JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 2, March–April 2006

A Novel Real Time Scheduling Framework for CORBA-Based Applications

G.Sudha Sadasivam, Assistant Professor, CSE department, PSG College of Technology, Coimbatore – 641004, Tamil Nadu, India.
Geetha Rani, Ramya and Saranya, UG students, CSE department, PSG College of Technology, Coimbatore – 641 004, Tamil Nadu, India.

Abstract

Many real-time application domains can benefit from flexible and open distributed architectures, such as those defined by the CORBA specification. Although CORBA is well-suited for conventional request/response applications, it is not yet suitable for real-time applications due to the lack of key Quality of Service (QoS) features and performance optimizations. This paper explains the design and implementation of a real-time scheduling service that can provide QoS guarantees for deterministic real-time CORBA applications. The scheduling service deals with multiprocessor task scheduling in a distributed environment. It involves a global scheduler and a local scheduler. The global scheduler schedules the tasks based on the Schedulability analysis and the request priority of the tasks. It uses genetic algorithms for intelligent multiprocessor scheduling of the tasks. The Run-Time Scheduler or the Local Scheduler maps client requests for particular servant operations into priorities that are understood by the local OS dispatcher. This Real-time Scheduling Service is implemented as a CORBA object that is responsible for allocating system resources to meet the QoS needs of the applications. It thus brings about extensibility and interoperability in a distributed object system.

1 INTRODUCTION

Distributed real-time embedded (DRE) systems are becoming increasingly widespread and important. Common DRE [Klefstad02] systems include Telecommunication networks, tele-medicine, manufacturing process automation and defense 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. This paved way for Real-Time CORBA (RT CORBA).

Cite this column as follows: G. Sudha Sadasivam, Geetha Rani, Ramya and Saranya: "A Novel Real Time Scheduling Framework for Corba-Based Applications", in *Journal of Object Technology*, vol.5, no.2, March-April 06, pp. 171-188 http://www.jot.fm/issues/issue_2006_03/article4

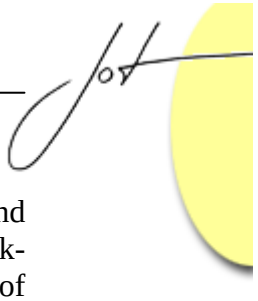
CORBA is distribution middleware that provides run-time support to automate many distributed computing tasks, such as connection management, object marshaling / demarshaling, object demultiplexing, language and platform independence, load balancing, fault-tolerance, and security. Real-time CORBA adds QoS control capabilities to regular CORBA which include improving application predictability by bounding priority inversions and managing system resources end-to-end. Real-time CORBA [OMG99] 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. 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) [RTJ01], 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 than regular Java.

ZEN [Krishna03] 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 pluggability. Zen employs the Micro-kernel architecture. It has eight core ORB services [Klefstad02] 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.

Kokyu [Gill01] developed on Adaptive Communication Environment (ACE) at the Washington University, is a middleware real-time scheduling framework. Kokyu is a portable middleware scheduling framework designed to provide flexible scheduling and dispatching services within the context of higher-level middleware. Kokyu currently provides real-time scheduling and dispatching services for TAO's [Schmidt97] real-time CORBA Event Service, which mediates supplier-consumer relationships between application operations. It does not consider multiprocessor scheduling strategies. The "Evolution Scheduler" [Erad97] designed at the Illinois Institute of Technology is based on Genetic Algorithms and Evolutionary Programming. It works with original Linux priority scheduler and provides intelligent scheduling. The evolution scheduling is acceptable for the real-time system and useful for its performance to achieve fair and effective scheduling.

A fault-tolerant extension to the myopic scheduling algorithm [Manimaran96] using two versions of the tasks is found to be effective. GA can be used for static scheduling [Ahmad01] where the state of all tasks and system resources must be known a priori and cannot change. These schedulers are limited to specific problems and systems. So a



heavily-loaded processor that might have more tasks in the near future is speculated and less number of tasks is allocated to it [Wakatani02]. Palis [Palis02] proposed a task-scheduling algorithm that provides quality of service guarantees in the context of reservation-based preemptive real-time systems. A Resource Manager System (RMS) [Azzedin02] is aware of the security requirements of the resources and tasks that it can perform. Task allocations are done to minimize the security overhead. However, the allocation algorithm is developed in the context of non-real-time applications, and it is not suitable for real-time systems where tasks' deadlines have to be considered. Dynamic GA schedulers create schedules at runtime, with knowledge about the properties of the system and tasks possibly not known in advance, allowing for variable system and task properties to be considered [Page04]. Communications costs and the possibility of variable processing resources are not considered. A scheduling strategy [Page05] to operate in an environment with dynamically changing resources and capable of adapting to variable communication costs and variable availability of processing resources has been implemented and found to perform well. Xie [Xie05] proposes a novel dynamic scheduling algorithm with security awareness, which is capable of achieving high quality of security for real-time tasks while improving resource utilization. Bagchi [Bagchi02] has used genetic algorithms to schedule multi satellite operations. In IIT, Mumbai [Burns00] scheduling is done in Flexible Real-time Systems using values. Work on dynamic real-time scheduling with periodic and non-periodic task arrivals is being done at Indian Institute of Technology, Delhi [Janin00].

TAO [Schmidt97], a real-time Object Request Bus (ORB) developed by Object Computing Research group at the Washington University and Vanderbilt University aims at optimizing collocation, common cases, ORB protocol overhead, Portable Object Adapter (POA) demultiplexing and strategy patterns for scheduling to make CORBA suitable for real time applications. It uses Rate Monotonic Scheduling (RMS), Maximum Laxity First (MLF) and Maximum Urgency First (MUF) algorithms for scheduling. It does not consider multiprocessor scheduling strategies. 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. 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 is implemented using distributed object technology, it provides facility for extensibility and modification. It thus reduces the software development lifecycle time.

2 DESIGN OF THE PROPOSED SCHEDULING FRAMEWORK

The Strategy pattern in Zen [Klefstad02] a research RTORB, proposes only static uniprocessor scheduling strategies. 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.

The main components of the scheduling service include the global scheduler, local scheduler, profiler and the local dispatcher. The operation of the scheduling framework for static and dynamic set of tasks is explained in this section.

Operation of the scheduler for a static set of tasks

Fig 1 shows the operation of the scheduling service for a static set of tasks. This service is implemented using Zen ORB. The various components of this service are implemented as CORBA objects. The functionality of these components is explained below

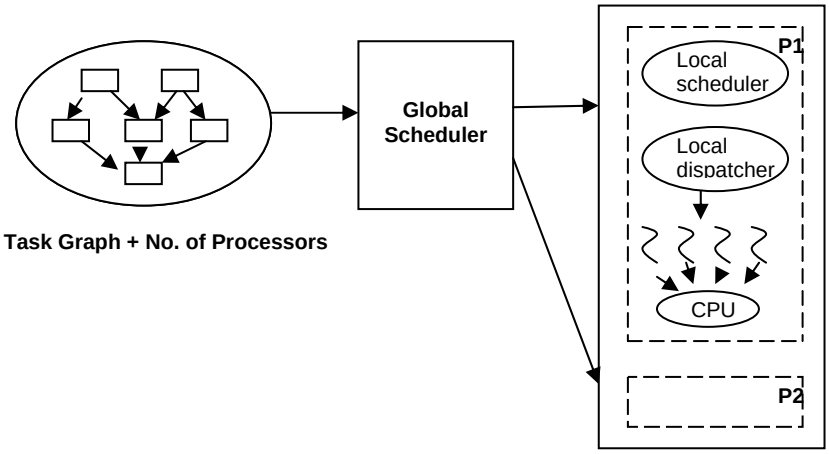


Figure 1: Operation of the scheduling service for a static set of tasks

1. **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. Genetic Algorithm is used to divide the input tasks into various sets based on their levels, arrival time, deadline, criticality and execution time. The global scheduler then decides the allocation of the set of tasks to the processors based on the processor loads.
2. **Local Scheduler-** A local scheduler is implemented to schedule the tasks for each processor. The processors are labeled as P1, P2 (fig 1). It uses any one of the three scheduling algorithms namely Earliest Deadline First (EDF), Minimum Laxity First (MLF), Maximum Urgency First (MUF) selected by the user to schedule the set of task it receives.
3. **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 preemption based on priorities.

Operation of the scheduler for a dynamic set of tasks

Since RT CORBA deals with real time systems, which accepts dynamic tasks, a dynamic scheduler is also designed. The various components of this scheduler shown in fig. 2 are implemented as CORBA objects. The functionality of each component is explained below.

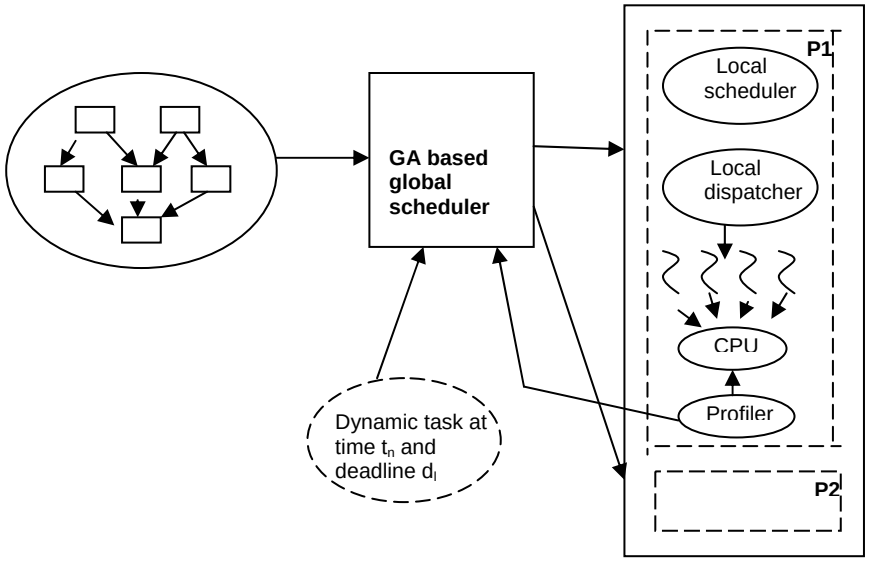


Figure 2: Operation of the Dynamic Scheduler

1. **Profiler-** A profiler in the local system takes care of monitoring the resource usage and supplies this information to the resource manager. It finds out the CPU utilization of each processor. This information is used by global scheduler to schedule the tasks.
2. **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 is allocated the new task. This processor, now checks for schedulability of the task in the local scheduler.
3. **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 chooses the best of the three scheduling algorithms, namely, Earliest Deadline First (EDF), Minimum Laxity First (MLF), and Maximum Urgency First (MUF) to reschedule the task set.
4. **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 preempted and the new task is given the thread to execute. Proper algorithms to prevent priority inversion are also implemented.

Genetic Algorithm

Genetic algorithm is used in the global scheduler, to adapt to varying resource environments and to produce near-optimal schedules. The processors of the distributed system are heterogeneous. The available network resources between processors in the distributed system can vary over time. The availability of each processor can also vary over time, since the processors are not dedicated and have other tasks that partially use their resources. Tasks are indivisible, independent of all other tasks, arrive randomly, and can be processed by any processor in the distributed system. When tasks arrive they are placed in a queue of unscheduled tasks. Batches of tasks from this queue are scheduled on the processors, based on their load. The allocation of tasks to the processors is carried out by a genetic algorithm. The algorithm is illustrated in figure 3

Require: Initial Population P;

repeat

Select basis B from Population P;

Generate new population P' by crossing individuals in the basis B;

Mutate individuals in P';

P ← P';

Evaluate the quality of the population P';

until stop criterion

Figure 3: Overview of the Genetic Algorithm

Encoding

Each possible solution in the population is represented as a chromosome. It represents the order in which the tasks are to be executed for a particular processor. Each gene in the chromosome represents the task. The tasks in the gene are encoded as a numeric value, which represents the task number.

For example, let us consider that there are three processors. Processor P1 is allocated the tasks T1, T3 and T5; Processor P2 is allocated the tasks T2 and T7 and Processor P3 is allocated the tasks T4 and T6. The encoding is represented in the Fig 4. Here one dimension of the chromosome is used for a particular processor. It is thus represented as a two dimensional array, with the first dimension representing the processors and the second dimension representing the task set allocated to the processors.

P1	1	3	5
P2	2	7	
P3	4	6	

Figure 4: Encoding of schedule

Fitness Function:

A fitness function attaches a value to each individual in the population. It indicates the goodness of the schedule. The objective function calculates the total execution time of the set of tasks allocated to each processor. The fitness function calculates the average of the total execution time of the set of tasks allocated to the processors.

- The objective function is represented as follows,

$$OF(P_i) = arr(t_1) + \sum_{i=1}^{n} E(t_i)$$

where,

i→ task
j → processor
n → Number of tasks
$arr(t_i)$ is arrival time of task t_i
$exec(t_i)$ is execution time of task t_i

$$E(t_i) = \begin{cases} exec(t_i) & \text{if } arr(t_i) \le fin(t_{i-1}) \\ arr(t_i)+exec(t_i) & \text{otherwise} \end{cases}$$

- The fitness function is represented as follows,

$$\text{Min} \left\{ \frac{\sum_{j=1}^{m} OF(Pj)}{m} \right\}$$

where

m → Number of processors

Selection

Elitism method of selection which is widely used by a number of researchers for task scheduling is adopted. This method finds out the best set of tasks for which the total execution time is minimum. This method also retains the best chromosome combination over a number of generations.

Crossover

After the selection process is complete crossover is performed on the remaining set of tasks, which meets the schedulability constraint. This operation can generate a better task set with minimum execution time for the processors. The process is explained with an example as shown in figure 5a.

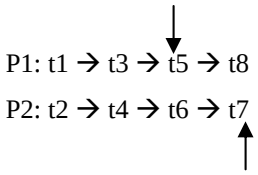


Figure 5a: Illustration of crossover operation

Crossover is performed on the set of tasks allocated to the processors on randomly selected levels. The down and up arrows specify the chosen levels as shown in figure 5a. All the set of tasks allocated to the processors are swapped from the specified levels. The result of crossover is shown in figure 5b.

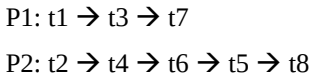


Figure 5b: Result of crossover operation

Mutation

Swap mutation is performed on the set of tasks that have been crossed over. It is used to generate a new combination of task set with minimum execution time, for the processors. This process is explained with an example as shown in figure 6a.

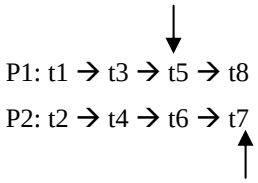


Figure 6a: Illustration of swap mutation operation

Mutation is performed on the set of tasks allocated to the processors on randomly selected levels. The down and up arrows specify the chosen levels. Only the task corresponding to that particular level is swapped. The result of mutation is shown in figure 6b.

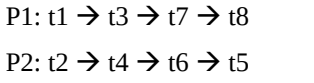


Figure 6b: Result of swap mutation operation

After performing selection, crossover and mutation, the best set of tasks with minimum execution time is sent to the processors. A local scheduler runs in every processor in the distributed system. The set of tasks is then locally scheduled using Earliest Deadline First (EDF), Minimum Laxity First (MLF) and Maximum Urgency First (MUF) algorithms.

Local Scheduler

Three types of scheduling algorithms are implemented in each processor to schedule the tasks. A local scheduler uses one of the following three algorithms to schedule the tasks. This section gives a detailed explanation of the three algorithms.

1) Earliest Deadline First:

Earliest Deadline First (EDF) is a dynamic scheduling strategy that orders the dispatch of operations based on time-to-deadline. Operation executions with closer deadlines are dispatched before those with more distant deadlines. The EDF scheduling strategy is invoked whenever a dispatch of an operation is requested. The new dispatch may or may not preempt the currently executing operation, depending on the mapping of priority components into thread priorities. Figure 7 explains the EDF algorithm with an example.

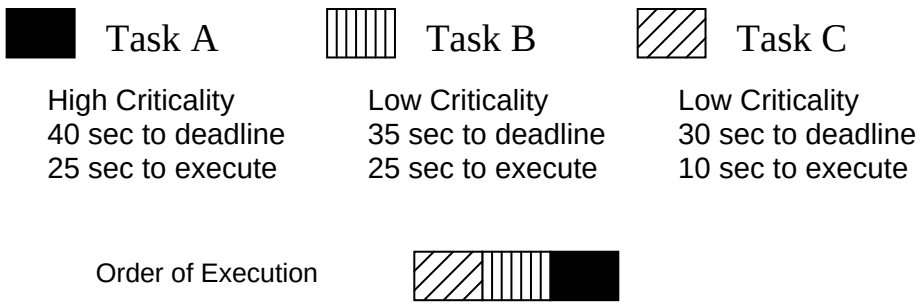


Figure 7: Order of execution of tasks in EDF

Fig 7 shows three tasks A, B, C with their deadline, execution time and criticality. The deadline of these three tasks are considered by the EDF scheduling algorithm and task C with the minimum deadline executed first. Task B is executed next followed by task A as shown in the Fig 7. EDF algorithm can perform well for different types of task distributions. Further, it has a higher CPU utilization. As EDF involves a higher overhead of evaluation at run-time and has no control over the tasks whose deadline is missed, MLF is used.

2) Minimum Laxity First:

Minimum Laxity First (MLF) refines the EDF strategy by taking into account the operation execution time. It dispatches the operation whose laxity is least. Laxity is the difference between the time-to-deadline and the remaining execution time.

Using MLF, it is possible to detect an operation that will not meet its deadline, prior to the deadline itself. If this occurs, a scheduler can reevaluate the operation before

allocating the CPU for the remaining computation time. One strategy is to simply drop the operation whose laxity is not sufficient to meet its deadline. This strategy may decrease the chance that subsequent operations will miss their deadlines, especially if the system is overloaded transiently. Fig 8 explains MLF algorithm with an example.

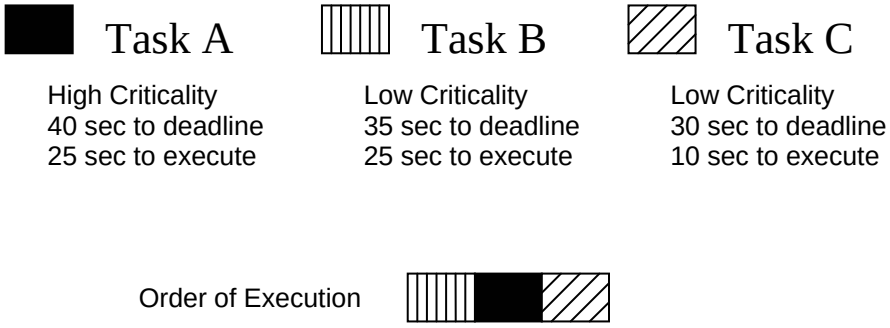


Figure 8: Order of execution of tasks in MLF

In MLF algorithm, the task with minimum laxity is executed first as shown in fig 8. Hence, task B gets executed first followed by task A and then by task C. MLF can detect an operation will not meet its deadline prior to the deadline itself. The main disadvantage of this algorithm is that it requires higher overhead to evaluate at run-time.

3) Maximum Urgency First:

The Maximum Urgency First (MUF) scheduling strategy is based on the criticality of the tasks. Task with the maximum urgency gets executed first. MUF is the default scheduler for the real-time operating system (RTOS). TAO supports a variant of MUF in its strategized CORBA scheduling service framework. MUF can assign both static and dynamic priority components. This hybrid priority assignment in MUF overcomes the drawbacks of the individual scheduling strategies by combining techniques from each. Fig 9 explains the MUF algorithm with an example. In this algorithm, task A having higher criticality is allocated first to the processor, followed by task B, based on laxity and then task C.

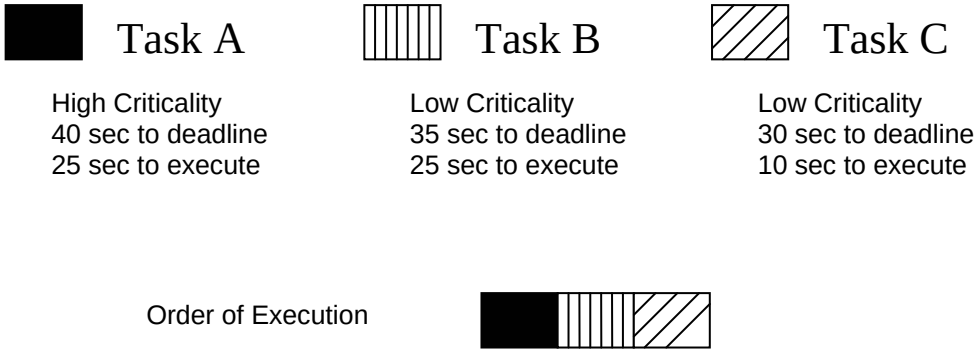


Figure 9: Order of execution of tasks in MUF

Real Time I/O Subsystem:

As this scheduling framework works on top of Zen, it uses the real-time IO subsystem of Zen [Klefstad02]. Zen's real-time I/O (RIO) subsystem minimizes priority inversion and hidden scheduling that arise during protocol processing. Zen minimizes priority inversion by pre-allocating a pool of kernel threads dedicated to protocol processing. These kernel threads are co-scheduled with a pool of application threads. The kernel threads run at the same priority as the application threads.

To ensure predictable performance, the kernel threads belong to a real-time I/O scheduling class. This scheduling class uses rate monotonic scheduling (RMS) to support real-time applications with periodic processing behavior. Once a real-time I/O thread is admitted by the OS kernel, Zen's RIO subsystem performs the following actions:

1. Computing its priority relative to other threads in the class, and
2. Dispatching the thread periodically so that its deadlines are met.

If all operations can be scheduled, the Scheduling Service assigns a priority to each request. At runtime, these priority assignments are then used by Zen's Runtime Scheduler. The Run-time Scheduler maps client requests for particular servant operations into priorities that are understood by the local endsystem's OS thread dispatcher. The dispatcher then grants priorities to real-time I/O threads and performs preemption so that schedulability is enforced at runtime.

Profiler

The scheduling framework uses Xprof profiler. Xprof profiler is a HotSpot profiler. HotSpot works by running Java code in interpreted mode, while running a profiler in parallel. The HotSpot profiler looks for "hot spots" in the code. Hot spots are methods that the JVM spends a significant amount of time running, and then compiles those methods into native generated code. Xprof tells the profiler to keep a record of the profile information, and output the information at termination to stdout.

The performance of the scheduler for varying set of tasks is illustrated in section 3.

3 EXPERIMENTAL RESULTS AND ANALYSIS

The scheduling framework has been tested and the results are illustrated in this section. The first subsection analyses the performance of the genetic algorithm. Second subsection gives an analysis of the performance of the local schedulers implemented using different scheduling algorithms. It also shows the performance of the schedulers for different number of tasks and for different types of task distributions. The third subsection analyses the overall performance of the scheduling framework.

Performance of the genetic algorithm:

Multiprocessor task scheduling is performed in the global scheduler using GA. This algorithm allocates the tasks to the processors, taking into consideration, the load of the processors and the deadline of the tasks. Fig 10 shows the performance of the GA for different number of tasks. It is found that the algorithm converges quickly, when the number of tasks is less. As the number of tasks increases, the convergence time also increases linearly.

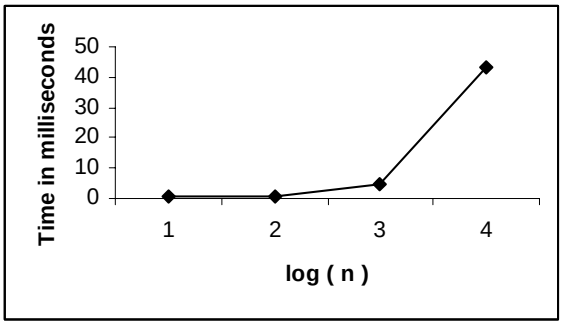


Figure 10: Efficiency of GA for different number of tasks (n) in the task set

Analysis of the Local Schedulers:

Every processor in the distributed system acts as a local scheduler. Three different scheduling algorithms have been implemented for local schedulers. They are Earliest Deadline First, Maximum Urgency First and Minimum Laxity First scheduling algorithms.

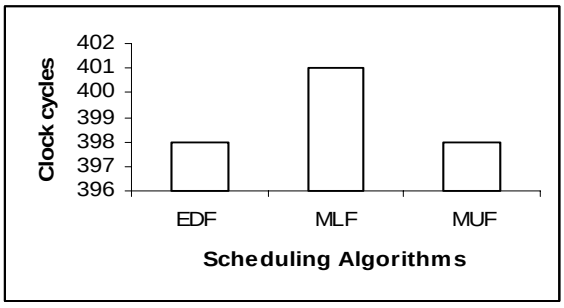


Figure 11: Efficiency of Scheduling Algorithms for 100 tasks

Figure 11 shows the performance of EDF, MLF and MUF a task set of size 100. It is found that the MUF performs well because it takes the criticality of the tasks and orders

the tasks. EDF considers the deadlines of the tasks and orders the tasks, whereas MLF considers the laxity of the tasks. It is found that EDF performs better than MLF.

Figures 12 a, b and c show the performance of EDF, MLF and MUF for task sets of different sizes. It is seen that the time taken for scheduling the tasks increases linearly with the number of tasks in the task set. The experimental results have been taken for 10, 100 and 1000 tasks in the task set.

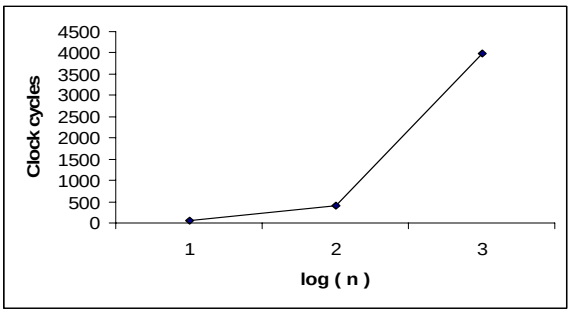


Figure 12a: Efficiency of EDF for different number of tasks

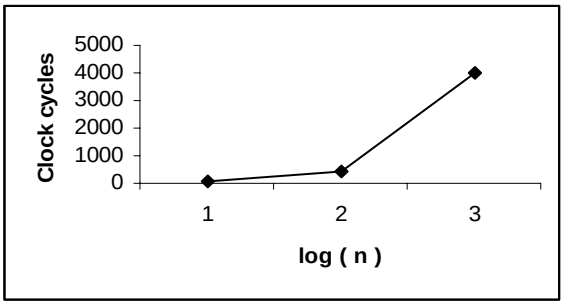


Figure 12b: Efficiency of MLF for different number of tasks

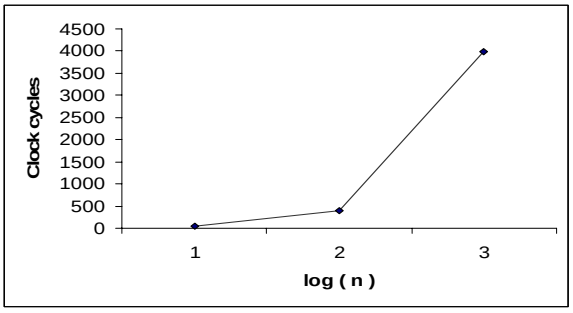


Figure 12c: Efficiency of MUF for different number of tasks

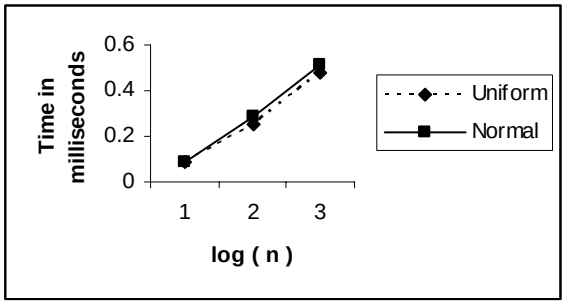


Figure 13: Performance of local schedulers for uniform and normal task distribution

Fig 13 shows the performance of the local schedulers for two different types of task distributions, namely, uniform and normal task distributions. It is found that when the task set is uniformly distributed, the local schedulers perform better, than when it is normally distributed. Also as the number of tasks increase, this difference becomes more prominent.

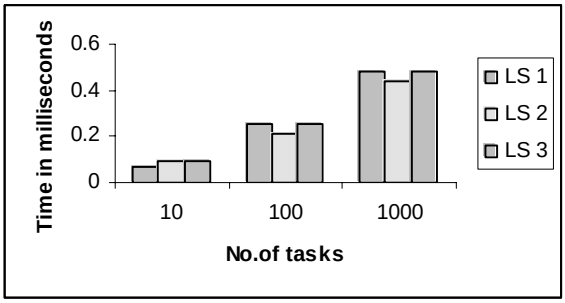


Figure 14a: Performance of different local schedulers in the distributed environment when tasks are in uniform distribution

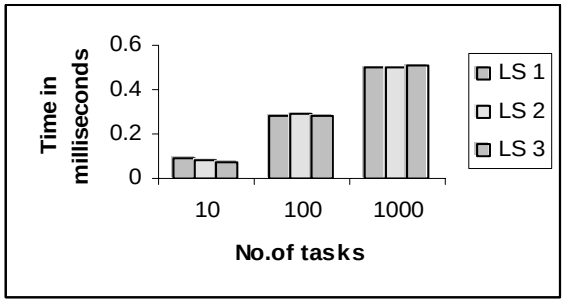


Figure 14b: Performance of different local schedulers in the distributed environment when tasks are in Normal distribution

Fig 14 shows the performance of the local schedulers for different types of task distributions. From Figs 14 a and b, it is seen that the load is equally distributed among the local schedulers, and hence all the schedulers take almost the same time to complete. The results also show that the load balance in the local schedulers is not affected by the size of the task set. Hence the load is uniformly distributed among the local schedulers. It

is also noticed that the time taken to execute the task set, even when the size of the task set becomes very large, is only increased by a small amount, since the task set is distributed among the different processors. In case of normal distribution, the unbalance in distribution of task set among the local schedulers is more when compared to that of the uniform distribution.

Analysis of the overall performance of the scheduling framework:

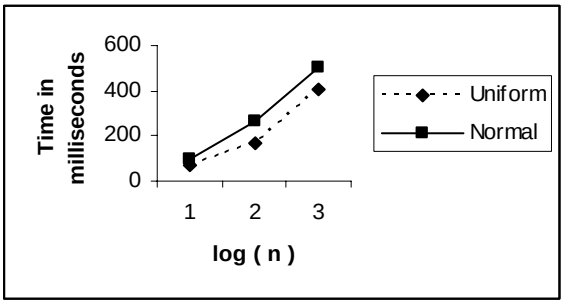


Figure 15: Performance of the Distributed Multiprocessor Scheduling System for uniform and normal distribution of tasks

The performance of the scheduling framework is shown in Fig 15. The system has been tested for uniform and normal distribution of tasks and for varying number of tasks. It is found that the task set with uniform distribution takes less processing time when compared to the task set with normal distribution, because there is more probability of the load being evenly distributed among the local schedulers. It is also observed that as the number of tasks increases exponentially, the processing time of the tasks, only increases linearly.

4 CONCLUSION

A CORBA-based RT scheduling framework has been developed to schedule heterogeneous tasks onto heterogeneous processors in a distributed computing system. It provides efficient schedules and adapts to varying resource availability. This includes processing resources and load balancing constraints. The algorithm also fully utilizes the dedicated processor running the scheduler. The GA employs a list scheduling heuristics to create a well-balanced randomized initial population. The fitness function utilizes the relative error metric internally to find schedules with a low makespan. Elitism type of selection is used to exploit past results to direct the search for efficient schedules. The crossover method promotes exploration of the search space, with random swaps and random re-balancing of processor queues.

The scheduler has been tested under different scenarios. The generality of the scheduler is tested for two different types of random distributions, namely, uniform and

normal distributions, each with thousands of different randomly generated sets of tasks. Experimental results show that the scheduling framework schedules the tasks on different processors equally, thus achieving a balanced load between the processors.

Since the scheduler considers load balancing between different processors, it can create better schedules and reduce the makespan. It is more suitable for real-world use because it considers properties of distributed systems, such as load balancing and variable availability heterogeneous processors, which other algorithms for the task scheduling problem do not consider.

REFERENCES

[Klefstad02] Raymond Klefstad, Douglas C. Schmidt, and Carlos O'Ryan, "Towards Highly Configurable Real-time Object Request Brokers", in the Proceedings of IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC), Washington DC, 2002.

[OMG99] Object Management Group, "Real time CORBA specifications", http://www.omg.org/pubs/docs/ptc/99-05-03.ps, 1999

[RTJ01] Real Time Java Expert Group, "Real Time Specification for Java", https://rtsj.dev.java.net/, Dec 2001.

[Krishna03] Arvind S. Krishna, Raymond Klefstad Douglas C. Schmidt Angelo Corsaro, "Towards Predictable Real-time Java Object Request Brokers", in the Proceedings of the 9th Real-time/Embedded Technology and Applications Symposium, 2003.

[Gill01] Christopher D. Gill, David L. Levine, and Douglas C. Schmidt The Design and Performance of a Real-Time CORBA Scheduling Service, Real-Time Systems: the International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, guest editor Wei Zhao, Vol 20, No 2, March 2001.

[Schmidt97] Douglas C Schmidt , Levin and Sumedh Mungee, "The Design of TAO Real time ORB", Journal of Computer Communications,1997.

[Erad97] Erad and Jinlong Lin, "Evolutionary Computation for Scheduling Controls in Concurrent Object-Oriented Systems", in the proceedings of CAINE-97 (ISCA 10th International Conference on Computer Applications in Industry and Engineering, 1997.

[Manimaran96] G Manimaran, C. Siva Ram Murthy, "A New Study for Fault-tolerant Real-time Dynamic Scheduling Algorithms", Third International Conference on High-Performance Computing (HiPC '96), December 19 - 22, India, 1996.

[Ahmad01] I.Ahmad, Y Kwok and M. Dhodhi, “Scheduling parallel programs using genetic algorithms”, Solutions to parallel and distributed computing problems, chapter 9, John Wiley and Sons, 2001, pp. 231–254.

[Wakatani02] Akiyoshi Wakatani, “Dynamic Task Scheduling with Speculative Approach and its application to Image Segmentation”, Konan University, 2002

[Palis02] M.A. Palis, “Online Real-Time Job Scheduling with Rate of Progress Guarantees," in the Proc. the 6th Int’l Symp. Parallel Architectures, Algorithms, and Networks, 2002.

[Azzedin02] Farag Azzedin, Muthucumaru Maheswaran, “Towards Trust- Aware Resource Management in Grid Computing Systems,” Proc. the 2nd IEEE/ACM Int’l Symp. Cluster Computing and the Grid, Germany, May 2002.

[Page04] A. J. Page and T. J. Naughton. “Framework for task scheduling in heterogeneous distributed computing using genetic algorithms”, In 15th Artificial Intelligence and Cognitive Science Conference, Ireland, pages 137–146, 2004.

[Page05] AJ. Page and Thomas J. Naughton “Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing”, Proceedings of the 19th IEEE/ACM International Parallel and Distributed Processing Symposium, Denver USA, April 2005.

[Xie05] T Xie, Andrew Sung and Xiao Qin, “Dynamic Task Scheduling with Security Awareness in Real-Time Systems”, Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05), the 4th Int'l Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems, IEEE/ACM, April 4-8, 2005.

[Bagchi02] Tapan P Bagchi, IIT Kanpur, "Development of Scheduling Algorithm and its Software Implementation for Multi-satellite Operations Scheduling by Genetic Algorithms", sponsored by ISRO, system implemented by ISTRAC, ISRO in 2002.

[Burns00] A. Burns, D. Prasad, A. Bondavalli, F. Di. Giandomenico, K. Ramamritham, J. A. Stankovic, and L. Strigini, “The Meaning and Role of Value in Scheduling Flexible Real-time Systems”, Journal of Systems Architecture, Vol. 46, 2000, pp. 305-325.

[Janin00] Prof BN Jain, dept of CSE, IIT, Delhi, “Real-time scheduling with periodic and non-periodic task arrivals”, 2000.

About the authors

Ms **G Sudha Sadasivam** is working as an Assistant Professor in Department of Computer Science and Engineering in PSG College of Technology, India. Her areas of interest include, Distributed Systems, Real Time Systems, Distributed Object Technology and Compiler Design. She has published 10 papers in referred journals and 15 papers in National and International Conferences. She is the co-coordinator of AICTE project titled, "Stub Code Optimization in Distributed Embedded Applications". You may contact her at sudhasadhasivam@yahoo.com

Geetha Rani Ravindranathan is working as a software analyst for Manhattan Associates Development Center in Bangalore, which provides solutions for Supply Chain Execution. She graduated from one of the most renowned colleges of Tamil Nadu, PSG College of Technology. Her areas of interest include Distributed Computing and Computer Networks. You may contact her at rgeetharani@gmail.com.

Ramya Gopalanis is working as an Assistant System Engineer for Tata Consultancy Services in Chennai. She graduated from one of the most renowned colleges of Tamil Nadu, PSG College of Technology. Her areas of interest include Operating System and Data Structures. You may contact her at ramya1.g@tcs.com

Saranya Suresh is working as a Project Engineer for Wipro Technologies in Chennai. She graduated from one of the most renowned colleges of Tamil Nadu, PSG College of Technology. Her areas of interest include Computer Networks and Operating System You may contact her at saranya.suresh@wipro.com