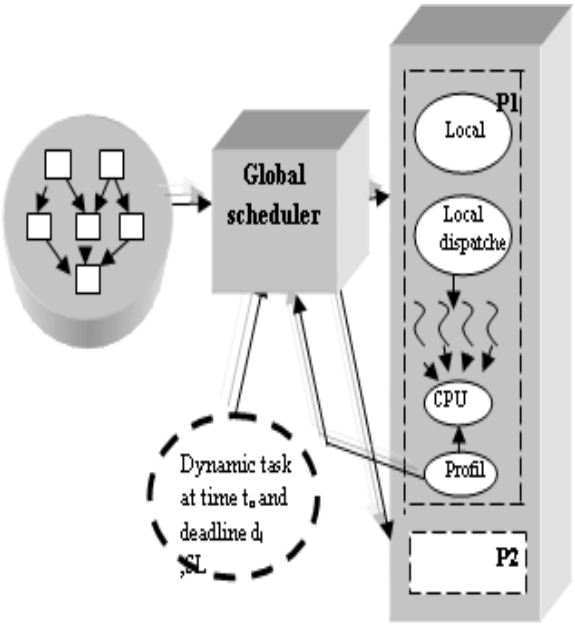

Figure 2: Operation of the dynamic scheduler

Global Scheduler- The purpose of the global scheduler is to allocate tasks to different processors in the distributed system**.** This scheduler schedules the tasks based on initial priorities given by the application programmer. It accepts a task graph as its input. A task graph defines the number of tasks to be executed and also specifies their execution time, arrival time, deadline, level and criticality. The global scheduler then decides the allocation of the set of tasks to the processors based on the processor loads.

Local Scheduler- A local scheduler is implemented to schedule the tasks for each processor. The processors are labelled as P1, P2 illustrated in Fig. 1. It uses Maximum Urgency First (MUF) to schedule the set of task it receives.

Local dispatcher- It selects the most eligible task from the list, allocates a thread for the task and assigns it to the processor. It also decides on the choices of pre-emption based on priorities.

Operation of the scheduler for a dynamic set task
Since RT CORBA deals with real time systems, which accepts dynamic tasks, a scheduler should be capable of accepting and scheduling dynamic tasks. The various components of this scheduler as shown in Figure 2 should be implemented as CORBA objects. The functionality of component is explained below.

Global Scheduler- As new tasks arrive, the scheduler makes decisions dynamically by balancing the load on the resources. Load balancing is done by profiling the load in different processors in the distributed system. The processor with minimum load and satisfying the security constraint is allocated the new task. This processor, now checks for schedulability of the task in the local scheduler. The Global Scheduler uses ML to schedule the tasks. It uses schedulability, load balancing and security constraints.

Local Scheduler- A local scheduler is implemented to schedule the tasks for each processor. When a new dynamic task is allocated to the local scheduler, it checks for the schedulability of the task. It then executes Maximum Urgency First (MUF) to reschedule the task set.

Local dispatcher- It selects the most eligible task from the list and assigns it to the processor. If the new task has higher priority than the currently executing task, the current task is pre-empted and the new task is given the thread to execute.

5. Expected Result

The performance of the proposed scheduling framework is to be evaluated based on the following criteria:

- Performance characteristics of the proposed fast multi paradigm algorithm.
- Trade off of the multiple objectives in the algorithm.
- Load balancing achieved.
- Performance of the scheduler for uniform task distribution.
- Performance of the scheduler for normal task distribution.

Result Analysis- Where x-axis is number of tasks and y-axis is amount of time in ms. When numbers of tasks are increasing the scheduling performance has been increased such as RMS and EDF alone, but RMS with multi paradigm technique increased the performance of distributed embedded system. The entire algorithm tested using ZEN.

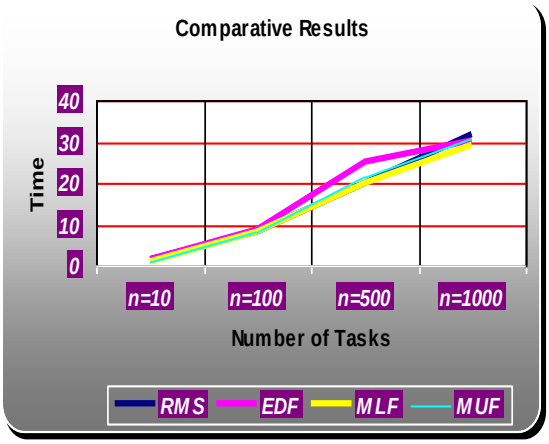

Figure 3: Zen developed by using Real Time Java Specification. So our application run any real time embedded system with minimum modifications

6. Conclusion and Future Work

The proposed CORBA-based RT scheduling framework schedules heterogeneous tasks onto heterogeneous processors in a distributed computing system. It aims at providing efficient schedules and adapting to varying resource availability. The functionality of the scheduler is to be tested for two different types of random task distributions, namely, uniform and normal distributions, each with thousands of different randomly generated sets of tasks. Since the proposed scheduler considers load balancing between different processors, it can create better schedules and reduce the make span. It is more suitable for real-world use because it considers properties of distributed systems, such as load balancing, security and variable availability heterogeneous processors, which other algorithms for the task scheduling problem do not consider. The experimental outcome demonstrates the better performance of MLF by analyzing the time taken for the execution of several numbers of tasks and further it can be compared with the combination of RMS and MLF which is the combination of static and dynamic strategies to reach the best performance.

References

[1] Christopher D. Gill and Douglas C. Schmidt "An Efficient Distributed Scheduling Framework for CORBA Application," IEEE/ACM Trans. On Distributed Systems, Vol. 10, No.4, pp.183-197, April 2008.

[2] Pyarali I., Schmidt D.C., Cytron R.K, "Techniques for Enhancing real-time CORBA Quality of Service" Proceedings of the IEEE Publication Date: July 2003, Volume: 91, Issue:7,pp.1070- 1085.

[3] Krishnamurthy Y., Pyarali I, Schmidt D.C., "The Design and implementation of Real-Time CORBA 2.0: Dynamic Scheduling in TAO" Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. 10th IEEE Publication Date: 25-28 May 2004, On page(s): 121- 129.

[4] Jose M. Lo´pez, Jose´ L. Dı´az, and Daniel F. MLrcı´a, "Minimum and Maximum Utilization Bounds for Multiprocessor Rate Monotonic Scheduling" IEEE Trans. on Parallel and Distributed Systems, Vol. 15, No. 7, July 2004.

[5] Arvind Krishna, Douglas C. Schmidt, and Raymond Klefstad, "Enhancing Real-Time CORBA via Strategies and Real-Time Java features," Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS), May 23-26, 2004, Tokyo, Japan.

Bineta Tresa Mathew was born in Kuwait on 29th Jan 1984. She received the B.E. degree in Information Technology from Anna University, India in 2006 and M.E. degree from Karunya University, India in 2009 in Computer Science Engineering. She is working as Assistant Professor in the department of Computer Science and Engineering, Birla Institute of Technology, Offshore campus, RAK, UAE. Her research interest includes genetic algorithm.