Accommodating QoS Prediction in an Adaptive Resource Management Framework

E. Huh[1], L. R. Welch[1], B. A. Shirazi[2], B. Tjaden[1], and C. D. Cavanaugh[2]

[1] 339 Stocker Center, School of Electrical Engineering and Computer Science,
Ohio University, Athens, OH 45701
[2] Department of Computer Science Engineering,
The University of Texas at Arlington, Arlington, TX 76019
[1]{ehuh|welch|tjden@ace.cs.ohiou.edu}
[2]{shirzai|cavan@cse.uta.edu}

Abstract. *Resource management for dynamic, distributed real-time systems requires handling of unknown arrival rates for data and events; additional desiderata include: accommodation of heterogeneous resources, high resource utilization, and guarantees of real-time quality-of-service (QoS). This paper describes the techniques employed by a resource manager that addresses these issues. The specific contributions of this paper are: QoS monitoring and resource usage profiling; prediction of real-time QoS (via interpolation and extrapolation of execution times) for heterogeneous resource platforms and dynamic real-time environments; and resource contention analysis.*

1 Introduction

In [1], real-time systems are categorized into three classes: (1) deterministic systems, which have a priori known worst case arrival rates for events and data, and are accommodated by the Rate Monotonic Analysis (RMA) approach (see [2]); (2) stochastic systems, which have probabilistic arrival rates for events and data, and can be handled using statistical RMA [3] and real-time queuing theory [4]; and (3) dynamic systems, which operate in highly variable environments and therefore have arrival rates that cannot be known a priori.

This paper presents a resource management approach for dynamic allocation to handle execution times represented using a time-variant stochastic model. Additionally, we show how to accommodate heterogeneity of resources and QoS prediction.

Section 2 provides an overview of the resource manager (RM) approach. Sections 3-5 explain each component used in our RM approach. Section 6 presents experimental assessments of our techniques.

2 Overview of RM approach

Our approach to resource management is based on the dynamic path model of the *demand space* [5], [8], [9]. This demand space model is a collection of dynamic real-time paths, each of which consists of a set of communicating programs with end-to-end QoS requirements. The demand space system model is described in Table 1.

Table 1. Demand space system model

Symbol	Description
P_i	a name of path "i"
a_{ij}	name of application j in path "i"
H_k	a name of host "k"
$\lvert P_i.DS \rvert = tl$	data stream sizes of path "i" (or workload or tactical load)
$T(a_{ij},tl)$	period of a_j in P_i with workload tl
$C_{obs}(a_{ij}, tl, H_k)$	observed execution time of a_j at cycle c with tl in path "i" on H_k
$C_{req}(a_{ij}, tl, H_k)$	required execution time of a_j at cycle c with tl in path "i" on H_k
$C_{prof}(a_{ij}, tl, H_p)$	profiled execution time of a_j at cycle c with tl in path "i" on H_p
$C_{pred}(a_{ij}, tl, H_k)$	predicted execution time of a_j at cycle c with tl in path "i" on H_k
$D_{obs}(a_{ij}, tl, H_k)$	observed queuing delay of a_j at cycle c with tl in path "i" on H_k
$CUP_{obs\backslash}(a_{ij}, tl, H_k)$	observed CPU usage on H_k for the a_j in P_i with tl
$CUP_{req}(a_{ij}, tl, H_k)$	required minimum CPU usage on H_k for a_j in P_i with tl
$CUP_{ureq}(a_{ij}, tl, H_k)$	required, unified minimum CPU usage on the target H_k for the a_j in P_i with tl
$MEM_{req}(a_{ij}, tl, H_k)$	memory usage of a_j in path "i" on H_k with tl
$\lambda_{req}(P_i)$	required latency of P_i (=QoS)
$\lambda_{pred}(c+1, P_i)$	predicted latency of path P_i at cycle c+1
$\psi(P_i)$	required slack interval for each QoS requirement = $[\psi_{min}(P_i), \psi_{max}(P_i)]$

Table 2. Supply space system model

Symbol	Description
H_k	host name "k"
$SPECint95(H_i)$	the fixed point operation performance of SPEC CPU95 of H_i
$SPECfp95(H_i)$	the floating point operation performance of SPEC CPU95 of H_i
$SPEC\_RATE(H_i)$	the relative host rating of H_i
$Threshold\_CPU(H_i)$	the CPU utilization threshold of H_i
$Threshold\_MEM(H_i)$	the memory utilization threshold of H_i
$CUP(H_i,t)$	the CPU usage (user + kernel) percentage of H_i at time t
$CIP(H_i,t)$	the idle-percentage of H_i at time t
$FAM(H_i,t)$	the free-available-memory of H_i at time t
$MF(H_i,t)$	the number of page faults on H_i at time t
$INT(H_{i,} t)$	the number of interrupts on H_i at time t
$CALL(H_{i,} t)$	the number of system calls on H_i at time t
$CTX(H_{i,} t)$	the process context switching rate on H_i at time t
$CMI(H_{i,} t)$	the number of packet-in received on H_i at time t
$CMO(H_{i,} t)$	the number of packet-out transferred on H_i at time t
$COL(H_{i,} t)$	the number of collisions occurred on H_i at time t
$LM_i(H_{j,} t)$	the i^{th} load metrics in host j at time; $LM_i(H_{i.} t) \in \{FAM(H_i,t), MF(H_i,t), INT(H_{i.} t), CIP(H_i t), CUP(H_i,t), CALL(H_i t), CTX(H_i t)\}$

We also model the resources or the *supply space* (described in Table 2), which consists of host features, host resources, and host load metrics.

The resource management problem is to map the set of all paths P_i onto the set of hardware resources, such that all $\lambda_{req}(P_i)$ are satisfied. Since the workloads of the P_i

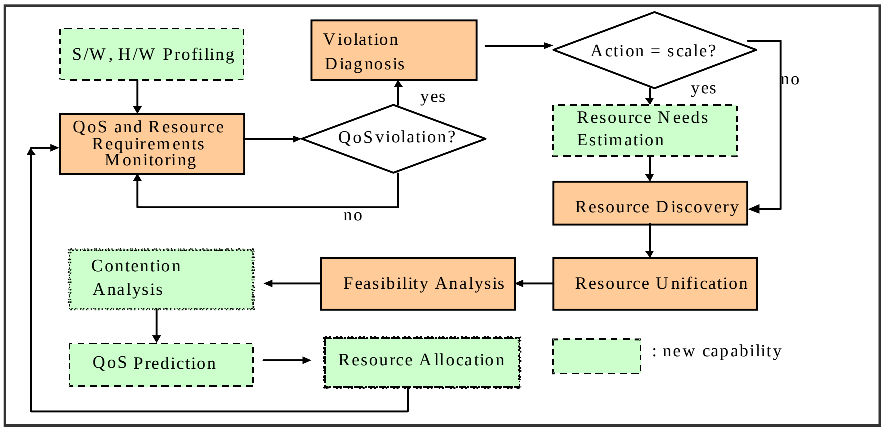

Fig. 1. Overview of resource manager

vary, the mapping needs to be adapted dynamically. The flow of our adaptive resource management approach shown in Fig. 1.
Each step is described in detail in the subsequent sections of this document.

3 Software and Hardware Profiling

In order to manage resources in an efficient manner, it is necessary to understand the resource usage characteristics of the members of the demand space and the relative resource capabilities of the members of the supply space.

S/W profiling measures an application's execution time, period, CPU usage, and memory usage that are collected passively by an external process (a monitor) that reads the *proc* table periodically to obtain process data. Three different techniques are tested as follows: (1) the process calls *getrusage* once per period, (2) an external monitor reads *ps_info* block in the *proc* table once per second, and (3) an external monitor reads *ps_usage* block in the *proc* table once per second.

An exponential moving average is applied to measurements for all techniques for filtering. Initial profiling is done during application development and profiles are refined through dynamic profiling. The accuracy of exponential moving average of *ps_usage* block in the *proc* table is almost as good as *getrusage* shown in Fig. 2.

H/W profiling measures capabilities of hosts relative to a reference host using the Standard Performance Evaluation Corporation (*SPEC*). *SPEC* is a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers (see [10]). To achieve overall, relative system performance, the mean throughput is compared to a reference machine, a Sun-Sparc-10/40Mhz.

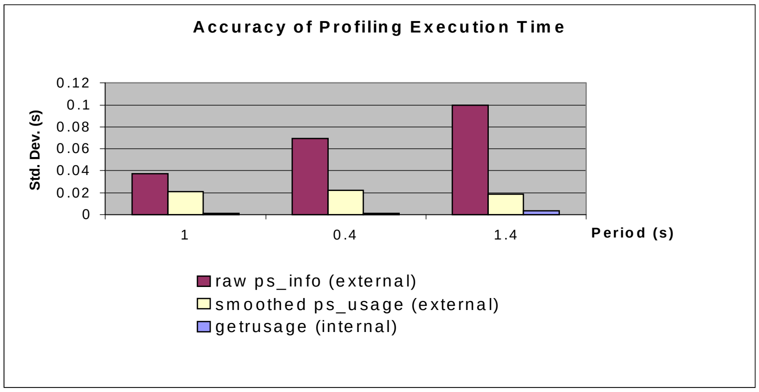

Fig. 2. Comparison of profiling techniques

We use *SPECfp95* (a measure of a host's floating point performance) and *SPECint95* (a measure of a host's fixed point performance) to derive the relative, *normalized* host rating as follows:

$$SPEC\_RATE(H_i)=AVG(SPECint95(H_i)/Max(SPECint95(H_j)),SPECfp95(H_i)/Max(SPECfp95(H_j))),\ \text{where}\ \forall_j.$$

4 QoS and Resource Utilization Monitoring

This section discusses our approach to QoS and resource monitoring, resource needs estimation, and resource discovery.

This module observes end-to-end real-time QoS of dynamic, distributed paths, and monitors resource requirements for dynamic software profiling to determine execution time, period, and memory usage. The memory usage (of main memory for allocation of workloads) is observed by taking the process residence set size from the *proc* table. The execution time of an application consists of the user- and the kernel-time, each of which corresponds to accurate computation of CPU utilization measured for a "move" action as follows:

$$CUP_{req}(a_{ij},tl, H_k)= C_{obs}(a_{ij}, tl, H_k) / T(a_{ij},tl).$$

Also, the *cycle time of the QoS monitor* called validity interval is used for the period ($T(a_{ij},tl)$) of an application to calculate the CPU resource requirement, while conventional approaches use the arrival time of workload for the period, which causes poor utilization on the dynamic environment.

Interpolation and extrapolation uses profiles to ***estimate resource needs*** of a new replica of a scalable application. When the current path QoS is greater than minimum slack of the QoS requirement, and QoS Manager (QM) recommends a "scale up" action decision based on the rate of workload trend, the resource requirements for the new workload tl (*tl = current tl / (current replicas+1)*) that will be distributed equally among replicas, need to be modified at run-time to request resource needs to the supply space. Hence, initial profiles of the violated application are the only way to decide required $C_{req}(a_{ij}, tl, H_k)$ and $MEM_{req}(a_{ij}, tl, H_k)$ for the various workloads as the boundary of execution time of an application is not obtainable on dynamic environments. The interpolation and extrapolation of resource needs for a “scale down” action proceeds exactly the same as a "scale up” action except for the calculation of workload tl (*tl = current tl / (replicas – 1)*).

The examined average of errors between the observed execution times and the estimated execution times that are examined by the piecewise linear regression using 2 data points is 12.1 milliseconds (1% CPU usage).

Resource discovery determines current utilization levels for communication and computation resources by using *vmstat* and *netstat* system calls once per second. These metrics are filtered by exponential moving average.

Communication resource management over the broadcasting type of networks (Ethernet/Fast Ethernet) is a hard problem as contention of those types of networks depends on the number of communication nodes, the size of packets, retransmission strategies, and collisions. The network load in terms of delay clearly has a strong relationship with collisions. Hence, our approach considers network load of hosts that are part of a real-time path is computed using the number of packet received / transmitted, and collisions. For a single host, the network load (net_load_of_host) is computed as follows:

$$\mathit{net\_load\_of\_host} = (1+ COL(H_{i,} t))*(CMI(H_{i,} t) + CMO(H_{i,} t))$$

5 Resource Selection

This section explains techniques for resource unification (mapping heterogeneous resource requirements into a canonical form), feasibility analysis, contention analysis, QoS prediction, and resource allocation analysis and enactment.

The role of ***resource unification*** is to map heterogeneous resource requirements into a canonical form of each resource metric. To allocate delivered $CUP_{req}(a_{ij}, tl, H_k)$, RM needs to determine the relative amount of the resources available on the target host. There might exist two approaches, which are static, and dynamic for resource management considering heterogeneity of resources. The static approach uses stable system information like benchmarks and CPU clock rate. In Globus project (see [6]), benchmark rates are used as the resource requirement (e.g. 100Gflops). The dynamic approach uses low level system parameters (see [7]). In Windows NT, a popular operating system, it is very complicated to access the dynamic system parameters that the operating system provides. Eventually, as a host-level, the global scheduler that will handle any types of systems, RM needs to use general system characteristics instead of dynamic, specific system parameters in the operating system layer. In our approach, using a static system information, *SPEC_RATE* shown in section 3, resources are unified into a canonical form as follows:

$$CUP_{ureq}(a_{ij}, tl, H_t) = C_{pred}(a_{ij}, tl, H_t) \;/\; T(tl, a_{ij})$$

$$C_{pred}(a_{ij}, tl, H_t) = C_{req}(a_{ij}, tl, H_p) * SPEC\_RATE(H_k) / SPEC\_RATE(H_t),$$

where H_t: a target host, H_k: host that the resource requirements are measured.

Feasibility analysis finds resources, which will meet $CUP_{ureq}(a_{ij}, tl, H_t)$. The *thresholds* are used for adaptable resource supply to tolerate the difference between unified and actual resources. For example, if the available CPU resource is greater than the unified CPU resource requirement plus the threshold, then the host becomes a candidate.

Contention analysis phase predicts queuing delays of applications among candidates. The queuing delay of an application in a path based real-time system is one of the critical elements for the RM to examine schedulability, when periods (of applications or paths) overlap each other. Currently, in our approach, observed system load metrics of hosts ($LM_i(H_t,t)$) are applied to get the delay on heterogeneous hosts.

First, we predict the queuing delay of the application in the target host by the observed queuing delay multiplied by the ratio of monitored load metrics between current host and target host ($D_{pred}(a_{ij},tl,H_t) = D_{obs}(a_{ij}, tl, H_k) * LM_i(H_t,t) / LM_i(H_k,t)$). Second, we use the execution time and current CPU usage on the target host ($D_{pred}(a_{ij},tl,H_t) = C_{pred}(a_{ij}, tl, H_t) * CUP(H_t,t)$). If one of our approaches can approximately uncover observed queuing delay, it becomes a generic solution in point of the host-level, global RM. An experiment is assessed for these approaches in section 6.

RM predicts a real-time QoS (considering contention) that will result from candidate reallocation actions. In general, when a customer requests QoS, this step tells the next QoS ($\lambda_{pred}(c+1,P_i)$) to the customer in addition to resource supply. If a single application in a path is violated, the path QoS is easily computed by adding predicted latency of the application instead of the current latency. Otherwise, the path QoS is accumulated until the last application's latency is predicted.

Resource allocation selects and performs a reallocation action based on the predicted QoS. Using the predicted QoS, RM can guarantee new reallocation. By testing predicted QoS path latency $\lambda_{pred}(c+1,P_i)$) by $\psi_{max}(P_i) < \lambda_{pred}(c+1,P_i) < \psi_{min}(P_i)$ called *pre-violation test*, QoS violation of the path at the next cycle can be detected by RM. Therefore, RM now can see QoS in addition to an amount of resources being supplied.

An allocation schemes for the violated application are considered based on QoS slack called "QoS Allocation (QA)", where QoS slack = $\lambda_{req}(P_i)$ **-** $\lambda_{pred}(c+1, P_i)$; which has passed the above *pre-violation test***.** A *greedy, heuristic QA* scheme finds a host H_i which has minimum $\lambda_{pred}(c+1,P_i)$; and it is in top 50th percentile of the average network load among all candidate hosts.

6 Experiments

We have used DynBench (see [8]) and D-SPEC (see [9]) as an assessment tool and specification language for dynamic resource management. DynBench uses an identical scenario for experiments, respectively. A CPU load generator has been developed to allow the user to adjust CPU usage. The profiled execution times are measured on a sun-ultra-1 (140Mhz). Prediction is performed on sun-ultra-10 (333Mhz) as a source node and sun-ultra-10 (300Mhz) as a target node, respectively.

Experiment 1 shows predicted latency of a filter application a target host. Initially, 30% of CPU usage is used. In Fig. 3, using general system load metrics, several

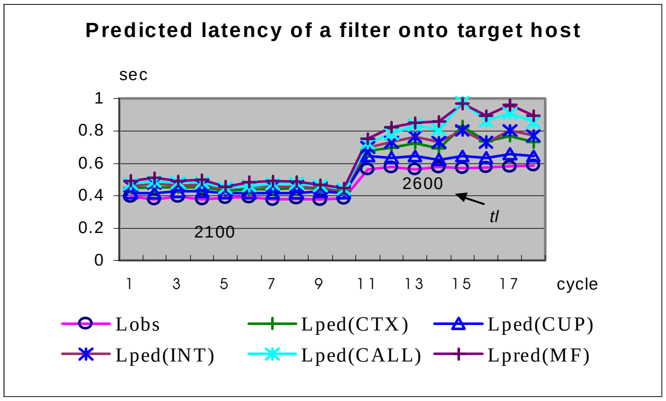

Fig. 3. Predicting latency of an application

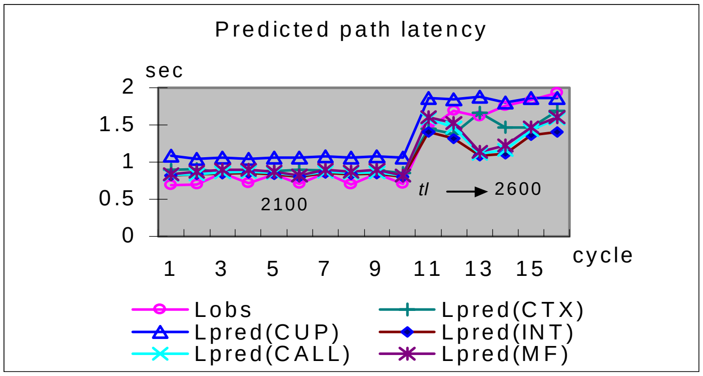

Fig. 4. Path prediction

methods of predicting latency are tested, and they are compared to the observed latency, which is measured at offline on the host with the same scenario

Each method specified in parenthesis predicts latency of the filter application as follows:

- Lobs : observed application latency
- $\mathrm{Lpred(CTX)} = C_{pred}(a_{ij},tl,H_t)+D_{obs}(a_{ij},tl,H_k) * \mathrm{CTX}(H_t,t)/\mathrm{CTX}(H_k,t)$
- $\mathrm{Lpred(CUP)} = C_{pred}(a_{ij},tl,H_t)+C_{pred}(a_{ij},tl,H_t) * \mathrm{CUP}(H_t,t)$
- $\mathrm{Lpred(INT)} = C_{pred}(a_{ij},tl,H_t) + D_{obs}(a_{ij},tl,H_k) * \mathrm{INT}(H_t,t)/\mathrm{INT}(H_k,t)$
- $\mathrm{Lpred(CALL)} = C_{pred}(a_{ij},tl,H_t)+D_{obs}(a_{ij},tl,H_k) * \mathrm{CALL}(H_t,t)/\mathrm{CALL}(H_k,t)$
- $\mathrm{Lpred(MF)} = C_{pred}(a_{ij},tl,H_t) + D_{obs}(a_{ij},tl,H_k) * \mathrm{MF}(H_t,t)/\mathrm{MF}(H_k,t)$

The results of experiment show that the queuing delay is a more important issue than the execution time to predict accurate latency of an application, when a host is overloaded. Overall average error (Lobs() - Lpred(CUP)) is *0.031 seconds.*

Experiment 2 shows the path latency comparison between the predicted (Lpred) and the observed (Lobs) latency. Note that predicted path latency is the sum of each application's predicted latency on to the target host. Fig. 4 explains that the Lpred(CUP) approach is most accurate approach, when the system is overloaded (CUP(H_k,t)) > 70 at workload 2600). To fully utilize CPU resource, the measurement of the queuing delay, when CPU usage is high, is very important for the distributed real-time system. The average error between the predicted and the observed latency is *0.084 seconds.*

7 Conclusions and Ongoing Work

The experimental results show that our approach achieves good CPU utilization by analyzing system contention and by predicting QoS accurately. The accuracy of the techniques is shown by noting that the predicted CPU resource needs differ from observed ones by no more than 4.5%. Ongoing work includes proactive RM and dynamic QoS negotiation.

References

1. Welch, L.R., Masters, M.W.,: Toward a Taxonomy for Real-Time Mission-Critical Systems. Proceedings of the First International Workshop on Real-Time Mission-Critical Systems 1999)
2. Liu, C.L., Layland, H.W.,: Scheduling Algorithm for multiprogramming in hard real-time environment. JACM, Vol. 20. (1973) 46-61
3. Atlas, A., Bestavros, A.,: Statistical Rate Monotonic Scheduling. Proceedings of Real-Time Systems Symposium (1998)
4. Lehoczky, J.P.: Real-Time Queueing Theory. Proceedings of IEEE Real-Time Systems Symposium, IEEE CS Press (1996) 186-195
5. Welch, L.R., Ravindran, B., Harrison, R., Madden, L., Masters, M., Mills, W.,: Challenges in Engineering Distributed Shipboard Control Systems. The IEEE Real-Time Systems Symposium. (1996)
6. Czajkowski, K., Foster, I., Kesselman, C., Martin, S., Smith, W., Tuecke, S.,: A Resource Management Architecture for Metacomputing Systems. Proceedings in IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing (1998)
7. Chatterjee, S., Strosnider, J.,: Distributed Pipeline Scheduling: A Framework for Distributed, Heterogeneous Real-Time System Design. In the Computer Journal, British Computer Society, Vol. 38. No. 4. (1995)
8. Welch, L.R., Shirazi, B.A.,: A Dynamic Real-Time Benchmark for Assessment of QoS and Resource Management Technology. IEEE Real-time Application System (1999)
9. Welch, L.R., Ravindran,B., Shirazi, B.A., Bruggeman,C., : Specification and Analysis of Dynamic, Distributed Real-Time Systems. Proceedings of the 19th IEEE Real-Time Systems Symposium, IEEE Computer Society Press (1998) 72-81
10. OSG Group: SPEC CPU95. http://www.spec.org.