Performance Analysis for Real-Time Grid Systems on COTS Operating Systems

Eui-Nam Huh[1] and Y. Mun[2]

[1] Seoul Women's University, School of Computer Science, Seoul, Korea
huh@swu.ac.kr
[2] Soongsil University, School of Computer Science, Seoul, Korea
mun@computing.ssu.ac.kr

Abstract. Computer technology for communication has become an integral aspect of daily operation. The exponential growth of internet services with dynamic inquiries in such areas as manufacturing, business, air traffic control and mission critical systems demands that there be quick, reliable and safe use of services. Each service must contain QoS metrics to assure security, performance and accuracy. A new paradigm of resource management middleware techniques is in this paper presented which can provide QoS for dynamic, distributed real-time systems on Common Off The Shelf (COTS) operating systems. Accommodation of dynamic environments enables the middleware to carefully consider an efficient design of resource profiling, resource needs estimation, resource unification, and performance analysis (or compliant with schedulability analysis) infrastructure providing significant benefits for QoS management on COTS operating systems. First, the use of low-cost COTS systems is extended to real-time computing without changing the operating system. Further, experiments for response time analysis confirm that the worst-case analysis poorly utilizes computational resources. Finally, it is shown that the new method of middleware design employing scalability of software and hardware system can be easily applied to legacy systems to manage resources efficiently for quick, reliable services and accurate QoS.

1 Introduction

Computer Technology for communication has become an integral aspect of daily operation. The exponential growth of internet services with dynamic inquiries in such areas as manufacturing, business, air traffic control and mission critical systems demands that there be quick, reliable and safe use of services. Each service must contain QoS metrics to assure security, performance and accuracy.

Generally, the real-time system is designed to predict response times of applications before they are executed in order to meet the QoS requirement in the metric of timeliness. The Rate Monotonic Analysis (RMA) introduced by [1] is used primarily to determine schedulability of an application by using *a priori* Worst-Case Execution Time (WCET) and the priority of the application. The priority of the application to be applied to RMA is dependent upon arrival patterns and rates. The

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2660, pp. 482–490, 2003.

application, which has a higher arrival rate of task or needs to be executed more frequently, has a higher priority level than any other applications.

However, as has been noted in [2], and [3] resources are poorly utilized if the average case is significantly less than the worst case. Another drawback of RMA is that it cannot *efficiently* accommodate high-priority jobs that have relatively low rates. It must, however, be noted that RMA can be made to work in such cases, by transforming low-rate, high-priority jobs into high-rate jobs – but this can be extremely wasteful in terms of resources.

It is stated in [4], and [3] that accurately measuring the WCET is often difficult, and is sometimes impossible. Puschner and Burns in [5] consider WCET analysis to be hardware dependent, making it an expensive operation on distributed heteogeneous clusters.

Statistical RMA in [6] considers tasks that have variable execution times and allocate resources to handle the expected case. The benefit of the approach is the efficient utilization of resources. However, there are shortcomings. First, applications which have a wide variance in resource requirements cannot be characterized accurately by a time-invariant statistical distribution; and secondly, deadline violations occur when the expected case is less than the actual case. Similarly, real-time queuing theory [7] uses probabilistic event and data arrival rates for performing resource a1location analysis. On the average, it provides good utilization of resources. It must be noted, however, that applications which have a wide variance of resource requirements cannot be characterized accurately by a time-invariant statistical distribution

Use of distributed systems appears to provide strong integration for quick services. However, distributed systems have heterogeneous resources. That is, they have different capacities. Thus, when data or tasks are distributed to heterogeneous systems to be processed concurrently, the load of data should be balanced. A major difference between the traditional load-balancing techniques demonstrated by [8] and the dynamic QoS-based resource management services lies in the overall goals. While load-balancing systems[1] attempt to achieve *system* performance goals such as minimized response time or maximized throughput, the dynamic QoS-based resource management service strives to meet the QoS requirements of each application being managed. Another major difference between these systems is their workload models. Traditional load balancing systems assume *independent* jobs with *known resource requirements*. In dynamic resource management systems, the workload requirements of applications are *dependent* (communicate with each other) and can vary, based on environmental conditions.

Presently, there is an increasing amount of research focused on QoS management and resource management techniques using distributed real-time systems. However, most of the Commercial Off The Shelf (COTS) systems use general purpose operating systems such as Windows, Unix, and Linux rather than real-time operating systems such as MACH, RT-Kernel, and Lynx.

Therefore, use of distributed systems with COTS operating systems to handle the QoS and resource management for the dynamic real-time system becomes a new

[1] See Load-Leveler (IBM 1993), Globus (http://www.globus.org), Beowulf (Becker et al 1995), and Condor (http://www.cs.wisc.edu/condor/manual).

challenge to handle complex problems. The COTS operating systems, especially, has become widely used in many application fields (especially in internet based applications) supporting QoS.

2 Contention Analysis for Tasks with Same Priority Level

The benefit of use of COTS operating systems can be extended, if they can be utilized for real-time tasks. With the small number of priority levels in COTS operating systems, tasks may often have the same priority which means they will use CPU time quanta (denoted TQ) equivalently.

Furthermore, dynamic environments with the unknown arrival times of processes make it difficult to predict the exact response time of the processes (or tasks). Thus, in the following section, the techniques which are the *worst-case* analysis and probabilistic analysis to predict response times of processes are studied.

2.1 Worst-Case

The *worst-case* of contention occurs when all tasks arrive at the same time and the target task starts last. The *worst-case* queuing delay of a task due to the same priority tasks can be computed. This approach considers the period of the tasks and the segments of execution times of all processes. The term, $MC(a_{xyz}, a_{ijk}, tl, H_q)$, is the modified execution time that an application of workload ($tl(a_{xyz})$) will contend with the target task (a_{ijk}) on the host (H_q). The queuing delay of a_{ijk} on H_q due to the same priority tasks, $SD_{pred}(a_{ijk},tl,H_q)$, is the approximated response time, minus execution time, as shown in formula (1).

$$SD_{pred}(a_{ijk},tl,H_q)=ar_i(a_{ijk})-C_{pred}(a_{ijk},tl,H_q),\ if\ ar_i(a_{ijk})=ar_{i+1}(a_{ijk}),\ \forall \mathrm{I} \qquad (1)$$

The i^{th} approximated response time, ar_i, starts with 0 and adds execution times of all same priority tasks within the period. The following steps show the computation process of the approximated response time.

Step 1. Compute the first approximation of response time (ar_0) by adding the time range of other tasks, $MC(a_{xyz},a_{ijk},tl2, H_q)$.

$$ar_0(a_{ijk})= C_{pred}(a_{ijk},tl,H_q) + \sum_x \sum_y \sum_z MC(a_{xyz},a_{ijk},tl2,H_q)$$

$$MC(a_{xyz},a_{ijk},tl2,H_q) = \begin{cases} SEG(a_{ijk},tl,H_q)*TQ(H_q), & if\ SEG(a_{xyz},tl2,H_q) > SEG(a_{ijk},tl,H_q) \\ C_{pred}(a_{xyz},tl2,H_q), & \text{if } SEG(a_{xyz},tl2,Hq) \le SEG(a_{ijk},tl,H_q) \end{cases}$$

$$where\ tl=tl(a_{ijk},\ c+1),\ Host(a_{xyz},\ t)= Host(a_{ijk},\ t),\ p(a_{xyz})=p(a_{ijk}),\ and\ x \neq i,\ \forall x,y,z$$

The conditions of other tasks (a_{xyz}) are as follows:

1) Should have the same priorities of tasks as the target task $(p(a_{ijk})=p(a_{xyz}))$,
2) Should run on the same host $(Host(a_{xyz}, t)= Host(a_{ijk}, t))$,
3) Should be on a different path from the target task $(x \neq i)$.

From condition 1), it can be seen that the execution times of tasks need to be re-computed due to the equivalent chance of execution among other tasks, which have a different required execution time. Because the $MC(a_{xyz},a_{ijk},tl2,H_q)$ depends on the number of segments of the target task, $SEG(a_{ijk},tl,H_q)$ $(=C_{pred}(a_{ijk},tl,H_q))/TQ(p,H_q))$, the segments of other tasks become the same as the segment of the target task, even though original execution times of other tasks may be greater than that of the target. The extra segments of the other tasks do not affect the response time of the target task.

In Step 2, the next approximation is performed to determine additional execution time required. If the periods of the tasks are smaller than the previous approximated response time ($ar_i(a_{ijk})/T(a_{xyz}) \geq 1$), additional executions occur. Otherwise, the next period does not appear.

Step 2. Compute the next approximation of the response time by using the previous approximation.

$$ar_{i+1}(a_{ijk})=C_{pred}(a_{ijk},tl,H_q) + \sum_x \sum_y \sum_z \left\lceil \frac{ar_i(a_{ijk})}{T(a_{xyz})} \right\rceil * MC(a_{xyz},a_{ijk},tl2,H_q)$$

The ceiling function determines if the period of a task is within the previous approximation. If all task periods are smaller than the previous approximation, none of the tasks starts during the approximation. Thus, Step 3 performs is necessary for finding the answer.

Step 3. Determine if the approximation is the answer.

if $(ar_i(a_{ijk}) == ar_{i+1}(a_{ijk}))$, $SD_{pred}(a_{ijk},tl,H_q) = ar_i(a_{ijk}) - C_{pred}(a_{ijk},tl,H_q)$
else increase i and repeat step 2

2.2 Rate of Progress (RP)

Welch et al (1995) introduced two approaches for estimating the contention using progress rate during the least common multiple (LCM) of periods of tasks, and for resource usage. These approaches were modified and applied to the dynamic real-time systems.

2.2.1 The First Rate of Progress Technique (RP-1)

The following formula (2) called ***RP-1,*** shows how it is processed to compute the queuing delay of the target application (a_{ijk}) due to the same priority tasks.

$$SD_{pred}(a_{ijk}, tl, H_q) = \left(\frac{SEG(a_{ijk}, tl, H_q)}{prate\ 1(a_{ijk})} - SEG(a_{ijk}, tl, H_q) \right) \times TQ(p, H_q) \quad (2)$$

In this approach, during LCM, the progress rate of the target task, *prate1*(a_{ijk}), is computed by considering the segments of the task that have the same priority. It is the ratio of execution segments of the target task versus other tasks (a_{xyz}), which will contend the target task. The $SEG(a_{ijk},tl,H_q)/prate1(a_{ijk})$ is the total number of segments considering contention. Hence, by subtracting required segments, $SEG(a_{ijk},tl,H_q)$, the contended segments due to the other tasks are produced. Finally, the contended segments are converted into time unit ($SD_{pred}(a_{ijk},tl,H_q)$: queuing delay) by multiplying the time quantum ($TQ(p,H_q)$).

In Step 1, LCM is computed by using periods of all task on the same host with the same priority, and during the LCM, required resource segments, ($S(a_{xyz},tl,H_q)$) for the target task on H_q are calculated.

Step 1. Compute the resource requirement of the target task (a_{ijk}) during the interval, [0,LCM].

The required resource segments for other tasks on H_q ($S(H_q,a_{ijk})$) in [0,LCM] can also be computed as in Step 1. The $LCM/T(a_{ijk})$ computes the number of arrivals of a_{ijk} during the LCM.

$$S(a_{ijk}, tl, H_q) = SEG(a_{ijk}, tl, H_q) \times \frac{LCM}{T(a_{ijk})}$$

Step 2. Compute the resource requirement of the other tasks (a_{xyz}) during the interval, [0,LCM].

$$S(H_q, a_{ijk}) = \sum_x \sum_y \sum_z SEG(a_{xyz}, tl, H_q) \times \frac{LCM}{T(a_{xyz})}$$

$$where,\ tl=tl(a_{ijk},\ c+1),\ a_{xyz} \in AL(H_q,t),\ x \neq i,\ p(a_{xyz})=p(a_{ijk}),$$
$$and\ Host(a_{ijk},t) = Host(a_{xyz},t)\ \forall x,y,z$$

It must be noted that the conditions of other tasks (a_{xyz}) are the same as the *worst-case*. For condition 3) in section 4.1, the applications in the same path as the target task, a_{ijk}, are ignored, as a_{ijk} can start after successor ($succ(a_{ijk})$) is finished, and the predecessor of the target task, ($pred(a_{ijk})$) can start after a_{ijk} is finished. It can be said that they are execution dependent.

$S(H_q, a_{ijk})$ plus $S(a_{ijk},tl, H_q)$ should be less than LCM; otherwise, tasks can not meet time requirements. The resource demands for the LCM is computed at Step 1 and Step 2. In Step 3, rate of progress ($prate1(a_{ijk})$), a metric to account contention, is computed simply by comparing the relative ratio of required resources.

Step 3. Compute rate at which requests of a_{ijk} for resource H_p are serviced.

$$prate1(a_{ijk}) = \frac{S(a_{ijk}, tl, H_q)}{S(H_p, a_{ijk}) + S(a_{ijk}, tl, H_q)}$$

2.2.2 The Second Rate of Progress Technique

An improved approach called ***RP-2*** considers the probability of the target task being blocked by any other tasks. The basic comcept is the same as ***RP-1*** introduced in section 2.2.1. The improved progress rate, $prate2(a_{ijk})$, is presented in this section. This technique considers all possible probability that the target task could progress, including computation of progress rate within contention with other tasks. The following formula (3) shows the computation of the queuing delay of the target task, a_{ijk}, on H_q, ($SD_{pred}(a_{ijk},tl,H_q)$).

$$SD_{pred}(a_{ijk}, tl, H_q) = \left(\frac{SEG(a_{ijk}, tl, H_q)}{prate2(a_{ijk})} - SEG(a_{ijk}, tl, H_q) \right) \times TQ(p, H_q) \tag{3}$$

To compute $prate2(a_{ijk})$, utilization of the target task ($U(a_{ijk},tl,H_q)$) and utilization of other tasks ($U(H_q,a_{ijk})$) on resource H_q are computed at Step 1. Welch et al (1995) compute the utilization of tasks over the LCM, but in this study, utilization of a task is computed over the period ($T(a_{ijk})$) of the task. The results are the same.

Step 1. Compute utilization of the target and other tasks.

$$CUP(a_{ijk}, tl, H_q) = C_{pred}(a_{ijk}, tl, H_q) \times \frac{1}{T(a_{ijk})}$$

$$CUP(H_q, a_{ijk}) = \sum_x \sum_y \sum_z C_{pred}(a_{xyz}, tl, H_q) \times \frac{1}{T(a_{xyz})}$$

$$where,\ tl=tl(a_{ijk},\ c+1),\ x \neq i,\ p(a_{xyz})=p(a_{ijk}),$$
$$and\ Host(a_{ijk},t) = Host(a_{xyz},t)\ \forall x,y,z$$

The condition of other tasks are the same as the *worst-case* and the ***RP-1***. In Step 2, the improved rate of progress is computed by considering: 1) the probability of being with no usage of the resource ($cp_0(a_{ijk})$), 2) the chance that only a_{ijk} uses the resource ($cp_1(a_{ijk})$), and 3) the chance that there is a_{ijk} progress rate within contention ($cp_2(a_{ijk})$).

Step 2. Compute progress rate

$$prate2(a_{ijk}) = cp_0(a_{ijk}) + cp_1(a_{ijk}) + cp_2(a_{ijk})$$

$$cp_0(a_{ijk}) = 1 - (U(H_q, a_{ijk}) + U(a_{ijk}, tl, H_q)) = CIP(H_q, t)$$
$$cp_1(a_{ijk}) = U(a_{ijk}, tl, H_q)$$
$$cp_2(a_{ijk}) = U(H_q, a_{ijk}) \times prate1(a_{ijk})$$

3 Contention Analysis for Tasks at Different Priority Levels

Priority scheduling in distributed environments is a sub-problem of distributed resource management. At the present time, real-time priority services on COTS operating systems can provide powerful means to improve performance of tasks requiring time constraints. As noted in the literature review, different aspects of the priority management problem have been addressed in a number of studies. Especially, in [9], fixed-priority assignment and static-task allocations appear to be the most common approaches employed by resource management systems in distributed real-time systems. However, the problem of the queuing delay estimation for the dynamic real-time systems using real-time priority level on COTS operating systems in order to manage QoS has not heretofore been addressed before. This section, computation of queuing delays due to the higher real-time priority tasks, $HD_{pred}(a_{ijk},tl,H_p)$, is ignored due to the page limit. Shortly, as used in section 2, all formulas are identical as previous sections except tasks' priorities considered for computation of process contention.

4 Synthesizing Contention Analysis for Response Time Prediction

In this section, the final prediction techniques for COTS operating systems under dynamic environments by synthesizing previous approaches are presented. These techniques are used for same priority level as well as for different priority levels. Therefore, the final response time of a_{ijk} ($= \lambda_{pred}(a_{ijk},tl,H_q)$) is derived by the following formula (4):

$$\begin{aligned} \lambda_{pred}(a_{ijk},tl,H_p) &= C_{pred}(a_{ijk},tl,H_p) + D_{pred}(a_{ijk},tl,H_p) \\ D_{pred}(a_{ijk},tl,H_p) &= SD_{pred}(a_{ijk},tl,H_p) + HD_{pred}(a_{ijk},tl,H_p) \end{aligned} \tag{4}$$

The following Table 1 shows possible choices (s1 to s13) for synthesizing $SD_{pred}(a_{ijk},tl,H_p)$ and $HD_{pred}(a_{ijk},tl,H_p)$ computed by 5 different approaches for each. The same technique used for computing $SD_{pred}(a_{ijk},tl,H_p)$ and $HD_{pred}(a_{ijk},tl,H_p)$, shown in diagonal of Table 1, might be possible selections for synthesizing to compute total response time. Combination of the *worst-case* and other probabilistic approaches like RP-2 also might be a good candidate..

Table 1. The synthesized cases for the final response time prediction.

Same priority *Different priority*	*Worst-Case*	*RP-1*	*RP-2*
Worst-Case	S1	S6	S7
RP-1	S10	S2	
RP-2	S11		S3

5 Experiments

In experiment, the filter application in DynBench [10] is also used as a target task examined on the dynamic and monotonic workload scenario. Several contention conditions derived by periods and workloads are considered in this experiment.

As Shown in Fig. 1 the combined scenario brings more variations in response time prediction, even though CPU is only used up to 65 percent. The "s1" overuses the CPU resource 29 percent in this experiment, while "s3" exceeds 2.8 percent only. Fig. 1 shows the comparison of each synthesized prediction technique.

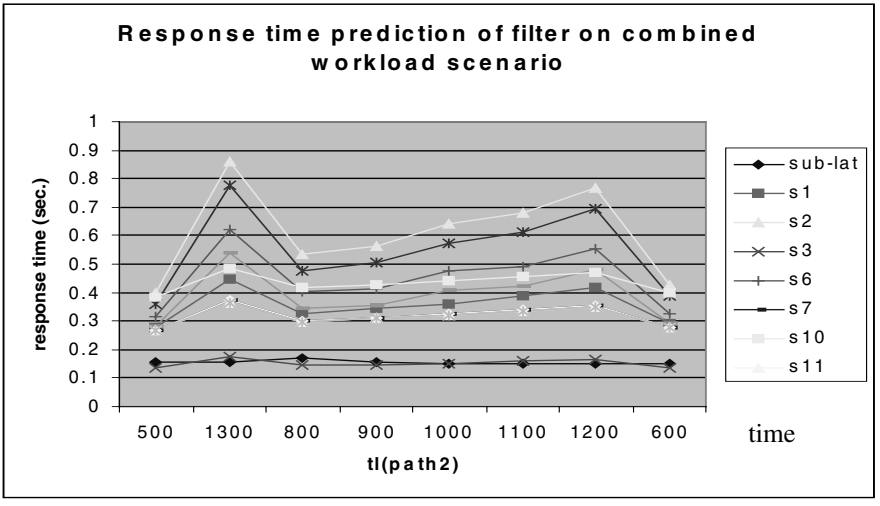

Fig. 1. Response time prediction by synthesized techniques on the combined workload

From the experiment presented, there is no measurement error under the 95% confidence interval. The “s3”, probabilistic contention analysis technique, RP-2, can accurately predict response time of an application on a COTS operating system, while the “s1” and the “s2” involving the *worst-case* analysis poorly predict response times. Therefore, these experiments for the dynamic real-time systems give strong analysis that the *worst-case* analysis poorly utilizes computational resources, and new approaches can predict response time on the COTS operating system with the dynamic environment constraints using current utilization. This suggest that schedulability of tasks can be predicted without using any real-time operating system components.

6 Conclusion

A new paradigm of the dynamic resource management technique was the focus of this research. This work mainly contributes to the area of QoS with an adaptive RM architecture, middleware and framework. This is accomplished with a mathematical

model for dynamic real-time systems, an execution time analysis, and a predictive response time analysis using real-time priority services on COTS operating systems for dynamic distributed real-time systems. The significant experiment results and solution procedures are deployed in analytical models. A statistical analysis tool was used to evaluate the accuracy of these new approaches compared with observed data and previous approaches.

Acknowledgement. This work was supported in part by the Korea Science and Engineering Foundation (R01-2001-000-00362-0) and by the Ministry of Information Communication of Korea, under the "Support Project of University Information Technology Research Center (ITRC)" supervised by KIPA.

References

1. Liu, C. L., and J.W. Layland, 1973. Scheduling algorithms for multi-programming in a hard-real-time environment. Journal of the ACM, Vol. 20: 46–61.
2. Ramamritham, J.A. Stankovic and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers, Vol.* 38(8), 110–123, August 1989.
3. Abeni, L. and G. Buttazzo, 1998. Integrating multimedia applications in hard real-time systems. *Proceedings of the 19th IEEE Real-Time Systems Symposium,* IEEE Computer Society Press: 3–13.
4. Stewart, D.B. and P.K. Khosla, 1997. Mechanisms for detecting and handling timing errors. *Communications of the ACM , Vol.* 40(1): 87–93.
5. Puschner, Peter and Alan Burns, 2000. A Review of Worst-Case Analysis. *The International Journal of Time-Critical Computing Systems*, Vol. 18: 115–128.
6. Atlas, A., and A. Bestavros, 1998. Statistical rate monotonic scheduling. *Proceedings of the 19th IEEE Real-Time Systems Symposium,* IEEE Computer Society Press: 123–132.
7. Lehoczky, John P., 1996. Real-Time Queueing Theory, *Proceedings of IEEE Real-Time Systems Symposium*, IEEE CS Press : 186–195.
8. Shirazi, B., A. R. Hurson, and K. Kavi, 1995. *Scheduling and Load Balancing in Parallel and Distributed Systems*, EEE Press.
9. Lehoczky, J.P, 1990. Fixed Priority Scheduling of Period Task Sets with Arbitrary Deadlines. *Proceedings of IEEE Real-Time System Symposium*, Los Alamitos, CA. IEEE computer society press.
10. Welch, L. R. and B. Shirazi, 1999. A Dynamic Real-Time Benchmark for Assessment of QoS and Resource Management Technology. *IEEE Real-time Application System*.