적응형 그리드 시스템를 위한 이질적인 자원 관리

허의남*, 이웅재*, 이종숙**

Heterogeneous Resource Management for Adaptive Grid System

Eui-Nam Huh, and Woong-Jae Lee and Jong-Sook Lee

Abstract

Real-Time applications on Grid environment have several problems in terms of resource management addressed as follows;(1) dynamic resource allocation to provide QoS objectives, (2) heterogeneous resources that is different scale, or capacity in same unit, and (3) resource availability, and resource needs. This paper describes the techniques of resource manager (RM) handling above problems to support QoS of dynamic real-time applications on Grid. The contributions of this paper to solve problems are as follows: unification of dynamic resource requirements among heterogeneous hosts, control of resources in heterogeneous environments, and dynamic load balancing/sharing. Our heuristic allocation scheme works not only 257% better than random, 142% better than round robin, and 36.4% better than least load in QoS sensitivity, but also 38.6% better than random, 28.5% better than round robin, and 31.6% better than least load in QoS.

Key Words: Heterogeneous, distributed real-time systems, dynamic, worst case execution time, resource manager, and QoS.

* 서울여자대학교 정보통신공학부
** 한국정보과학기술연구원

1. Introduction

This paper describes techniques for managing heterogeneous host resources to support QoS of real-time Grid applications. Our approach is based on the path paradigm. A path-based real-time application (see [1][2]) typically consists of a detection & assessment path, an action initiation path and an action guidance path. The paths interact with the environment by evaluating streams of data from sensors, and by causing actuators to respond (in a timely manner) to events detected during evaluation of sensor data streams.

An overview of our approach for the resource manager (RM) is shown in Figure 1. The "Specification Language (Dependable Extensible Specification Language-DXSL)"is used to describe requirements of applications. The DXSL also defines information about the hosts and networks such as speed, OS type, the number of CPUs, benchmark rate, bandwidth, and interconnected equipment. The "Middleware" collects QoS metrics from the monitor, compare to the requirement, and request or control the application and resources, if QoS violations occur. The RM is the brain, which makes allocation decisions to achieve QoS objectives.

This paper focuses on the resource management component, and discusses a new technique for dynamic feasibility analysis on heterogeneous resource platforms. Most previous work in distributed real-time systems assumed that all system behaviors follow a statically known pattern (see [3] [4]). When applying the previous work to some applications (such as shipboard AAW systems [1]), problems arise with respect to scalability of analysis and modeling techniques; furthermore, it is sometimes impossible to obtain some of the parameters required by the models. In contrast, DeSiDeRaTa RM (see [2], [5]) allows the modeling of systems that work in environments that have unknown scenarios (such as battle environments) (see [6]) the dynamic path paradigm is based on obtainable parameters, since it evolved from the study of existing computer systems and the large granularity of the path makes it more scalable than task allocation approaches.

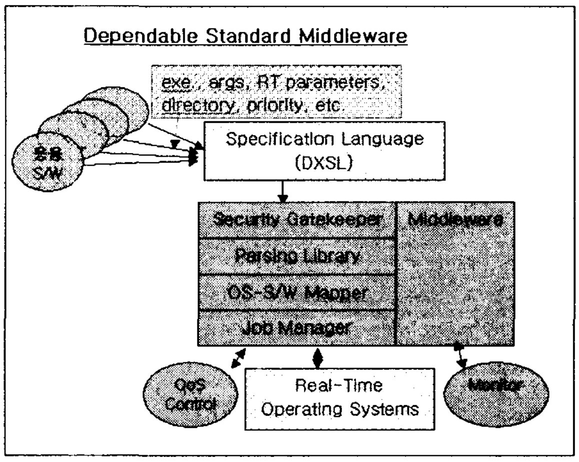

Dependable Standard Middleware
exe., args, RT parameters, directory, priority, etc.
응용 S/W
Specification Language (DXSL)
Security Gatekeeper
Parsing Library
OS-S/W Mapper
Job Manager
Middleware
QoS Control
Real-Time Operating Systems
Monitor

Figure 1. QoS and Resource Management Architecture

The new contributions of this paper are as follows: (1) unification of dynamic resource requirements among heterogeneous hosts, (2) control of resources in heterogeneous environments, and (3) dynamic load balancing/ sharing.

Section 2 shows the feasibility analysis and laxity based RM approach with system model. Section 3 shows the results of experiments. Finally, Section 4 is the summary and conclusion.

2. LaxityBasedResourceAllocation

In this section, the resource management approach is explained. Basic steps of adaptive resource management are follows: (1) Resource Requirement Analysis , (2) Resource Discovery, (3) Resource Unification, (4) Feasibility Analysis, and (5) Optimization.

These steps are explained in detail in the remainder of this section. First, a mathematical model, which is used in the detailed explanation, is presented.

Table 1. The system resource model

SYMBOL	NOTATION
a_{ij}	name of application j in path i
$tl = tl(c, P_i)$	workload or tactical load at cycle c in path i
$CUP_{obs}(a_{ij}, tl, H_k)$	the CPU user-percentage of host H_k for the application a_{ij} in path i at work load tl
$CUP_{uni}(a_{ij}, tl, H_k)$	the unified CPU user-percentage of host H_k for the application a_{ij} in path i at work load tl
$CUP(H_i, t)$	the CPU user-percentage of host H_i at time t
$CIP(H_i, t)$	the idle-percentage of host H_i at time t
$MEM_{obs}(a_{ij}, tl, H_k)$	the memory usage of application a_{ij} in path i on host H_k at work load tl
$FAM(H_i, t)$	the free-available-memory of host H_i at time t
$X_{obs}(a_{ij}, tl, H_k)$	the execution time of application a_{ij} in path i on host H_k at work load tl
$P_{obs}(a_{ij}, tl, H_k)$	the period of application a_{ij} in path i on host H_k at work load tl
$CCR(H_i)$	CPU Clock rate in MHz at host H_i
$SPECint95(H_i)$	the fixed point operation performance of SPEC CPU95 of host H_i
$SPECfp95(H_i)$	the floating point operation performance of SPEC CPU95 of host H_i
$SPEC\_RATE(H_i)$	The relative SPECCPU95 rating of host H_i
$Threshold\_CPU(H_i)$	The CPU threshold to tolerate the unified CPU resource requirement
$Threshold\_MEM(H_i)$	The memory threshold to tolerate the memory resource requirement

Table 1 shows the system resource model. a_{ij} and tl represent application and workload of an application, respectively. Indices starting with CUP stand for CPU usage. CIP is CPU idle percentage of a host. MEM and FAM relate to memory usage. X and P are the execution time and period of an application a_{ij}, respectively. CCR stands for CPU clock rate of a host. The SPEC CPU95 host benchmark consists of SPECint95 and SPECfp95 that show relative performance of fixed and floating point operations in a system. SPEC_RATE is overall relative system rank. Indices, Threshold, are certain amount of resource to tolerate different amount of resource requirement.

The steps taken by RM are now explained in detail. The resource requirement step works as follows. QM detects QoS violation by monitoring QoS of a path and each application and requests additional resources based on decisions. When a significant amount of workload is observed, QM analyzes the latency of each application. If a_{ij} uses more resources than others, or the latency of a_{ij} is higher than minimum QoS slack, then QM triggers request of additional resources with another copy of the application. This is called "scale-up" decision. When workload is not changed, but QoS violation occurs, QM triggers migration of a_{ij} running on the overloaded host. It is called "move"decision. Therefore, different resource requirements should be measured according to decisions. Hence, for "move" decision, RM measures dynamic resource requirement of CPU for the violated application using $CUP_{obs}(a_{ij}, tl, H_k) = X(a_{ij}, tl, H_k) / P(a_{ij}, tl, H_k)$. For the *"scale-up"* decision, the resource requirement is measured by $CUP_{obs}(a_{ij}, tl, H_k)$ / (number_of_replicas + 1).

```
1.  QM request resources, $CUP_{obs}(a_{ij}, tl, H_L)$, $MEM_{obs}(a_{ij}, tl, H_L)$
2.  Get the host list, HL, including host load indices, load metrics(LM)
3.  Calculate EMA of LM
4.  No_of_Candidate_Host = 0 ;
5.  Create Linked List of HL_CPU ;
6.  Create Linked List of HL_MEM ;
7.  For (k = first(HL(H_i, t)) ; k <= last(HL(H_i, t)))
8.      $CUP_{uni}(a_{ij}, tl, H_k) = CUP_{obs}(a_{ij}, tl, H_L) * SPEC\_RATE(H_L) / SPEC\_RATE(H_k)$;
9.      $Feasible_{CPU}(H_k, t) = CIP(H_k, t) - CUP_{uni}(a_{ij}, tl, H_k) - Threshold\_CPU(H_k)$;
10.     $Feasible_{MEM}(H_k, t) = FAM(H_k, t) - MEM_{obs}(a_{ij}, tl, H_k) - Threshold\_MEM(H_k)$;
11.     If(Feasible_{CPU}(H_k, t) > 0) && (Feasible_{MEM}(H_k, t) > 0 )
12.       $l_{CPU}(H_k,t) = Feasible_{CPU}(H_k, t) ) * SPEC\_RATE((H_k)$ ;
13.       $l_{MEM}(H_k,t) = Feasible_{MEM}(H_k, t) )$ ;
14.       Append $H_k$ and $l_{CPU}(H_k,t)$ to HL_CPU ;
15.       Append $H_k$ and $l_{MEM}(H_k,t)$ to HL_MEM ;
16.       No_of_Canidate_Host ++ ;
17.     // end if 11
18. Loop 7;
19. Sort HL_CPU in descending order of $l_{CPU}(H_i,t)$
20. Sort HL_MEM in descending order of $l_{MEM}(H_i,t)$
21. If ( No_of_Candidate_Host == 0 ) Return(Target_Host = first(HL_CPU))
22. Target_Host = first(HL_CPU) ;
23. While(true)
24.  If((Target_Host is Alive) && (Target_Host is in top 50th percentile of HL_MEM))
25.         Return (Target_Host) ;
26. Else Target_host = next(HL_CPU) ;
27.  Loop 23 ;
```

Figure 2. Laxity based RM Algorithm

The resource discovery step is explained here in detail. Monitoring of resource availability in dynamic environments has more difficulties than in static environments, because of unknown system activation. FAM(H_k,t), CIP(H_k,t), and CUP(H_k,t) are collected for all host "k" once per second. And these load metrics (LM) are filtered by exponential moving average (EMA) as illustrated below for CUP:

EMA(CUP(H_j,t)) = (1-b)*(CUP(H_j,t)) + b* EMA(CUP(H_j,t-1)), where, t is a cycle, and b = e^{-T} (T is constant, and T=1 is used).

Each resource has various scales and capacities even in the same unit among heterogeneous platforms. In this step, resource unification method is explained in detail.

Definition: *Resource unification* produces a canonical form of each resource metric.

RM allocates and controls resources accurately, if each resource is unified. Consider $CUP_{obs}(a_{ij}$, tl, $H_k)$ as resource requirement. To allocate the amount of the resource, RM needs to analyze the requirement and map it to target hosts. There are two approaches, static and dynamic. The static approach uses stable system information like benchmarks, or CPU clock rate. It will decide relative amount of system resources efficiently but inaccurately for dynamic environments. The dynamic approach of predicting execution time using dynamic system information has high complexity for real-time systems, as an application uses several different resources such as I/O disk, memory, and CPU, each of which has different performance among hosts. Therefore, a static approach is selected as follows. For the unification of resources, the results of a variety of realistic SPEC CPU95 will give valuable insight into expected real performance among heterogeneous hosts. However, no one benchmark can fully characterize overall system performance. SPEC CPU95 measures the performance of CPU, memory system, and compiler code generation by running 18 programs that are well designed to gather their throughput. The geometric mean is used to represent system overall performance compared to a reference machine, Sun-sparc-10/40MHz. This standardized set of benchmarks (SPECint95 and SPECfp95) is adaptable to the recent generation of high-performance computing efficiently (HPC) [7]. Hence, the following formula (1) is used to *unify* CPU resource, $CUP_{uni}(a_{ij}$, tl, $H_t)$, onto target host, H_t from $CUP_{obs}(a_{ij}$, tl, $H_k)$ on source host, H_k.

$CUP_{uni}(a_{ij}$, tl, $H_t)$ = $CUP_{obs}(a_{ij}$, tl, $H_k)$ * SPEC_RATE(H_k) / SPEC_RATE(H_t) ---- (1)
Where for all j, SPEC_RATE(H_j) = AVG(SPECint95(H_j) /Max(SPECint95(H_j), SPECfp95(H_j)/Max(SPECfp95(H_j))).

Another piece of static system information, CCR(H_j) is considered but it is inapplicable to unification of resources, because a different number of CPU cycles between RISC and CISC are used, and because different VLSI technology is used, for example, Sun Ultra1-167MHz has better performance than SPARC5-170MHz.

Now, the feasibility analysis steps are illustrated as follows. The best-host approach(see [8]) without consideration of resource availability does not guarantee load balance. Therefore, this step distinguishes feasible hosts in terms of resource availability based on the unified resource. Furthermore, in formula (2), the *thresholds* for the load balancing process include CPU idle time and available memory; the current CPU and memory usage of the process that is to be migrated are compared against the thresholds to determine the destination host. If a host satisfies the condition of feasibility analysis in formula (2) and no faults are detected on the host, then it is a candidate host.

$\text{FeasibleCPU}(H_i,t) = \text{CIP}(H_i,t) - \text{CUP}_{uni}(a_{ij}, tl, H_i) > \text{Threshold\_CPU}(H_i)\&$
$\text{FeasibleMEM}(H_i,t) = \text{FAM}(H_i,t) - \text{MEM}_{obs}(a_{ij}, tl, Hi) > \text{Threshold\_MEM}(H_i)$ ------------(2)

Finally, the optimization step works as follows. Optimized resources give a good information RM for efficient allocation and reservation.

Definition: *Laxity* is an available amount of unified resources after allocation of requested resources delivered from QM for the violated applications.

FeasibleCPU(H_i,t) is the available amount of resources after allocation of aij. Unifying the FeasibleCPU(H_i,t) gives the optimized resource availability. This optimization is an important QoS factor. Formula (3) and (4) show the Laxity of CPU, $\mathcal{L}\text{CPU}(H_i,t)$, and Laxity of memory, $\mathcal{L}\text{MEM}(H_i,t)$.

$\mathcal{L}\text{CPU}(Hi,t) = \text{FeasibleCPU}(H_i,t)*\text{SPEC\_RATE}(H_i)$
----------------------------------- (3)

$\mathcal{L}\text{MEM}(H_i,t) = \text{FeasibleMEM}(H_i,t)$ ----------- (4)

Based on optimized resources, the resource allocation schemes, max-laxity host(*MLH*) shown in formula (5), and min-laxity host(*mlh*) shown in formula (6) are carefully considered. Other approaches such as random(*ra*), round-robin(*rr*), and least-load host(*llh*) have been tested and compared with our allocation schemes of resource optimization. But the least load approach (resources are not unified) shown in formula (7) does not guarantee QoS requirement as the available resources in the supply space do not correspond to resource requirement in demand space.

$MLH = \text{Max}_i(\mathcal{L}\text{CPU}(H_i,t))$ and $\text{Top}_i(\mathcal{L}\text{MEM}(H_i,t), 50)$
----------------------------------- (5)

$mlh = \text{min}_i(\mathcal{L}\text{CPU}(H_i,t))$ and $\text{Bot}_i(\mathcal{L}\text{MEM}(H_i,t),50)$
----------------------------------- (6)

$llh = \text{Max}_i(\text{CPI}(H_i,t)*W_{cpu} + \text{FAM}(H_i,t)* W_{mem})$
----------------------------------- (7)

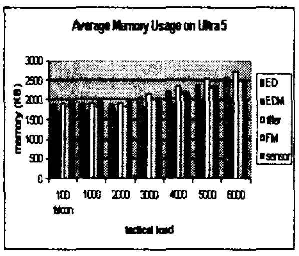
where $\text{Top}_i(\mathcal{L}\text{MEM}(H_i,t), 50)$: the host "i" is in top 50th percentile in laxity of memory, $\text{Bot}_i(\mathcal{L}\text{MEM}(H_i,t), 50)$: the host "i" is in bottom 50th percentile in laxity of memory, and $W_{cpu}+W_{mem} = 1$, for all i.

In our approach, the other resource requirements like network bandwidth, I/O disk are applicable in a similar way. The final decision is made based on the laxity of each resource using heuristic algorithm: find a host that has maximum CPU; if the host is in top 50th percentile of the host list (sorted by $\text{MEM}(H_i,t)$); select the host; if not, examine the next host that has maximum $\text{CPU}(H_i,t)$. Instead of resource allocation, control of heterogeneous resources is an efficient way to provide quick resource management. Dynamic CPU proportion change on Linux using the Quasar scheduler [9][10] and priority handling on NT and Solaris are implemented in our scheme.

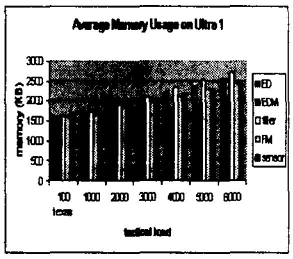
Furthermore, for accurate allocation, the RM should consider not only load balance based on resource availability, but also a measure of system contention called *slowdown* factor.

3. Experiment

We have used DynBench[11] as an assessment tool for DeSiDeRaTa. It uses an identical scenario for experiments. The experimental system parameters and heterogeneous environment are as follows: 1 Linux Pentium 200mhz, 1 NT Pentium-III 500mhz, 2 NT Pentium-II 400MHz, 2 NT Pentium 200MHz, 2 Solaris Sparc-5 170MHz, 2 Solaris Ultra-1 167MHz, 1 SunOS on ULTRA10 300MHz, and 100Mhz Fast Ethernet.



(a) MEM(a_k, t), Sun-Ultra-5



(b) MEM(a_k, t), Sun-Ultra-1

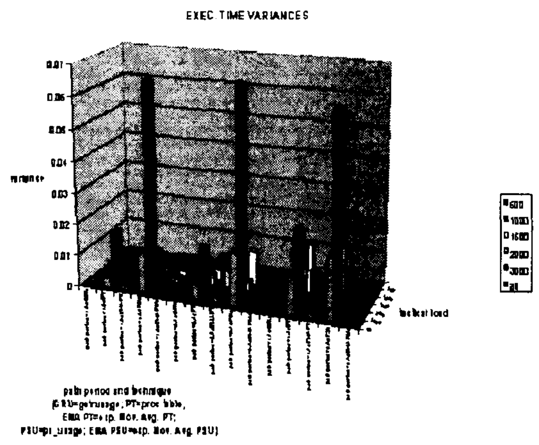


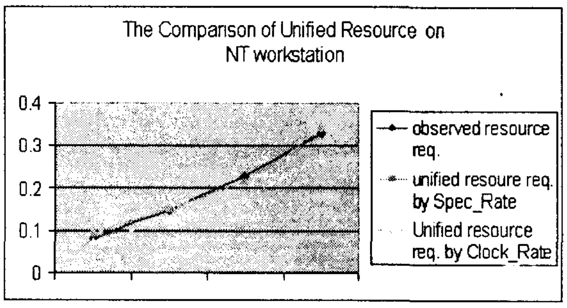
Figure 3. The Dynamic measures of monitored resource requirement

First experiment monitors and analyzes resource requirements corresponding to step 1 in section 2. Second experiment measures the unification approaches requirements corresponding to step 3 in section 2. Third experiment compares different allocation schemes corresponding to step 5 in section 2. Detail experiments are presented in the remainder of this section.

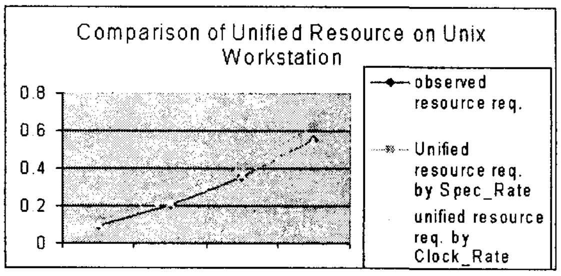
Experiment 1 shown in Figure 3 describes the measures of variance of execution time with different methods and different periods in (c), and variance of memory among hosts in (a) and (b). The three different monitoring techniques, *getrusage()* (*GRU*) system call, reading *process table(PT)*, and *ps(PSU)* command, are used. From the experiments (c), the variance of execution time measured by reading *PT* is high, and is dependent on the monitor cycle time as the period for accessing PT cannot exactly cover the range of process execution time. It is impossible to collect exact resource usage of a process at a particular instant of time. However, the GRU system call shows accurate process resource usage in terms of variance of execution time. The exponential moving average (EMA) of each method is used for filtering. The maximum difference of memory usage by the evaluate and decide application(ED) on two different hosts is 48Kbytes (from (a) and (b) in Figure 3). Hence, Threshold_MEM(H_k,t) and Threshold _CPU(Hk,t) are necessary components in finding candidate host. The variances of memory requirement of applications are measured by zero.

The second experiment described in Figure 4 shows the difference between observed resource usage and unified resource estimated by SPEC_RATE and Clock_Rate(CCR). For example, the execution time is collected on Pentium-200MHz, and we multiply the measured execution time and SPEC_RATE

/CCR of the target host, PentiumIII-500MHz. Next we experiment with the same scenario on PentiumIII-500MHz to observe actual execution time of the process to compare with previous estimated execution time experimented on NT in Figure 4 (a). The difference between unified resource by CCR and the observed resource is 8% on NT, and 3.5% on Sun. The difference of unified resource by SPEC_RATE has 1% on NT, and 11% on Sun.



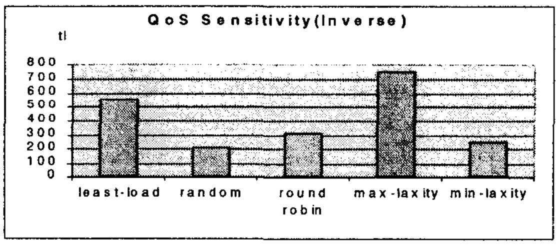
(a) Pentium (200MMX) vs Pentium-III (500Mhz)

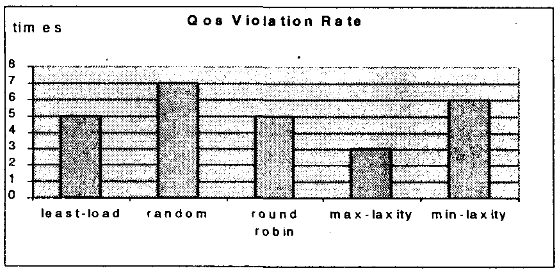


(b) Ultra-10(300Mhz) vs Ultra-1 (140Mhz)

Figure 4. Resource Unification by SPEC_RATE and Clock_Rate

Experiment 3 proposed three measurements - QoS violation rate (QVR), QoS Sensitivity (QSS), and QoS (to compare QoS characteristics by different allocation decision algorithm as shown in Figure 5). The QVR is the number of violations within 2 minutes by increasing workload. QSS is the amount of workload to trigger second violation after the first violation. QoS is the latency of a path improved by first allocation. This experiment shows clearly that the approach that ignores heterogeneity (proposed by Ravindran[8]) is much worse than our scheme in terms of QVR, QSS, and QoS. Our Max scheme improves 26.4% better in QoS, 36.4% better in QSS, and 60% less in QVR than his approach. Our scheme for various heterogeneous hosts that use SPEC_RATE outperforms in efficient manner than the clock rate approach.





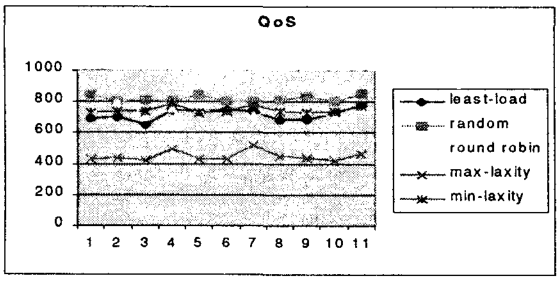


Figure 5. Comparison of Resource Management Schemes

4. ConclusionandFutureStudy

This paper presents 5 solutions of resource allocation for dynamic real-time systems. Our "Max" scheme works not only 257% better than *ra*, 142% than *rr*, 36.4% better than , and 200% better than the "min" scheme in *QSS*, but also 38.6% better than *ra*, 28.5% better than *rr*, 31.6% better than , and 20.9% better than "min" scheme in *QoS*. Controlling heterogeneous resources using CPU proportion change and priority change is useful for the server programs. The efficiency of resource allocation in terms of QoS objectives for scalable and moveable clients is better than that of the control. Also, heterogeneous network resource monitoring and allocation, and the decision mechanism between allocation and control is an important issue in providing QoS requirements.

Acknowledgement

This work is partially supported by Seoul Women's University special research fund.

References

[1] L. R. Welch, B. Ravindran, R. Harrison, L. Madden, M. Masters and W. Mills, "Challenges in Engineering Distributed Shipboard Control Systems," *The IEEE Real-Time Systems Symposium*, December 1996.

[2] L.R Welch, B. Ravindran, B.Shirazi and C. Bruggeman, "Specification and Analysis of Dynamic, Distributed Real-Time Systems", *in Proceedings of the 19th IEEE Real-Time Systems Symposium*, 72-81, IEEE Computer Society Press, 1998.

[3] J. Stankovic, and K. Ramamritham, "Advances in Real-Time Systems," IEEE Computer Society Press, April 1992.

[4] S. Son, "Advances in Real-Time Systems," Prentice Hall, 1995.

[5] L.R. Welch, P V. Werme, B. Ravindran, L. A. Fontenot, M.W.Masters, D. W. Mills and B. Shirazi, "Adaptive QoS and Resource Management Using Analytical Modeling of Electrostatic Structures", Proc. IEEE Micro Electro Mechanical Systems Workshop, pp. 120-127, 1990.

[6] Gary Koob, "Quorum," Proceedings of the DARPA ITO General PI Meeting, pages A-59 to A-87, October 1996.

[7] SPECCPU95 Benchmark http://www.spec.org

[8] Binoy Ravindran, "Modeling and Analysis of Complex, Dynamic Distributed Real Time System", Thesis, Computer science and engineering, The University of Texas at Arlington, 1998.

[9] Ashvin Goel, David Steere, Calton Pu, Jonathan Walpole, "Adaptive Resource Management via Modular Feedback Control", http://www.cse.ogi.edu/DISC/projects /quasar/ publications.html

[10] David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole, "A Feedback-driven Proportion Allocator for Real-Rate Scheduling", Operating Systems Design and Implementation (OSDI), Feb 1999.

[11] B. Shirazi, L. R. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, and E. Huh, "Dynbench: A Dynamic Benchmark Suite for Distributed Real-Time Systems," Lecture Notes in Computer Science, Volume 1586, p1335-1349, Springer-Verlag, 1999.

주 작 성 자 : 허 의 남
논문투고일 : 2003. 10. 21
논문심사일 : 2003. 11. 26(1차), 2003. 12. 01(2차), 2003. 12. 24(3차)
심사판정일 : 2003. 12. 24

● 저자소개 ●

허의남
1990 부산대학교 전산통계 학사
1995 텍사스대학교 전산학과 석사
2002 오하이오대학교 전산학과 박사
2002~2003 삼육대학교 컴퓨터과학과 조교수
2003~현재 서울여자대학교 정보통신공학부 조교수
관심분야: 네트워킹, 그리드, 실시간 미들웨어, 유비쿼투스 컴퓨팅

이웅재
1981년 연세대학교 전산학과 졸업(학사)
1985년 Univ. of Illinois at Chicago전산학과 졸업(석사)
1991년 Illinois Institute of Technology 졸업(박사)
1993년~현재 : 서울여재대학 정보통신공학부 교수
관심분야 : 인공지능, 자연어처리, 멀티미디어 etc.

이종숙
1989년 충남대학교 계산통계학과 졸업(이학사)
1991년 충남대학교 대학원 전산학과 졸업(석사)
2001년 University of Canterbury 컴퓨터 과학과 전산학 (박사)
관심분야 : 인공지능, 자연어처리, 멀티미디어 etc.