Scheduling Algorithm for Real-Time Applications in Grid Environment

Lichen Zhang
Department of Computer Science and Technology
Guangdong University of Technology
510090 Guangdong, China
Email: lchzhang@gdut.edu.cn

Abstract: As the associated human community, instruments, and resources required for data processing become increasingly distributed, real-time online instrument systems connected by wide area networks will be the norm for scientific, medical, and similar data-generating systems. Such systems have rigorous Quality of Service (QoS) objectives. They must behave in a dependable manner, must respond to threats in a timely fashion and must provide continuous availability, even within hazardous and unknown environments. Furthermore, resources should be utilized in an efficient manner, and scalability must be provided to address the ever-increasing complexity of scenarios that confront such systems. Advances in networking infrastructure have led to the development of a new type of "computational grid" infrastructure that provides predictable, consistent and uniform access to geographically distributed resources such as computers, data repositories, scientific instruments, and advanced display devices . Such Grid environments are being used to construct sophisticated, performance-sensitive applications in such areas as supercomputer-enhanced instruments, desktop supercomputing, tele-immersive environments, and distributed super computing. Such applications designed to execute on "computational grids" frequently require dynamic scheduling of multiple resources in order to meet performance requirements. Motivated by these concerns, we have developed a general scheduling algorithm in Grid environment. In this paper, we propose a three level dynamic scheduling method. A set of thresholds and information list are used to collect the information of the grid. In first level, as task arrive in the one node of the system. the scheduler uses the information about the task and the state of node , and attempts to guarantee the new task by the execution of this node. If this task can be scheduled on this node and the time constraints can be met, we execute this task on this node. If this task cannot scheduled, the second level scheduling is started., we use the information list of the cluster that is connected closely by the node to find whether this cluster can accept this new task and meet its time constraint. If the cluster can accept this new task, we transfer this new task to one underloaded node of this cluster and executes this new task on this underloaded nodded. If new task cannot be accepted by the cluster closely connected by the node, the third level scheduling is started., we use the information list to find a remote cluster in the grid to accept this task and execute it on the remote cluster in the grid.

1. Introduction

As the associated human community, instruments, and resources required for data processing become increasingly distributed, real-time online instrument systems connected by wide area networks will be the norm for scientific, medical, and similar data-generating systems. Such systems have rigorous Quality of Service (QoS) objectives. They must behave in a dependable manner, must respond to threats in a timely fashion and must provide continuous availability, even within hazardous and unknown environments. Furthermore, resources should be utilized in an efficient manner, and scalability must be provided to address the ever-increasing complexity of scenarios that confront such systems. The difficulties in engineering such systems arise from several phenomena, one of the most perplexing being the dynamic environments in which they must function. Systems which operate in dynamic environments may have unknown worst-case scenarios, may have large variances in the sizes of the data and event sets that they process (and thus, have large variances in execution latencies and resource requirements), and cannot be characterized (accurately) by constants, by intervals or even by time-invariant statistical distributions. This environment gives rise to the need for a variety of capabilities: dynamically schedulable resources, easily administered and enforced use conditions and access control for all elements, systems designed to adapt to varying conditions in the distributed environment, automated control and guidance systems that facilitate remote (in time, space and scale) operations, and a myriad of reservation and scheduling capabilities for all of the resources involved.

Advances in networking infrastructure have led to the development of a new type of "computational grid" [1][2][16][17] in frastructure that provides predictable, consistent and uniform access to geographically distributed resources such as computers, data repositories, scientific instruments, and advanced display devices . Such Grid environments are being used to construct sophisticated, performance-sensitive applications in such areas as supercomputer-enhanced instruments, desktop supercomputing, tele-immersive environments, and distributed super computing.

Recent advances in real-time systems technology have given us many good schemes for scheduling hard real-time applications[5][6][8][9][21][22]. A weakness shared by most existing schemes is that schedulability of each application can be determined only by analyzing all applications in the system together, i.e., by a global schedulability analysis. While the necessity of global schedulabiiity analysis imposes no serious problem when the system is closed, it does so when the system is open. Here, by ***closed system, we*** mean one in which detailed timing attributes of all real-time applications on each processor are known. Often times, the applications are developed together, and the schedulability of every combination of applications that can run at the same time is determined a ***priori.*** In contrast, in a grid ***environment,*** applications may be developed and validated independently. During run-time, the user may request the start of a real-time application whose schedulability has not been analyzed together with currently executing applications. The system must determine whether to accept the request and admit the new application. It usually admits a new real-time application only when the application and all the existing real-time applications are schedulable. A global schedulability analysis as an acceptance test is sometimes not feasible because many characteristics of real-time applications in the system are unknown. Even when it is feasible, such an acceptance test can be time consuming when the applications are multi-threaded and complex.

In this paper, we propose a three level dynamic scheduling method. A set of thresholds and information list are used to collect the information of the grid. In first level, as task arrive in the one node of the system. the scheduler uses the information about the task and the state of node , and attempts to guarantee the new task by the execution of this node. If this task can be scheduled on this node and the time constraints can be met, we execute this task on this node. If this task cannot scheduled, the second level scheduling is started., we use the information list of the cluster that is connected closely by the node to find whether this cluster can accept this new task and meet its time constraint. If the cluster can accept this new task, we transfer this new task to one underloaded node of this cluster and executes this new task on this underloaded nodded. If new task cannot be accepted by the cluster closely connected by the node, the third level scheduling is started., we use the information list to find a remote cluster in the grid to accept this task and execute it on the remote cluster in the grid.

2 . Grid Architecture

Grid Architecture comprises four general types of componts[1][13]: Grid Fabric, Grid middleware, Grid developments and Tools and Grid application and portals. There are three main issues that characterize computational grids:

1.**Heterogeneity:** a grid involves a multiplicity of resources that are heterogeneous in nature and might span numerous administrative domains across wide geographical distances.

2.**Scalability**: A grid might grow from few resources to millions. This raises the problem of potential performance degradation as a Grids size increases. Consequently, applications that require a large number of geographically located resources must be designed to be extremely latency tolerant.

3.**Dynamicity or Adaptability:** in a grid, a resource failure is the rule, not the exception. In fact, with so many resources in a Grid, the probability of some resource failing is naturally high. The resource managers or applications must tailor their behaviour dynamically so as to extract the maximum performance.

The steps necessary to realize a computational grid include:

- The integration of individual software and hardware component into a combined networked resource.
- The implementation of middleware to provide a transparent view of the resources available.
- The development of tools that allows management and control of grid applications and infrastructure.
- The development and optimization of distributed applications to take advantage of the resources.

The componts that are necessary to form a grid are following:

- Grid Fabric: it comprises resources-specific and site-specific mechanisms which, when in place, enable or facilitate the creation of higher-level distributed "Grid Services." Examples of Fabric mechanisms might include network quality of service support in routers and end systems; resource management interfaces supporting advance reservation, allocation, monitoring, and control of computers, storage systems, etc.; instrumentation interfaces; high-speed network interfaces; and specific implementations of network protocols. These Grid Fabric mechanisms serve to Grid-enable basic resources, augmenting or complementing the basic communication functionality provide by the Internet Protocol suit to allow, for example, the coordinated allocation of computers, networks, and storage systems.
- Grid Services or Middleware: it offers core services such as remote process management, co-allocation of resources, storage access, information security, authentication, and Quality of Service (QoS) such as resource reservation and trading.
- Grid Development Environments and Tools: These offer high-level services that allows programmers to develop applications and brokers that act as user agents that can manage or schedule computations across global resources.
- Grid Applications and Portals: They are developed using grid-enabled languages such as HPC++, and message-passing systems such as MPI. Specific Gird-aware application are implemented in terms of various Application Toolkit components, Grid Services, and Grid Fabric mechanisms.

The management of processor time, memory, network, storage, and other component in a grid is clearly very important. The overall aim is to efficiently and effectively schedule the applications that need to utilize the available resources in the metacomputing environment [14]. From a user's point of view, resource management and scheduling should be transparent; their interaction with it being confined to a manipulating mechanism for submitting their application. It is important in a grid that a resource management and scheduling service can interact with those that may be installed locally. The architectural model of resource management systems is influenced by the way the scheduler is structured. The structure of scheduler depends on the number of resources on which jobs and computations are scheduled, and the domain in which resources are located. Primarily, there are three different models for structuring schedulers:

- **Centralized scheduling model**: This can be used for managing single or multiple resources located either in a single or multiple domains. It can only support uniform policy and suits well for cluster management systems such as Condor, LSF, and Condine. It is not suitable for grid resource management systems as they are expected to honor policies imposed by resource owners.
- **Decentralized scheduling model:** In this model schedulers interact among themselves in order to decide which resource should be applied to the jobs being executed. In this scheme, there is no central leader responsible for scheduling, hence this model appears to be highly scalable and fault-tolerant. As resource owners can define the policy that schedulers can enforce, the decentralized scheme suits grid systems. Because the status of remote jobs and resources is not available at single location, the generation of highly optimal schedule is questionable. This model seems difficult to implement in the grid environment, as domain resource owners do not agree on a global policy for resource management.
- **Hierarchical scheduling model:** This model fits for grid systems at it allows remote resource owners to enforce their own policy on external users. This model looks like

a hybrid model, but appears more like centralized model and therefore suits grid.

A idea grid environment will therefore provide access to the available resources in a seamless manner such that physical discontinuities such as differences between platforms, network protocols, and administrative boundaries become completely transparent. In essence, the grid middleware turns a radically heterogeneous environment into a virtual homogeneous one.

With the recent adoption of the CORBA Component Model (CCM) [12] application designers now have a standard way to implement, manage, configure, and deploy components that implement and integrate CORBA services. CCM standard not only enables greater software reuse for servers, it also provides greater flexibility for dynamic configuration of CORBA application. Thus, CCM appears to be well –suited for real-time applications based on the Computing Grid.

Meeting the QoS requirements of distributed real-time applications requires an integrated architecture that can deliver end-end QoS support at multiple levels in real-time and embedded systems. Distributed object computing (DOC) middleware based on the real-time CORBA (RT_CORBA) [12] offers solutions to some resource management challenges and developers of real-time systems, particularly those systems are designed using dynamic scheduling techniques. The OMG has formed a Special Interest Group (RT SIG) [23] with the goal of extending the CORBA standard with real-time extensions. One of the requirements that has been established by the RT SIG involves providing global real-time scheduling to support the enforcement of end-to-end timing constraints on client server interactions. The specify requirements for scheduling in real-time CORBA involve the need for a global priority that has meaning across the entire distributed systems. A scheduling service in a real-time CORBA systems must provide this global priority, based on the timing constraints expressed by the client's method invocations, and on the server's own timing requirements. The service must also ensure that the global priority can be mapped to the scheduling environments of all local operating systems involved in the execution. Finally if the real-time CORBA systems is a dynamic systems, the scheduling service must ensure that the global priority correctly reflects the requirements of the associated execution for the duration of the execution. That is, it may be necessary to modify the value of the global priority based on changes that occur in the system.

The Dynamic Real-Time CORBA system provides the groundwork for a full dynamic real-time CORBA design. However, there is still much work to be done. A number of dynamic scheduling algorithms have been proposed, RT-CORBA must be enforced with them.

3. Dynamic Scheduling algorithms

Dynamic scheduling [3][5][18][19][20][23] in real-time systems involves dynamically making a sequence of decisions concerning the assignment of system resources to real-time tasks. System resources include processors, memory, and shared data structures. Tasks may have arbitrary time constraints, different important levels, and fault tolerance requirements. Unfortunately, making these scheduling decisions is difficult, partly because the decisions must be made without the full knowledge of the future arrivals of tasks and partly because scheduling has to deal with many complex issues, e.g.,, multiprocessors and fault tolerance.

Ramamritham's algorithm[23]

Ramamritham et al. proposed combining local and global scheduling approaches in distributed systems for both periodic tasks and aperiodic tasks. In their approach, periodic tasks are assumed to be known a priori and can always be scheduled locally. On the other hand, an aperiodic task may arrive at a node at any time and will be scheduled locally on the node if its deadline can be met there; otherwise, the task will be transferred toa romote node, which was called global scheduling. If non of the remote nodes can guarantee the deadline of this aperiodic task, it will be rejected and may seriously affect the system performance. Hence, the main effort in [23] was to design a heuristic, global scheduling policy so as to reduce the number of rejected aperiodic tasks. Three algorithms, were used to select a remote node for each aperiodic task which cannot be guarantee an aperiodic task locally, it will attempt to locate a remote node which can guarantee the task.

Xu's algorithm [22]

J.Xu proposed a approach that integrates run-time scheduling with pre-run-time scheduling, for the scheduling of both periodic and asynchronous processes with hard or soft deadlines, and different a priori knowledge of the process characteristics. A guiding principle for this approach is that the scheduling algorithm should exploit to a maximum extent of knowledge about system processes characteristics that are available to the scheduler both before run-time and during run-time.

Deng's algorithm[21]

Z.deng and J.W.S.Liu Proposed a two-level hierarchical scheme for scheduling an open system of multi-threaded, real-time applications on a single processor. It allows different applications to be scheduled according to different scheduling algorithms.

Chang's Algorithm[19]

H.Y. Chang proposed a dynamic scheduling algorithm for a soft real-time system. The algorithm has a two phase polling strategy and is based on the "Shortest Processing Time First' local scheduling policy. Unlike hard real-time systems in which a late job may entail catastrophic outcomes, a soft real-time system functions correctly as long as the deadline miss ration and the expected lateness are below pre-defined levels.

Shin's algorithm[20]

In the paper [20], K.G. Shin and Y. C. Chang proposed a load sharing method with state-change broadcast (LSMSCB) for distributed real-time systems, in which each node maintains state information of only a small set of nodes in its physical proximity, called a buddy set. The load state of a node is defined by three thresholds: TH_u, TH_f andTH_v. A node is said to be underloaded if its queue length (QL) is less than or equal to TH_u, ***medium-loaded***: if $TH_u < QL \leqslant TH_f$, ***fully load***:

if $TH_f < QL \leqslant TH_v$, ***overloaded:*** if $QL > TH_v$. When a node becomes fully loaded (underloaded) duo to the arrival and/or transfer (completion) of tasks, it will broadcast its change of state to all the other nodes in its buddy set. Every node that receives this information will update its state information by eliminating the fully loaded node from, or adding the underloaded node to, its ordered list (called a preferred list) of available receivers. An overloaded node can select, without probing other nodes, the first underloaded node in its preferred list and transfer a task to that node. Moreover, the buddy set of the nodes in one buddy set are different but are not disjoint, thus allowing the surplus tasks in a buddy set to be transferred to many different buddy sets, i.e., system-wide load sharing. As a result, this method is shown to enable tasks to be completed before their deadlines with much higher probability than other known methods.

Corsaro's algorithm[24]

In open distributed real-time and embedded (DRE) systems, different ORB endsystems may use different scheduling policies. To ensure appropriate end-to-end application behavior in an open architecture, however, DRE systems must enfore an ordering on activities originating in an endsystems and activites that migrate there, based on the relative importance of these activities. This paper describes the meta-programming techniques applied in Juno, which is an extension to Real-Time CORBA that enhances the openness of DRE systems with respect to thire scheduling policies by enabling dynamic ordering of priority equivalence classes.

Foster's algorithm[25]

I.Foster proposed an approach to QoS that combines features of both reservations and adaptation to enhancing the end-to-end performance of network applications. At the core of this approach is a QoS architecture in which resources are enhanced with:
Online control interfaces that allow applications, or agents acting on their behalf, to modify resource characteristics dynamically;
Sensors that allow applications to detect when adaptation is required; and
Decision procedures that support the expression of a rich set of resource management policies.

4. Dynamic Real-Time Scheduling in Grid Environment

Our model of a Grid is as follows. The grid consists of M clusters connected by Internet. Each cluster consists N_i nodes which a communication network. The network is assumed to be logically connected in that every node can communicate with every other node. One node in Cluster M_k can communicate another node in cluster M_j by serverk in cluster M_k and server j in cluster M_j. A stream of jobs is submitted locally to node k. We assume that the nodes are heterogeneous in the sense that each node may have a different arrival rate of externally submitted jobs, but homogeneous in the sense that a job submitted at any node in the cluster k in the Grid can be processed at any other node in the cluster k. We also assume that a job on one node of cluster k can be sent to another node of cluster j to process. We consider a real-time system in which a job is lost if it cannot started within a given time constraint, i.e., within a fixed time after its arrival. If a constraint of the job on one node in cluster k cannot be met locally, however, it may transfer it to another node in cluster k if a underloaded node can be found in cluster k. If a underload node cannot be find in cluster k, it may transfer it to another node in cluster j if cluster j is underloded.

In dynamic scheduling algorithms, there are five separable and yet integrated components:

(1) Local scheduling determines the sequencing of local job
(2) Information policy dictates the methods of exchanging load status among nodes and clusters in the Grid.
(3) Initial policy decide when to initiate a migration request.
(4) Candidate selection policy determines how to choose a job for migration.
(5) Location policy defines the terms under which two processors may transfer a job.

In this paper, we proposed a dynamic scheduling algorithms based on Kang G. Shin's method [20]. Each cluster maintains state information of only a small set of clusters in its physical proximity, called a buddy set. The load state of a cluster is defined by three thresholds: CTH_u, CTH_f and CTH_v.
There are four states for each cluster:

- ***underloded:*** A cluster is said to be underloaded if its number of jobs (tasks) (N_c) is less than or equal to CTH_u,
- ***medium-loaded***: if$CTH_u < N_c \leqslant CTH_f$
- ***fully load***: if $CTH_f < N_c \leqslant CTH_v$
- ***overloaded:*** if $N_c > CTH_v$.

For each node in any cluster, The load state of a node is also defined by three thresholds: NTH_u, NTH_f and NTH_v.

- ***underloded:*** A node is said to be underloaded if its number of jobs (tasks) (N_n) is less than or equal to NTH_u,
- ***medium-loaded***: if$NTH_u < N_n \leqslant NTH_f$
- ***fully load***: if $NTH_f < N_n \leqslant NTH_v$
- ***overloaded:*** if $N_n > NTH_v$.

Each cluster has one server that can collect the state information of other clusters and the state information of each node of this cluster. When a node becomes fully loaded (underloaded), it will send its change of state to the server, and server of the cluster will update its state information table. If the cluster changes his state, e.g, from underload to fully loaded, it will broadcast its change of state to all the other clusters in its buddy set. Every cluster that receives this information will update its state information by eliminating the fully loaded cluster from, or adding the underloaded cluster to, its ordered list (called a preferred list) of available receivers. Thus dynamic scheduling method is organized at three level. In first level (node scheduling), as task arrive in the one node of the system. The scheduler uses the information about the task and the state of node , and attempts to guarantee the new task by the execution of this node. If this task can be scheduled on this node and the time constraints can be met, we execute this task on this node. If this task cannot scheduled, the second level scheduling (cluster scheduling) is started., we use the state information table of the cluster that is connected closely by the node to find whether this cluster can accept this new task and meet its time constraint. If the cluster can accept this new task, we transfer this new task to one underloaded node of this cluster and executes this new task on this underloaded node. If new task cannot be accepted by this cluster, the third level scheduling (grid scheduling) is started., we use the information list to find a remote cluster in the grid to accept this task and execute it on the remote cluster in the grid.

Algorithm

Initiation:

- Creat the thresholds CTH_u, CTH_f and CTH_v on the server of each cluster
- Creat the state information table , buddy set and preferred list on the server of each cluster
- Creat the thresholds NTH_u, NTH_f and NTH_v on each node

Node Sheduling

```
While a task arrive on one node in a cluster
  do
    if this node is idle
      then
          this task will be executed immediately
          send a message to the server to update
            the state information table.
      else
        If QL>NTH_v
          then
             goto cluster scheduling
          else
             make this task in queue
             send a message to the server to
             update state information table.
While a task finished his execution
  do
    if queue of this node is not idle
      then
          task in the head of queue will be
            executed immediately
          send a message to the server to update
          state information table
      else
          send a message to the server to update
          the state information table.
```

Cluster Scheduling

```
If a message to update information arrives
  then
      update state information table.
If the state of cluster changes,
  then
      goto grid scheduling.
If overloaded task is transferred to the server
  then
      if find a underloaded node in local cluster
        then
          migrate overloaded task to this Node to
          execution, update state information table
      else
          goto grid scheduling
```

Grid Scheduling

```
If the state of cluster changes,
  then
      broadcast its change of state to all other clusters
      in its buddy set.
If this cluster is overloaded,
  then
      transfer to overloaded tasks to the first
      underloaded cluster in its preferred list and
      transfer a task to that cluster.
If one overloaded task from other clusters arrives,
  Then
      goto cluster scheduling
```

Since communication cost must be kept below a given required level while minimize the resultant overhead , the main issues of the scheduling algorithms are how to define state of each node and state of each cluster, how to collect state information, and how to redistribute loads among nodes and clusters, such that overloaded nodes will locate underloaded nodes to share their loads in local cluster or remote cluster with a very high probability. Buddy sets, preferred lists, state information table, and threshold patterns are the most important features to resolve these issues.

The buddy set of a cluster is a set of clusters in its physical proximity. Since state information for different clusters is exchanged only within a buddy set and since a constant buddy set size of 10 to 115 nodes is shown to work well regardless of the system size, the communication overhead is reduced to a constant from O(N2), as compared to the case when state information is exchanged in entire systems of N clusters. In order to avoid more than one overloaded cluster "dumping" their loads on one underloaded cluster or surplus tasks being confined in a certain region, the clusters in a buddy set are ordered into a preferred list such that each cluster will be selected as the k_{th} preferred cluster by one and only one other cluster. I t has been shown that the preferred lists can effectively solve both the coordination and congestion problems, thus meeting task deadlines with a high probability.

The exact analysis of scheduling algorithm is difficult. The composite task arrival process at a node or a server of cluster is composed of the local (external) task arrivals and task transfers, the latter of which is itself a composite process of task transfers from different nodes or clusters. One difficulty in estimating the composite task arrival rate is that the transferred-in task arrival process (and thus the composite arrival process) may not be Poisson even if the local task arrival process is Poisson. This is because the probability of sending a task to (or receiving a task from) a node or a cluster depends on the state of both nodes or clusters, making the splitting process non-Poisson, and task transmission times may not be exponentially distributed, making the process of transferred-in tasks non-Poisson. Furthermore, even if we assume the composite arrival process to exhibit behaviors similar to a Poisson process, the transferred-in task arrival rate from a node is not known due to the dynamic change of system state. In this paper, approximate method was used in the algorithm analysis. .Bayesian estimation is used for the on-line computation of the composite task arrival rate on a node. We consider Poisson external task arrivals and we further approximate the composite task arrival process to be Poisson . This approximation rests on a general result of renewal theory which states that the superposition that the arrival rate of tasks with increasingly many component processes yields a Poisson process.

5. Conclusion

The Computational Grid provides a promising platform for the efficient execution of complex real-time applications. Scheduling such application is challenging because target resources heterogeneous ,because their load and availability varies dynamically, and because the tasks have the strict timing constraints. In this paper, we proposed an dynamic real-time scheduling algorithm for real-time application running on the Grid. The scheduling model was organized in three levels: node scheduling, cluster scheduling and grid scheduling. This model fits for grid systems and it integrates local scheduling and global scheduling. This model looks like a hybrid model, but suits grid.

Acknowledgments

This work is partly supported by the National Natural Science Fund, Natural science Fund of Guangdong province, "Thousand, Hundred, and ten" outstanding person fund of Education Department of Guangdong Province, Natural science fund of Education Department of Guangdong Province.

References

[1]Foster, Buliding the Grid: An integrated services and toolkit architecture for next generation networked applications, Http://www.computingportals.org/gce-papers/building_the_grid.htm.

[2]M.Baker and G.Fox, Metacomputing: Harnessing informal supercomputers, High performance cluster computing: Architecture and Systems, R.Buyya (ed.), Volume 1, Prentice Hall PTR, NJ, USA, 1999.

[3]L.R.Welch et al., Specification and modeling of dynamic, distributed real-time systems, Proceedings of the 19th IEEE Real-Time Systems Symposium, 72-81, IEEE Computer Society Press, 1998.

[4] D.C.Schmidt et al., The design and performance of real-time object request brokers, Computer Communication, Vol.21, pp.294-324, Apr.1998.

[5]R.M. Kavi, "Real-time systems: Abstractions, languages, and Design Methodologies", IEEE Computer Society Press, 1992.

[6]J.A. Stankovic and K.Ramarithm, Hard real-time systems, IEEE computer Society, Order Number819, 1988.

[7]S.P. Reiss, 'PECAN: program development systems that support multiple views', IEEE Trans. on Software Eng., SE-11, (3), 1985.

[8]A.M.V. Tilborg and C.M. Koob, Foundations of real-time computing: Formal specifications and methods, Kluwer Academic publishers, 1991.

[9]J.Xu and D.L.Parnas, On statisfying timing constraints in hard real-time systems, Proceeding of the ACM SIGSOFT'91 Conference on Software for Critical Systems, 1991.

[10] L.Zhang, et al., Methodology of real-time system design using multiprocessors, Microprocessors and Microsystems, Vol 17, No 4, May 1993.

[11] L.Zhang and B.Chaib, A design methodology for real-time to be implemented on multiprocessors, the Journal of system and software, April, 1996, 33: 37-56.

[12] L.Dipippo et al., Expressing and enforcing timing constraints in a dynamic Real-Time CORBA system, Real_time Systems, Vol.16, issue 2/3, May 1999.

[13]M. Baker et al., The grid: international efforts in global computing, Internal conference on advances in infrastructure for electronic business, science, and education on the Internet, Italy, 2000.

[14]R. Buyya et al., An architecture for a resource management and scheduling system in a Global computational grid, HPC ASIA'2000, China, IEEE CS Press, USA, 2000.

[15]S.Chapin et al., A grid resource management architecture, Grid Forum scheduling working group, nov. 1999.

[16] J.Dongarra, An overview of computational grids and survey of a few research projects, Symposium on global information processing technology, Japan, 1999.

[17] I. Foster and C. Kesselman, The grid: Blueprint for a new computing infrastructure, Morgan kaufmann publishers, USA, 1999.

[18]H.Casanova et al., Adaptive scheduling for task Farming with Grid middleware, International Journal of Supercomputer applications and high performance computing, 1999.

[19]H.Y.Chang, Distributed scheduling under deadline constaints, a comparision of sender-initiated and receiver-initiated approaches, Proceedings of IEEE Real-Time Systems Symposium, 175-180, IEEE Computer Society Press, 1986.

[20]K.G.Shin and Y.C. Chang, Load sharing in distributed real-time systems with state change broadcast. IEEE Trans. Comput. C-38, 8 (august 1989), 1124-1142.

[21]Z.Deng and J.W.S.Liu, Scheduling real-time application in an open environment, Proceedings of the 18th IEEE Real-Time Systems Symposium, 308-319, IEEE Computer Society Press, 1997.

[22]J.Xu and D.L.Parnas, Integrating run-time scheduling and pre-run-time scheduling of real-time processes, Real-time programming 1998, 73-80, Elsevier Scice Ltd.

[23]Ramamritham et al., Distributed scheduling of tasks with deadlines and resources requirements, IEEE Trans. Comput. C-38, 8 (August 1989), 110-1123.

[24]A.Corsaro et al., Formalizing meta-programming techniques to reconcile heterogeneous scheduling policies in open distributed real-time systems, Proceedings of the 3rd international symposium on distributed objects and application, September 8-10, 2001, Rome, Italy.

[25]I.Foster et al., A distributed resource management architecture that supports advance reservations and co-allocation, Proceedings of International workshop on quality of service, pp.27-36, June 1999.