A Task Migration Scheme for High Performance Real-Time Cluster System

Makoto Suzuki
School of Science for Open and Environmental Systems
Keio University
Yokohama, Kanagawa, 223-8522, Japan

Hidenori Kobayashi
School of Science for Open and Environmental Systems
Keio University
Yokohama, Kanagawa, 223-8522, Japan

Nobuyuki Yamasaki
Dept. of Information and Computer Science
Keio University
Yokohama, Kanagawa, 223-8522, Japan

Yuichiro Anzai
Dept. of Information and Computer Science
Keio University
Yokohama, Kanagawa, 223-8522, Japan

Abstract

In a real-time system, it is attractive to use cluster computing system for realizing high performance and high availability. The objectives of the real-time clusters are maximize guarantee ratio and/or minimize the probability of failure to complete task in time. Because task arrivals are usually uneven among the nodes, some nodes may get temporarily overloaded while others are left underloaded or idle, which leads low performance for real-time clusters. Hence, we need a effective load sharing scheme.

In this paper, we describe task migration scheme for real-time cluster environment. Our system uses a single address space architecture because of high predictability. Hence, we have adopted Position Independent Code(PIC) to realize task migration. In our scheme, we introduce "*migration type*", which decides how to migrate a task by timing constraints of task, communication and whether first allocation or not. Using this scheme, any kind of real-time/non real-time task could be transferred without a crisis of deadline miss.

1 Introduction

It is well known that clusters of workstations are becoming a popular low cost alternative to supercomputers for high performance computing. In real-time system, it is also attractive to use cluster computing system for realize high performance and high availability. The correctness of real-time applications depends not only on the result, but also their timeliness. Hence, we have to consider the predictability in constructing real-time clusters.

In a real-time cluster system, the communication subsystem is a critical infrastructure for predictability[1]. The arrival of a packet generate a hardware interrupt and this in turn generate software interrupt that performs the protocol processing of packets making them ready for applications. These system-level activities for processing network packet preempt user-level applications and this processing time is charged to the preempted process. Consequently, the handling of incoming network packets leads low predictability of the system. In the extreme case, the packets can arrive at a rate fast enough where the system will spend all of its time on receiving and storing the packets, and the application is never scheduled to run, a phenomenon popularly called " receiver-livelock"[2].

There are two approaches to the above problem. One is centralized approach, which use global scheduling such as gang scheduling or co-scheduling. In gang scheduling, communicating tasks are scheduled simultaneously. So that the time used for network packet processing can reserved as task's execution time in advance. The other is distributed approach, which is reservation based. This approach uses local scheduler and require all communicating task to reserve real-time channel[3] in advance. When a task require a channel establishment, all the network related resources, such as network bandwidth, receiver node's buffer and CPU utilization for processing packet, are reserved.

While centralized approach is attractive to support parallel job scheduling and provide predictability, there are some shortcomings that is cost for paying a synchronization overhead and higher chance of fragmentation(of time slot), which leads low system utilization. So that we adopt a distributed approach to construct a real-time cluster.

From the performance point of view, real-time cluster system is different from that in general-purpose cluster system. The latter tries to achieve high throughput and to minimize average task response

time, whereas the former is intended to maximize guarantee ratio and/or minimize the probability of failure to complete task in time. Because task arrivals are usually uneven among the nodes, some nodes may get temporarily overloaded while others are left underloaded or idle[4]. These case leads low performance for real-time clusters. For example, upon arrival of aperiodic task which has some deadline to complete, if the node is busy, the task probably be rejected because the time for searching other node to execute incoming task may be over the task's deadline. Thus, we need a effective load sharing scheme.

In this paper, we focused on task migration scheme in real-time cluster environment designed to support high performance distributed real-time applications.

The rest of this paper is organized as follows. The hardware used for real-time cluster system and assumption task model are described in Section 2. The proposed scheme of task migration is described in Section 3. This paper Conclude with Section 4.

1.1 System Model

1.2 *Responsive Processor*

We have used *Responsive Processor*[5] as a processing node in real-time cluster. *Responsive Processor* is integrates many functions into an ASIC chip, such as a RISC processing core(SPARC), *Responsive Links* that realize real-time communication, many peripheral functions including SDRAMI/Fs, PWM generators, A/D converters, D/A converters, etc. for parallel/distributed real-time processing.

1.3 Task Model

For the *Responsive Processor* we developed a real-time operating system, which support periodic task and aperiodic task [6]. Periodic tasks are scheduled based on EDF algorithm, and aperiodic tasks submitted acceptance test based on "remaining" which is allocatable time to the task at the instant.

We assume three types of task, that is periodic task which has hard real-time constraint, sporadic task which has soft or firm real-time constraint and aperiodic task which has soft or non real-time constraint. All resources which is required by real-time task is known in advance. We consider parallel job as a number of communicating tasks.

1.4 Single Address Space Architecture

Because multi address space system architecture needs cache flush whenever context switch is occur, this cache flush result in low predictability of the system. To avoid this problem, we adopted single address space architecture in our operating system. Moreover, memory protection is realized by TLB's function.

To realize task migration in single address space architecture, we have used Position Independent Code(PIC), which faculty is a part of Executable and Linking Format(ELF). PIC is a executable code which can be loaded at any address in preparation for execution. Usually, all absolute symbol values must be located in a table, the global offset table(GOT), and accessed via GOT. Similarly, all function symbols are stored in a procedure linkage table(PLT), and run-time resolver is called before execution. We made such run-time resolver in our real-time operating system to support dynamic linked library.

2 Task Migration Mechanism

2.1 Load Sharing Algorithm

When system detect extreme uneven load balance and/or it is not possible to schedule incoming task locally, the system migrate tasks from a heavily loaded node to a lightly loaded node. This load sharing algorithm is implemented based on the following four interrelated policies.

Information Policy

This dictate how information should be exchanged among cluster nodes. Based on collected information from other nodes, each node decides a node to which task should be transferred. Because of predictability, we have adopted periodic information exchange, which can reserved as real-time channel in advance. The contents of exchanged information are CPU utilization, memory utilization and network utilization. This information policy is implemented as *exchanger* in Figure 1.

Transfer Policy

This decides when tasks are to be migrated. Our transfer policy determines whether a task can be guaranteed locally, and when the load of a node is more than the system average load. If incoming task cannot be scheduled locally, *admission controller* send "task transfer request" to *cluster manager* which is a entity of transferring task. When *exchanger* detect the uneven load balance, task transfer request is issued by one.

Location Policy

This decides the choice of a suitable node for a task

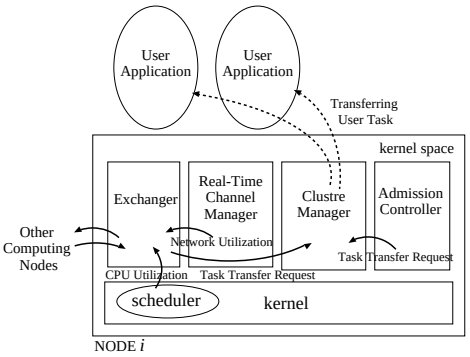

Figure 1: Structure of the Cluster Management

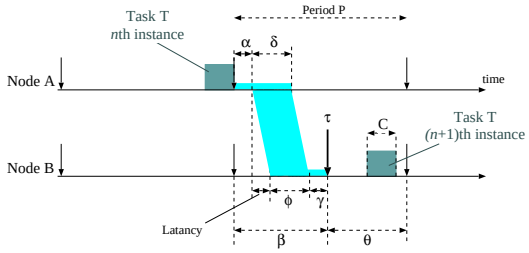

Figure 2: Sequence of the Task Migration

transfer, based on collected information by information policy. In our scheme, location policy manages a list, which hold other node's information in "capable node" order. Once transfer policy decides a task transfer, location policy find a suitable receiver node through polling in order of the list.

Selection Policy

This decides which task are to be transferred from the current node. We have introduced "migration cost", which is the degree of difficulty to migrate a task, based on memory size, CPU utilization, strictness of deadline and communications. When transfer policy of a node decides to transfer a task because the task cannot be guaranteed locally, selection policy basically decides not guaranteed task to transfer.

2.2 Reservation Mechanism

Once task transfer is decided using above load sharing algorithm, next three phases are proceeded. First phase is a reservation phase(admission control), in which all the needed resources such as CPU bandwidth, network bandwidth and memories for transferring task and execution of task, are reserved. If required resources are not reserved, this task transfer request is rejected. We designed it to guarantee execution of task migration and next task's instance in a period(Figure 2). Second, all the state of task and address space are transferred to destination node, which is transfer phase. Third, the state must be reconstructed correctly in establishment phase.

In the reservation phase, network utilization is reserved at first. If migrating task has m address size and maximum network bandwidth is N, needed network utilization is $m/(N \times \delta)$. When all available bandwidth is reserved, δ is minimized. α and γ are considered as a part of task's execution time for reservation. In δ and ϕ, there are some blocking time which is used for DMA transfer from Dual Port Memory(DPM) to SDRAM. If such blocking time is ε and which occurs n times, needed processor utilization for safety task migration is

$$U_A = \frac{C+\alpha}{P} + \sum_i \frac{C_i + (\varepsilon \times n)}{P_i}.$$

In the same way as U_A, U_B which is needed processor utilization for node B could be obtained.

2.3 *Migration Type*

In real-time environment, we have to consider how to transfer task without a crisis of deadline miss of transferred task. From this perspective, we introduce "*migration type*", which describe how to migrate a task. *Migration type* of a task is determined by the case whether a task transfer is required because the task cannot guaranteed locally or because the load of the local node is much higher than system average. In addition to the case, *migration type* is determined by the type of task.(e.g. hard real-time, soft real-time, non real-time, periodic, aperiodic)

We have defined four *migration type*:

1. fully reservation(FR),
2. copy and sync(CS),
3. loosely reservation(LR) and
4. no reservation(NR) type.

Fully reservation type reserve the system resources most strictly like the explanation in 2.2. This type is used for hard real-time periodic task. If a task's instance could be executed independently of other instance, the task could be migrated by copy and sync type. This type is separate address spaces transfer form transition of execution. Therefore, the resource which reserved for task migration is much less than fully reservation.

Table 1: The Relationships between *Migration Type* and Other Factors

Task Type	Timing Constraints	First Allocation	Independence of each Task Instance	*Migration Type*
Periodic Task	Hard	Y	Y/N	LR
		N	Y	CS
			N	FR
	Firm	Y	Y/N	LR
		N	Y	CS
			N	FR or LR
Sporadic Task	Firm	Y/N	-	LR
Aperiodic Task	Soft	Y/N	-	LR or NR
	Non	Y/N	-	NR

Loosely reservation type relaxed time constraints of migrating task. Fully reservation guaranteed execution of next instance in a period. However, loosely reservation doesn't guarantee it. This type is used for soft-real time task or first allocation.

No reservation doesn't reserve recourses at all. All of the migration process is executed in slack time. So that, there are no guarantee for timing constraint of migrating task. This type is used for non real-time tasks.

We show the relationship between *migration type* and task's properties such as timing constraints in Table1.

3 Conclusion

A method for migrating tasks which have timing constraints in real-time cluster system has been designed and implemented. Our system uses a single address space architecture because of high predictability, so that we have used Position Independent Code to realize task migration. Our task migration scheme features the *migration type*, which decides how to migrate a task by timing constraints of task, communication and whether first allocation or not. Using this scheme, any kind of real-time/non real-time task could be transferred without a crisis of deadline miss.

Acknowledgements

This study was performed through Special Coordination Funds of the Ministry of Education, Culture, Sports, Science and Technology of the Japanese Government.

References

[1] M. Apete, S. Chakravarthi, J. Padmanabhan, and A. Skjellum, "A Synchronized Real-Time Linux Based Myrinet Cluster for Deterministic High Performance Computing and MPI/RT," in *International Workshop on Parallel and Distributed Real-Time System(WPDRTS 2001)*, pp. 92–100, April 2001.

[2] S. Ghosh and R. Rajkumar, "Resource Management of the OS Network Subsystem," in *Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 271–279, 2002.

[3] D. Ferrari and D. C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 3, pp. 368–379, 1990.

[4] K. G. Shin and C.-J. Hou, "Design and Evaluation of Effective Load Sharing in Distributed Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, pp. 704–719, July 1994.

[5] N. Yamasaki, "Design and Implementation of Responsive Processor for Parallel/Distributed Control and Its Development Environments," *Journal of Robotics and Mechatronics*, vol. 13, no. 2, pp. 125–133, 2001.

[6] H. Kobayashi, "Design and Imprementaion of a Distributed Real-Time Operating System based on the Imprecise Computation Model," Master's thesis, Keio University, 2002.