Transaction Routing in Real-Time Shared Disks Clusters*

Kyungoh Ohn, Sangho Lee, and Haengrae Cho

Department of Computer Engineering, Yeungnam University,
Gyungsan, Gyungbuk 712-749, Republic of Korea
hrcho@yu.ac.kr

Abstract. This paper proposes a real-time transaction routing algorithm, which allocates real-time transactions to a node in a shared disks cluster. Unlike traditional routing algorithms, which consider load balancing and transaction affinity only, our algorithm also considers transaction priorities inherent to real-time applications.

1 Introduction

Although there has been a great deal of independent research in real-time processing and shared disks (SD) cluster, their aggregation has very little attention [3]. The cluster technology enables highly available real-time database services, which are the core of many telecommunication services. The cluster can also achieve high performance real-time transaction processing by exploiting inter-node parallelism.

This paper proposes a real-time transaction routing algorithm in the SD cluster. The transaction routing is a process of the front-end router to select an execution node for an incoming transaction. The traditional transaction routing algorithms of the SD cluster have two design goals: *load balancing* and *transaction affinity* [4, 5, 6]. The load balancing means to avoid overloading individual node. The transaction affinity means to execute transactions with similar data access pattern on the same node (*affinity node*). To support real-time transactions, we have additional goal of *transaction priority.* This goal means to reduce the number of transactions missing their deadlines by considering the deadline as a priority. The contribution of this paper is to extend a well-performed traditional algorithm, named DACA (*Dynamic Affinity Cluster Allocation*) proposed by authors [4], to the real-time transaction processing.

2 Algorithm

We propose a new real-time transaction routing algorithm, named P-DACA (*Priority conscious DACA*). In this section, we first describe our underlying model

* This research was supported by University IT Research Center Project.

and then summarize the basic idea of DACA algorithm [4]. Lastly, we present P-DACA algorithm that extends DACA for real-time transaction processing.

2.1 Assumption

A transaction router selects an execution node of each incoming transaction. The node schedules the execution of its transactions with *earliest deadline first* (EDF) policy [1]. We assume the *mixed* real-time transaction workload [2], which consists of real-time transactions and non real-time transactions. A real-time transaction is assigned with a deadline. Executing the real-time transaction after its deadline is meaningless.

We assume that the record-level locking is in effect. In real-time applications, the locking has to handle the *priority inversion* problem that lower priority transactions block the execution of higher priority transactions. To prevent the problem, we consider the locking model of [2]. Specifically, a real-time transaction aborts non real-time transactions holding locks in conflict mode. On the other hand, a non real-time transaction always waits at lock conflict. The same procedure holds between real-time transactions with different priorities. High priority transactions are always guaranteed to acquire locks without waiting.

2.2 DACA

To alleviate the routing overhead, DACA considers balancing the load of each *affinity cluster* (AC) [5]. An AC collects several transaction classes with high affinity to a given set of tables. The transaction router maintains routing parameters. Specifically, when a transaction router allocates a transaction of an affinity cluster AC_q to a node N_p, it increments both #T(AC_q) and #T(N_p), which means the number of active transactions at AC_q and at N_p respectively. Both counters are decremented when a transaction commits or aborts.

There are two overload types: *AC overload* and a *node overload*. The AC overload implies that transactions of an AC are rushed into the system. The node overload occurs when a node N_p is allocated to several ACs and #T(N_p) is over average. DACA balances the load of each node according to the overload type. If AC_q is overloaded, then DACA allocates more nodes to AC_q by *node expansion* strategy. If there is no AC overload but N_p is overloaded, then DACA distributes some ACs assigned to N_p to other node by *AC distribution* strategy.

DACA can make an optimal balance between the affinity-based routing and load balancing as follows. First, DACA tries to reduce the number of nodes allocated to the overloaded AC if the load deviation of each node is not significant. This allows DACA to reduce the frequency of inter-node cache invalidations. Next, DACA prohibits allocating both an overloaded AC and other ACs to a node. As a result, DACA can achieve high buffer hit ratio for the overloaded AC. Even though several non-overloaded ACs may be allocated to a single node, efficient handling of the overloaded AC is more important to improve the overall transaction throughput.

2.3 Priority Conscious DACA (P-DACA)

Before describing the details of P-DACA, we first illustrate the problem of DACA when transactions have priorities. Example 1 shows the problem.

Example 1: Suppose there are two ACs (AC_1, AC_2) and two nodes (N_1, N_2). N_1 is an affinity node of AC_1 and executes a transaction t_1 of priority 100. N_2 is an affinity node of AC_2 and executes a transaction t_2 of priority 60. At this time, suppose a new transaction t_3 of priority 70 arrives, and t_3 belongs to AC_1. Then DACA allocates t_3 to its affinity node N_1 if N_1 does not result in overload state. However, t_3 has lower priority than t_1 and cannot be executed until t_1 completes. So t_3 has a higher probability of missing its deadline. On the other hand, if t_3 is allocated to N_2, then it can be executed immediately. □

The goal of P-DACA is (a) to reduce the number of transactions missing their deadlines, and (b) to take advantages of affinity clustering as DACA. To achieve this goal, P-DACA performs the following three basic steps sequentially to decide where a new real-time transaction t_r will be routed to.

1. If there is an affinity node of t_r where the priority of t_r becomes the highest one, then allocate t_r to that node.
2. If there is a non-affinity node of t_r where the priority of t_r becomes the highest one, then allocate t_r to that node.
3. If there is no node that can execute t_r immediately, then allocate t_r to one of its affinity nodes.

The basic steps can be optimized if the transaction router maintains a *priority list* for each node. The priority list is a sorted list in descending order of priorities of active real-time transactions at the node. Then a new real-time transaction t_r has to be allocated to a node where t_r can be executed faster than other nodes. P-DACA checks this condition by comparing the relative position of t_r in the priority list of each node. Suppose $P(t_r)$ means the priority of t_r, and t_r is included in the affinity cluster AC_r. $\mathcal{S}(AC_r)$ is a set of affinity nodes of AC_r. The followings summarize the transaction routing algorithm of P-DACA.

```
1. #T($AC_r$) = #T($AC_r$) + 1;
2. If ($t_r$ is a real-time transaction) then
   (a) If ($\exists N_p \in \mathcal{S}(AC_r)$, rank($P(t_r), N_p$) $< w_1$ AND rank($P(t_r), N_p$) $<$
       rank($P(t_r), N_i$), for all $N_i \in \mathcal{S}(AC_r)$, $i \neq p$) then goto step 4;
   (b) Else if ($\exists N_p \notin \mathcal{S}(AC_r)$, rank($P(t_r), N_p$) $< w_2$ AND rank($P(t_r), N_p$) $<$
       rank($P(t_r), N_i$), for all $N_i \notin \mathcal{S}(AC_r)$, $i \neq p$) then goto step 5;
   (c) Else goto step 3.
3. Select $N_p$, where #T($N_p$) is minimum for all $N_i \in \mathcal{S}(AC_r)$;
4. If ($AC_r$ is overloaded) then call node_expansion($AC_r$);
5. Else if ($N_p$ is overloaded) then call ac_distribution($N_p$);
6. #T($N_p$) = #T($N_p$) + 1; Insert $P(t_r)$ into the priority list of $N_p$;
7. Return $N_p$.
```

At step 2, the function of $\texttt{rank}(P(t_r), N_p)$ returns the number of transactions whose priorities are higher than $P(t_r)$ at N_p. If the function returns 0, t_r will be a transaction with the highest priority at N_p. The values w_1 and w_2 are windowing constraints that limit the acceptable rank of t_r. w_1 is usually bigger than w_2 since an affinity node of t_r could complete t_r faster. Note that if w_1 and w_2 are set to 1, the algorithm works similarly to the above basic steps. A non real-time transaction is allocated to one of its affinity nodes with the lightest load (step 3). If allocating t_r would cause an AC overload or a node overload, then the resolution strategy of DACA has to be performed (step 4 and 5). Example 2 shows how P-DACA can resolve the problem of Example 1.

Example 2: Suppose that the information of ACs, nodes, and transactions are same to Example 1. Suppose also that both w_1 and w_2 are set to 1. Then the transaction router allocates t_3 to N_2 that can execute t_3 immediately. This is because (a) the affinity node of t_3 is N_1 but $\text{rank}(P(t_3), N_1) = 1 = w_1$, and (b) even though N_2 is not an affinity node of t_3 but $\text{rank}(P(t_3), N_2) = 0 < w_2$. Note that if w_1 is set to 2, t_3 is allocated to N_1.□

3 Concluding Remarks

We proposed a new transaction routing algorithm for the real-time SD cluster, named P-DACA. The notable features of P-DACA are two-fold. First, P-DACA allocates a real-time transaction to a node that guarantees higher probability of completing the transaction within its deadline. Even though the transaction could miss its deadline due to succeeding transactions with higher priority, the selection strategy is the best choice at the current state. Next, P-DACA tries to allocate a transaction to its affinity node if possible. So P-DACA can achieve high buffer hit ratio. This reduces the transaction execution time, and the probability of missing deadline can be reduced as a result.

References

1. Lam, K-Y., Kuo, T-W. (ed.): Real-Time Database Systems: Architecture and Techniques. Kluwer Academic Publishers (2000)
2. Lam, K-Y., Kuo, T-W., Lee, T.: Strategies for resolving inter-class data conflicts in mixed real-time database systems. Journal of Syst. and Soft. **61** (2002) 1-14
3. Lee, S., Ohn, K., Cho, H.: Feasibility and Performance Study of a Shared Disks Cluster for Real-Time Processing. Lecture Notes in Computer Science **3397** (2005) 518-527
4. Ohn, K., Cho, H.: Cache Conscious Dynamic Transaction Routing in a Shared Disks Cluster. Lecture Notes in Computer Science **3045** (2004) 548-557
5. Yu, P., Dan, A.: Performance Analysis Clustering on Transaction Processing Coupling Architecture. IEEE Trans. Knowledge and Data Eng. **6** (1994) 764-786
6. Yu, P., Dan, A.: Performance Evaluation of Transaction Processing Coupling Architectures for Handling System Dynamics. IEEE Trans. Parallel and Distributed Syst. **5** (1994) 139-153