Load Sharing in Hybrid Distributed - Centralized Database Systems

Bruno Ciciani, Daniel M. Dias, and Philip S. Yu

IBM Research Division, T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

In a hybrid distributed-centralized database system architecture, some transactions run at (geographically) distributed systems, and other transactions at a central computing complex. Such a system can provide the advantage of distributed systems for transactions that refer principally to local data, and also the advantage of centralized systems for transactions that access a lot of non-local data. Several transaction processing applications such as reservation systems, insurance and banking exhibit regional locality, and can be applied to the hybrid system structure. In this paper we study static and dynamic load sharing strategies for such hybrid database systems. The strategies take into account not only the difference in load at different sites, but also the effect of routing on data contention and transaction abort probabilities. In our dynamic strategies are examined and are compared with an optimal static strategy. A dynamic strategy that is based on analytical estimates of the effect of routing on all transactions in the system, rather than that of the incoming transaction alone, is found to be best among the strategies examined.

1. Introduction

Various transaction processing applications such as reservation systems, insurance and banking exhibit regional locality and load fluctuations. The hybrid distributed-centralized database system architecture [CIC87] is a number of geographically distributed systems connected by a network to a central computing complex maps well to these types of applications. The architecture is motivated by the desire to combine advantages of the centralized database architecture with those of the distributed database architecture. In the fully centralized system, where user terminals are connected by a network to the central computing complex, all transaction input messages are shipped to the central site, where the transaction is processed, and output messages are sent back to the terminal; hence the centralized system does not make use of geographical locality of data reference. In the geographically distributed database approach [GRAY86, LARS85, WIL82], the databases are partitioned and distributed among regional processing systems, and some request routing mechanism is provided to support the access of remote systems [CORN86, IBM83]. The performance of the fully distributed system depends critically on the number of remote calls that a transaction makes for data; the performance of the distributed system is better than the centralized system if the number of remote calls per transaction is significantly less than one, but is much worse otherwise [DIAS87]. In [CIC87A,B], protocols for the hybrid system were investigated, and the hybrid architecture was shown to provide the advantages of (geographically) distributed systems for transactions that refer principally to local data, and also provide the advantage of centralized systems for transactions that access a lot of non-local data. These protocols use a pessimistic approach for concurrency control among transactions running at the same system, and an optimistic approach for concurrency and coherency among transactions running at different sites.

In this paper we investigate load sharing strategies in such hybrid systems. Static load sharing strategies for star configured systems have been investigated in the literature [TANT84, TRI83]. Optimal deterministic allocation of tasks have been considered in [STON77, 78]; while optimal probabilistic assignment is considered in [TANT85]. Dynamic transaction routing strategies that minimize the estimated response time of each incoming transaction are proposed and studied in [YU86] for a partitioned database environment. A principal difference between the previous work [WANG85] and this study is the interaction between jobs run on different systems. Here we consider a transaction processing environment in which transactions may be run either at distributed sites or at the central site, by replicating data appropriately and by using global concurrency and coherency control protocols. Since transactions running at the central and local sites may access the same data, data contention results and this is manifested as aborts of local or central transactions. Hence, routing decisions depend not only on load and communications delay, but also on data contention among transactions. In the literature, transactions (jobs) are considered to be logically independent so that there are no contention or aborts. The effect of concurrency and coherency control protocols is also ignored in the literature.

We first look at optimal static load sharing, using an analytical model that captures the factors alluded to above. Next, we examine two dynamic load sharing schemes, that attempt to minimize the incoming transaction response time. These schemes base their estimates on (a) CPU queue length at central and local sites and (b) number of transactions, at the central and local sites (including transactions in commit processing, I/O, and contention wait). The schemes then use a similar analytical model to that for static load sharing to estimate response times. These schemes, as the previous work in the literature, attempt to minimize the response time of the incoming transaction. In this environment, this can be at the expense of transactions running at the central site, so that the average response time of all jobs is not minimized. We examine a dynamic strategy that attempts to take into account the effect of routing on all jobs in the system, rather than on the incoming transaction alone. In the literature, the send-to-shortest queue strategy is found to be the best for the environment with Poisson arrivals and identical exponential servers [WN87]. Since the servers are not identical in this environment, send-message threshold heuristics [EAGE86A,B] are examined. We compare the two dynamic load sharing schemes using estimates of response time with threshold heuristics based on queue length, and on measured response time. We use simulations to study the performance of these schemes.

These strategies require information on the system state. If a significant amount of overhead is needed to collect this information, then dynamic schemes become impractical. For this reason, the dynamic
The scenario considered is illustrated in Figure 2.1. User workstations are assumed to be connected to a geographically distributed computer system through a long-haul communications network. The distributed systems in turn are connected to a central computing complex through a long-haul communications network.

The communication mechanism itself does not ensure that transactions use consistent data. We assume that when data is replicated at the central and distributed systems, the data at the central site would have to access remote data at the distributed systems resulting in poor performance at the central site. For transactions accessing data that are completely local to a distributed system (and this is often a significant fraction for cases where the distribution is of interest), it should be possible to commit such transactions [DATE81] without waiting for a communication to the central site; otherwise, it would be preferable to ship the entire transaction to the central site eliminating the advantage of the distributed system. This means that when data that is replicated at the central and distributed sites is updated at the distributed site, the update must be propagated asynchronously to the central site; consequently, the data at the central and distributed sites are not always identical, and the concurrency/ coherence protocol must ensure that transactions use consistent data.

The following classes of transactions are identified. Some transactions only require local data, and do not need data from any other site (class A). The second class B of transactions usually require non-local data. We assume that with some preprocessing it is possible to identify the class of a transaction. Class A transactions can be run at either the local system or the central system. The lock manager maintains two fields for each lock — a concurrency control field (share or exclusive) and a coherence control field (used to maintain coherence of data between the central and distributed sites). The concurrency control field is used with usual locking protocols to ensure that compatible lock requests are granted and that incompatible lock requests are queued. The coherence control field is used to maintain a count (see below) of updates being propagated to the central site, and is initially set to zero. At transaction commit time, first a check is made if the locally running transaction has been marked for abort (by a committed shipped or central transaction), as explained below. If so, the local class A transaction is aborted and restarted. If the local class A transaction is not marked for abort, the concurrency control field is used to release the share or exclusive lock held by the transaction (transactions queued on the entity can then be granted the lock); further, the coherence field count of the lock is incremented to indicate pending response from the central site. Asynchronously, a message is sent to the central site indicating the locks released by the transaction, and a copy of the blocks updated by the transaction is sent along with this message. When the central site responds that it has processed the message (as detailed below) the coherence field pending message count is decremented. Meanwhile the local transaction completes without waiting for this message to be sent or response received, and thus provides good response time for purely local transactions. Further, another local transaction can lock and update the data, increment the coherence count and send another asynchronous update message to the central site before the acknowledgement for the first message is received. These asynchronous messages may also be batched to reduce the overheads involved. Note however, that the communications protocol must ensure that these asynchronous messages are delivered and processed at the central site in the order that they were originated. If the communication mechanism itself does not ensure this, then a higher level protocol must ensure that if a transaction commits and has locks on items with non-zero coherence count, then (though the transaction can commit) the asynchronous messages to the central site are queued until the previous messages to the central site are acknowledged.

Central (class B) and shipped (class A) transactions are transmitted to the central site where they access (replicated) data locally. At the central site, local locks are used for concurrency control among transactions running at the central site. However, as described above, transactions at the distributed sites may concurrently use and update data. In contrast to the central site, the local site checks if the data was propagated asynchronously to the central site. When the central site receives a message from the distributed sites indicating locks obtained and updates, any locks at the central site on the data are invalidated (transactions holding these locks are marked for abort), the data is updated and an acknowledgement is sent back to the distributed site. As transaction commit time, the central site checks if the transaction has been marked for abort due to invalidated locks held; if not, the central site simultaneously sends, to all the sites that are masters of the data locked, (for shipped class A transactions only the source local site is involved) a list of the locks and mode required along with a copy of the block updated by the transaction. This is referred to as the authentication phase. The distributed sites examine the lock status (both concurrency and coherence); if the coherence control status is not null (i.e. there are some in flight asynchronous updates to the data), then a negative acknowledgement is sent to the central site. If the coherence control field is null, and the locks requested by the central/shipped transaction are compatible with the locks currently held at the distributed site, then the locks are also granted to the central/shipped transaction, and a positive acknowledgement returned to the central site. If, the locks held at the distributed site are incompatible with those requested by the central/shipped transaction, the local transactions holding these locks are marked for abort, the central/shipped transaction is granted the locks and the locks held by
the conflicting local transactions are released. (Note that when the corresponding local class A transactions abort, they will no longer be holders of the locks granted to the central/shipped transaction; the local transactions must reacquire the locks Inter.) If the central site receives a positive acknowledgement from all the involved distributed sites, it checks to ensure that the acquired locks were not invalidated by asynchronous updates at distributed sites, and then it sends a commit message to all the involved sites. If not, it re-executes the transaction and repeats the process. When the distributed sites receive the commit message, the database is updated and the locks corresponding to the transaction at the central site are released.

3. Load Sharing Strategies

In this section we consider several load sharing strategies for the hybrid architecture and protocols outlined above. Recall that Class A transactions refer only to local data, and that the central site replicates the data held at the central site. Hence, Class A transactions may be run either at the local or central site. For cases where distributed systems are of interest, a large fraction of the transactions (typically of the order of 75%) are of this class. Class B transactions refer to global data and are shipped to the central site. (Potentially, these transactions could be run at a local site, making remote function calls to the central site to obtain required data; however, we do not analyze this possibility here.) The load sharing decision considered here is whether to run a Class A transaction at the local site, or to ship it to the central site for execution.

The objective of the load sharing strategy is to minimize the average transaction response time. Notice that, when a local transaction is shipped to the central site, we have to consider the communications delay, the load and MIPS difference between the local and central sites, and the possibility of increasing the probability of data contention and aborts due to shipping. These factors mean that we consider load sharing as an optimization, rather than load balancing (that would balance utilization levels).

In the case of static (probabilistic) load sharing, it is assumed that the arrival rates of the transaction types are known. Then, an analytical model is used to estimate the average response time for different probabilities of shipped incoming class A transactions to the central site for execution. The value of $p_{shipped}$ that results in the minimum average response time is selected, and used in simulations as a basis for comparison. The dynamic load sharing schemes use simple measures of the system state on which to base the decision of whether to ship the class A transaction or not. The principal dynamic schemes are based on estimating the response time of shipped or local class A transactions using a variation of the same analytical model used for the static load sharing. These schemes are compared with other heuristics for dynamic load balancing.

We first present the static load sharing, since the analysis method used forms the basis of the dynamic load balancing. The analysis method is an extension of that in [CIC87B] and transaction parameters are based on those reported in [YU87]. Then, the schemes for dynamic load sharing are detailed.

3.1 The Model and Static Load Balancing

As outlined in the scheme above, transactions at the local site use locking to resolve contention with other transactions running locally. Similarly, transactions running at the central site also use locking to resolve contention with other centrally running transactions. An optimistic approach is used to resolve local-central contention, and either local transactions or central transactions are aborted due to such local-central contention. The analysis in this section is an extension of that in [CIC87B]. Hence, the methodology is outlined, and some details omitted.

Because of the difference in their response times we distinguish six kinds of transactions: (i) newly arrived class A transactions assigned to run at local sites, with average response time of $R_{classA}$, (ii) transactions that run at the local site after being aborted at least once (re-run local transactions) with average response time of $R_{classB}$ (iii) new class A transactions shipped to the central sites with average response time of $R_{classA}^{ship}$, (iv) shipped class A transactions that run at the central site after being aborted at least once (re-run shipped transactions) with average response time of $R_{classA}^{ship}$, (v) newly arrived class B transactions assigned to the central site (that run for the first time), with response time $R_{classB}$, and (vi) class B transactions that run at the central site after being aborted at least once (re-run transactions) with average response time of $R_{classB}^{ship}$. Locks are assumed to be all held until the end of the transaction. For the estimation of contention and abort probabilities, we are leaving out an initial portion of the transaction response time, during which the transaction goes through a transaction set-up phase (during which no locks are held). A transaction that is re-run after an abort is modeled to find all data referenced in its main memory. Further, locks at either the local (respectively central) site are not released after an abort. The difference between $R_{classA}$, $R_{classB}$, $R_{classA}^{ship}$ and $R_{classB}^{ship}$, respectively, is the I/O time and processing time and wait time for locks.

We define a collision as an event in which a transaction requests a lock that is currently held by another transaction in an incompatible mode. Collisions among local (respectively central) transactions are manifested as lock contention wait for the element to be unlocked by the local (respectively central) transaction holding the lock. Collisions between local and central transactions manifest themselves as aborts of either the local or central transaction. The probability of contention per lock request can be projected as below. We start from the lock contention probability $P_{c}$, that is observed in the trace driven simulation of a centralized system, subject to a transaction rate of $\lambda$ [YU87]. The average response time, when holding locks, for transactions in the measured system is denoted as $R$. Extending the asymptotic results of the analysis in [MTR84] and [LAVE84], lock contention is adjusted by projecting it to be directly proportional to (a) transaction rate per database, (b) number of locks per transaction to a database and (c) average holding time for a lock. This method for estimating lock contention probability is generalized to estimate collision probabilities in the hybrid system in [CIC87A, B], where simulation estimates are shown to support this methodology.

We define lock collision probability for a centrally running (class B or shipped class A) transaction $P_{c, cen} = P_{c, cen} + P_{c, cen}$, where $P_{c, cen}$ is the probability that the transaction with which the collision occurs is a centrally running (class B or shipped class A) transaction, while $P_{c, cen}$ is the probability of a similar collision with a local transaction holding the lock desired. We divide $P_{c, cen}$ into $P_{c, cen} - P_{c, loc}$, and $P_{c, cen}$, depending on whether the local transaction, which was the current holder, was a new transaction or a re-run transaction, respectively. Similarly $P_{c, loc}$ is defined as the lock collisions per lock request for a local transaction. $P_{c, loc} = P_{c, loc} + P_{c, loc}$, where $P_{c, loc}$ is the probability of collision with a central (class B or shipped class A) transaction and $P_{c, loc}$ is the probability of collision with another local transaction. We divide $P_{c, cen}$ into $P_{c, cen} - P_{loc}$ and $P_{c, cen}$, the first two factors result in aborts, while the third term is due to central transactions holding locks at local sites during the authentication phase, and these translate into waits of local transactions. (The shipped class A and class B transactions have different behaviour during the authentication phase. For simplicity in the presentation, we assume that their response times are equal.)

Each local system has a local database and is subject to a transaction rate $\lambda$. Of these transactions, a fraction $p_{loc}(1 - p_{lock})$ execute locally. The central site has a copy of each database in the distributed systems. The transactions arriving at the central site are assumed to access the databases at the central site uniformly, thus offering a transaction arrival load of $N((1 - p_{loc}) + (p_{loc} + p_{lock}))$ per database.
at the central site. Based on our assumptions about the growth in lock collision probability, we may write:

\[ P_{\text{cen cen}} = P_{\text{cen cen}} + P_{\text{cen cent}} \]

\[ P_{\text{cen cen}} = C \lambda (1 - P_{\text{loc}} + (P_{\text{loc}} \times P_{\text{shp}})) N_{l} \frac{\beta}{2} \]

\[ P_{\text{cen cent}} = C \lambda ((1 - P_{\text{loc}} + (P_{\text{loc}} \times P_{\text{shp}})) \frac{P_{\text{ac}}}{1 - P_{\text{ac}}} N_{l} \frac{\beta}{2} \]

where the constant \( C = \frac{\beta}{(\lambda N_{l} \frac{\beta}{2})} \) is derived from the trace analysis. The terms \( \frac{\beta}{2} \), and \( \gamma \) are the average lock holding times for the first and rerun central transactions. \( P_{\text{cen cen}} \) is the sum of two terms. In the first term, \( P_{\text{cen cen}} \) is the probability of collision with a new central or shipped transaction, \( \lambda (1 - P_{\text{loc}} + P_{\text{loc}} \times P_{\text{shp}}) \) is the new transaction arrival rate per database at the central site. Each transaction makes \( N_{l} \) local requests per transaction. The average lock holding time of a new transaction is \( \beta \). The second term, \( P_{\text{cen cent}} \) is the probability of collision with re-run transactions at the central site. The term \( P_{\text{ac}} \) is the probability that a central transaction aborts at the end of its first execution, and \( P_{\text{loc}} \) is the abort probability for a central rerun transaction. Central transactions re-execute at the rate \( \lambda (1 - P_{\text{loc}} + P_{\text{loc}} \times P_{\text{shp}}) \frac{P_{\text{ac}}}{1 - P_{\text{ac}}} \) per database and the average lock holding time is \( \gamma \). The collision probabilities among local transactions can be derived in a similar manner.

The collisions between local and central transactions translate into aborts of either the central or local transaction involved. We illustrate the approach for estimating the abort probabilities \( P_{\text{ac}} : P_{\text{ac}} \times P_{\text{loc}} : P_{\text{loc}} \times P_{\text{shp}} \), and \( P_{\text{loc}} \), by describing the derivation for \( P_{\text{loc}} \) (i.e. the abort probability of a new local transaction). A new local transaction aborts due to collision with either a new shipped or central transaction, or with a re-run shipped or central transaction, provided that the local transaction finishes after the authentication phase of the shipped or central transaction with which there was a collision. (Note that the corresponding central transaction would abort if the local transaction finishes before its authentication.) The sources of collision in this case are,

\[ P_{\text{loc cen}} = C \lambda (1 - P_{\text{loc}} + P_{\text{loc}} \times P_{\text{shp}})) N_{l} \frac{\beta}{2} \]

\[ P_{\text{loc loc}} = C \lambda ((1 - P_{\text{loc}} + P_{\text{loc}} \times P_{\text{shp}})) \frac{P_{\text{ac}}}{1 - P_{\text{ac}}} N_{l} \frac{\beta}{2} \]

\[ P_{\text{cen loc}} = P_{\text{cen loc}} = C \lambda P_{\text{loc}} (1 - P_{\text{shp}})) N_{l} \frac{\beta}{2} \]

where \( \beta \) is the average lock holding time for the first run of a local transaction. From each of these collision probabilities, a collision rate can be computed, and then distributed among local and central aborts. For instance, the abort rate of new local transactions corresponding to \( P_{\text{loc cen}} \) is, \( P_{\text{loc cen}} \times P_{\text{loc}} \times N_{l} \times \lambda \times P_{\text{loc}} (1 - P_{\text{shp}}) \), where \( P_{\text{loc}} \) is the probability that the local transaction finishes after the termination of the central transaction plus the communications delay given that there is collision of this type. These local aborts are distributed over \( P_{\text{loc}} (1 - P_{\text{shp}}) \) new local transaction per second running at the local site. Thus the corresponding local abort probability is \( P_{\text{loc cen}} = P_{\text{loc}} \times N_{l} \). The probability \( P_{\text{loc}} \) can be estimated based on approximations for the distributions of the residual time of a local and a central transaction. For a collision corresponding to \( P_{\text{loc cen}} \), considered above, the local transaction makes a lock request when a (first run) central transaction is holding the lock. We assume that the local transaction makes lock requests uniformly over the time of its first run. Then the residual time of the local transaction for this case is uniformly distributed between 0 and \( \beta \). For the same collision, to estimate the central transaction residual response time we observe that the probability of collision is proportional to the number of locks held; hence the probability of remaining time \( x \) is approximated as being proportional to \( (\beta - x) \), for \( 0 \leq x \leq \beta \). During the communication delay from the central to local site for authentication, the remaining time is uniformly distributed. Based on these residual time distributions, the probability of \( P_{\text{loc}} \) is computed.

Once the collision and abort probabilities are estimated, the response times of the local, shipped, and central transactions can be estimated as in [1187B]. The local transaction response time is estimated as

\[ \text{RTL} = \text{R}_{\text{die}} + \frac{\text{R}_{\text{cpu}}}{1 - \text{R}_{\text{die}}} + \frac{\text{R}_{\text{proc}} + \frac{\text{R}_{\text{proc}}}{\text{R}_{\text{cpu}}}}{1 - \text{R}_{\text{die}}} \frac{\text{R}_{\text{die}} - \frac{\text{R}_{\text{proc}}}{\text{R}_{\text{cpu}}}}{1 - \text{R}_{\text{die}}} \]

In this equation, \( \text{R}_{\text{die}} \) is the initial I/O time per transaction before processing of the application begins (during which no locks are held). The instructions executed by the transaction (for the first run) are divided into those for database call processing associated with \( \text{R}_{\text{cpu}} \) (10 calls per transaction with 30K instructions per call) and those for message processing and transaction initiation, associated with \( \text{R}_{\text{proc}} \) (150K instruction per transaction). \( \text{R}_{\text{die}} \) is the I/O time associated with database call processing. \( \text{R}_{\text{cpu}} \) is the probability of collision on a lock request with a central or shipped transaction that holds locks at the local site during its authentication phase, and \( N_{l} \) is the number of locks per transactions. The denominator in the third term causes an expansion in response time due to local lock contention wait. Finally, the fourth term is for rerun local transactions due to aborts. In a similar manner [1187B], the response time of the central/shipped transaction can be derived.

### 3.2. Dynamic Load Sharing

In the dynamic load sharing schemes, for each incoming transaction a routing decision is made based on the current state of the system, rather than the a-priori decision of the static load sharing scheme. The typical approach used in the literature is to attempt to minimize the response time of the incoming transaction. This is the first approach that we examine. To do this, we estimate the class A average response time using a similar model to that used for the static load balancing. We note, however, that minimizing the class A transaction response time can adversely affect currently running transactions. Therefore, we next examine schemes that attempt to route the incoming transactions so as to minimize the expected response time of the currently running transactions. Finally, we examine heuristics that route transactions based directly on queue length or measured response time.

Recall that the dynamic load sharing schemes must use minimal information that is easy to gather, so as to be practical, and this information is delayed due to communications delays between the local and central sites. Ideally, the response time of the new transaction should be analyzed with a transient model and with complete instantaneous knowledge of the states of the local and central sites. For applicability, the two proposed strategies use steady state solution techniques to estimate the response time and base the routing decisions on these estimates. As we will see, such an approach yields good results.

#### 3.2.1 Minimization of Incoming Transaction Response Time

The first dynamic scheme in this section uses information of the CPU queue length at the local and central system, the number of locks held at the local and central site, and a count of the number of transactions running locally, and currently shipped to the central system. From this information, the response time is estimated for the options of running the transaction locally, or shipping it to the central site. The
option that results in a shorter estimated response time is used. The second scheme uses information of the number of transactions running at the local and central sites instead of the CPU queue length.

(a) Response Time Based on Queue Length Strategy

The data used by this scheme is the CPU queue length (including any running jobs) $q$ at the ith local system, $n_i$ at the central system, the number of locks $n_{lock}$, $n_{lock}$ held at the ith local and central sites respectively, the number of class A jobs $n_{jobA}$, $n_{jobA}$ running at and shipped from the ith local system. From this we estimate the utilization factors, the probabilities of contention, and probabilities of abort. Then we use the model of the previous section to estimate the response times. First we estimate the local utilization $\rho_i$ and central utilization $\rho_c$

\[
\rho_i = \frac{q_i + a}{q_i + 1 + a} \quad ; \quad \rho_c = \frac{q_c + a}{q_c + 1 + a}.
\]

Excluding $a$ and $\alpha$ this is the standard formula for an M/M/1 queuing model. The correction terms $a$ and $\alpha$ are to take into account the increase in utilization due to the routing of the new transaction. Two cases are analyzed. If the incoming class A transaction is run locally, $\alpha = (R_{CPU} + R_{CPU}/(1 + \rho_i))/R_{CPU}$ (using the same notation as in the previous section) and $a = 0$. Similar equations are used if the transaction is routed to the central site. The probabilities of contention are estimated from the number of locks held. For instance, $P_{LOC \times LOC} = n_{lock}/n_{lock}$. Similarly, the other contention probabilities are estimated and are then used to estimate the abort probabilities. Once these are computed, the response time is estimated using the equations derived in the previous section. Finally, the transaction is routed to the site (local or central) which results in a smaller estimated average response time.

(b) Response Time Based on Number of Transactions in System Strategy

The data used by this scheme is the number of transactions $n_i$ at the ith local system, $n_c$ at the central system, and the other data used in the previous scheme. From this we estimate the utilization factors, and then proceed as in the previous case. The local utilization is estimated as $\rho_i = \alpha (n_i + a)$, and the central utilization $\rho_c = \alpha (n_c + a)$. Excluding $a$ and $\alpha$ this is the fraction of time each transaction spends at the CPU, times the number of transactions at the system. The correction terms $a$ and $\alpha$ are to take into account the increase in utilization due to the routing of the new transaction. Two cases are analyzed. If the incoming class A transaction is run locally (remotely) $\alpha = 1$ (0), and $a = (0)$. The probabilities of contention are estimated from the number of locks held. For instance, $P_{LOC \times LOC} = n_{lock}/lockspace$. Similarly, the other contention probabilities are estimated, and are then used to estimate the abort probabilities. Once these are computed, the response time is estimated using the equations derived in the previous section. Finally, the transaction is routed to the site (local or central) which results in a smaller estimated response time.

3.2.2 Minimization of the Average Transaction Response Time

Two schemes similar to the above two cases are used here, and an identical method is used to estimate the average response times of local $R_{CPU}$ and central transactions $R_{CPU}$ where $i$ is used to distinguish each of two cases: (1) running the transaction locally or (2) shipping the transaction to the central site. Instead of comparing the response time of the incoming transaction for the two cases, we estimate the average response time of the the currently running transactions for the cases (1) and (2) respectively, as

\[
\frac{(n_i + 1) \times R_{CPU}^L + n_c \times R_{CPU}^C}{n_i + n_c + 1}
\]

and

\[
\frac{(n_c + 1) \times R_{CPU}^C + n_i \times R_{CPU}^L}{n_i + n_c + 1}.
\]

In this expression, $R_{CPU}^L$ and $R_{CPU}^C$ are the estimated local (shipped) response times for cases (1) and (2) above; $n_i$ and $n_c$ are the number of transactions at the ith local site (at which the incoming transaction arrives) and the central site respectively. Finally, the transaction is routed according to the case that results in the smaller estimated average response time.

3.2.3 Heuristic Based on Measured Response Time

In this heuristic, the measured response time of the last class A transaction that is shipped and of that retained local are captured. If the current value of the response time of the last shipped transaction is less than that retained local, then the next incoming transaction is shipped, and vice versa. In this manner, the heuristic attempts to keep the response time of the shipped and local transaction comparable.

3.2.4 Heuristic Based on Queue Length

This heuristic bases the routing decision solely on the measured queue length at the local and central sites. In the basic scheme, a transaction is shipped if the measured central queue length is less than the local queue length. The scheme is then extended to base the routing decision on utilization levels estimated from the queue length. From the queue length, the utilization factors at the central and local systems are estimated using a similar equation to that used in 3.2.1. Since the response time is not estimated, the current utilization excluding the new transaction is estimated. If the estimated local utilization is more than the estimated central utilization by some threshold, then the incoming transaction is shipped, otherwise it is retained local.

4. Simulation Study and Performance Comparisons

In this section we present simulation results for the load sharing strategies discussed earlier, and compare the efficacy of the approaches.

4.1 Simulation

The load sharing strategies were simulated using a discrete event simulation. Simulation results are presented (unless otherwise specified) for a 10 site distributed system, with each local site connected to the central site through a link with 0.2 second communications delay; the central CPU has 15 MIPS while each local site has 1 MIPS. Transactions arrivals are Poisson processes, with the same arrival rate at each distributed site. The probability of local transactions is chosen as 0.75. Overheads and pathlengths are the same as described in Section 3. In the simulation, a global lock space of 32K elements (lockspace) is used. For local transactions, each local site makes lock requests uniformly over one tenth of the lock space, while central transactions make lock requests uniformly over the entire lock space. In analyzing this case the constant C of Section 3.2.3, is estimated as $C = n_i/lockspace$. The simulation maintains lock tables and explicitly simulates lock contention, and waits for locked entities, queuing and processing at the CPU, communications delay and overhead, I/O waits, aborts of central and/or local transactions, and commit processing. The CPU service times correspond to the time to execute the specific instruction pathlengths given in Section 3 (and are not exponentially distributed). The CPU is released by a transaction when lock contention occurs, for each I/O, and for the communication to another
In the case of a contention that leads into a deadlock the transaction is aborted and all locks held are released.

4.2. Performance Comparisons

We compare the protocols for the parameters given above. Figures 4.1 and 4.2 show the average (class A and B) transaction response time versus the total throughput. First note from Figure 4.1 that without any load sharing, the local systems quickly become overloaded with increasing transaction rate and the maximum transaction rate supportable is limited to about 20 transactions per second. Statique load sharing has a significantly better response time than no load sharing, and allows about 30 transactions per second to be supported. The "best" dynamic scheme is explained below. From Figure 4.2 we observe that the dynamic load sharing heuristic based on measured response time (marked as curve A in Figure 4.2) allows a larger transaction rate to be supported than no load sharing, but does worse than the static load sharing and the other dynamic load sharing schemes. A possible explanation for this is that the routing decision is based on the previously shipped transaction response time, and not on the current state of the system. The heuristic based on queue length (curve B in Figure 4.2) is a little worse than the static load sharing scheme. In the simulation to obtain this plot, a transaction was shipped to the central system if the local queue length was larger than the central queue length. Note that in all the dynamic schemes, except the ideal case described below, the information of the queue length at the central site is delayed, and is only updated during authentication of a centrally running transaction.

The queue length heuristic does not take into account the contention level, communication delay, or the faster processor at the central site. Part of the reason why it does well is that some of these factors cancel each other out. In particular, the communications delay increases the response time of shipped transactions, but the faster central CPU reduces the response time. The strategy does capture some of the effect of aborts and reruns, because these lead to larger CPU utilization and longer queues. Thus, if aborts at the central site increase, the queue length increases, and the strategy will ship fewer transactions to the central site. The dynamic schemes that attempt to minimize the incoming transaction response time, based on CPU queue length (curve C in Figure 4.2) and number in system (curve D), are a little better than the static scheme, with the latter scheme performing slightly better than the former. This is because the schemes attempt to predict the behaviour of the transaction taking into account the contention level, abort probabilities, communications overheads, commit and processing times. These schemes result in better response time at high loads, and allow a larger transaction rate to be supported. A possible reason that the scheme based on the total number of transactions in the systems does better is that it captures more information, including the effect of transactions that are in I/O and contention wait. Finally, the two dynamic schemes (curve D derived from queue length information, and curve F derived from number in system) that attempt to minimize the average response time perform better than their counterparts that attempt to minimize the incoming transaction response time. This is because these schemes attempt to take into account the effect of the routing decision on other concurrently running transactions at the local and central site.

Figure 4.3 shows the fraction of shipped class A transactions versus the total transaction rate for the various schemes. The static load sharing ships no transactions for small transaction rates (less than 5 transactions per second), then begins to ship an increasing fraction of transactions up to a rate of 25 transactions per second, beyond the fraction of shipped transactions decreases gradually. The reason is that for small transaction rates, the local utilization is small and there is no advantage in shipping, while at large transaction rates, the central site starts to become overloaded. The dynamic scheme based on measured response time tends to ship a larger fraction of the incoming class A transactions than the other schemes. The other dynamic schemes ship an increasing fraction of transactions with increasing transaction rate.
up to close to the maximum supportable transaction rate, and then ship a smaller fraction, as for the static case. Notice that the dynamic schemes ship a smaller fraction than the static scheme, except at very small transaction rates. Since the response time using (some of the) dynamic schemes is better than that of the static scheme, the dynamic scheme must ship transactions at the correct moment, as compared to the static scheme that ships transactions probabilistically.

In Figure 4.4 we examine a method of improving the performance of the heuristic scheme based on queue length. This is based on the fact that the MIPS at the central and local sites are different, and executing a transaction at the central site incurs a communications delay. Thus, shipping the transaction to the system with the shortest queue would attempt to balance the load but need not result in the smaller response time. In the modified heuristic, the utilization is computed from the queue length (as in Section 3), and the transaction is shipped to the central site if the estimated local utilization is more than the central utilization by some threshold. In cases A, B, C, and D of Figure 4.4, the threshold is 0, -0.1, -0.3, and -0.2 respectively. (Note that the negative sign means that the local utilization is smaller than the central utilization.) The best performance is obtained for a threshold of about -0.2. For smaller values of the threshold, (e.g. -0.3) the performance gets worse. The reason is that the smaller processing time of the larger central processor dominates the effect of the communications delay (for this value of 0.2 sec communications delay). The optimum value of the threshold is a function of communications delay and the specific values of the MIPS at the local and central sites, and on the number of local systems. We observe that a closer look at the simulation results indicated that the difference in utilization factor obtained by the dynamic schemes based on analysis, is close to the optimum value of the threshold above. Figure 4.4 also indicates the characteristic obtained by the (best) dynamic load sharing strategy. Notice that the performance of the best dynamic loading scheme is a better than the tuned heuristic based on queue length (with threshold of -0.2).

Next, we examine the effect of a larger communications delay. Figures 4.5 through 4.7 are similar to the above except that a communications delay of 0.5 seconds is used. In this case we find that the benefit due to static load sharing is much smaller than that for smaller communications delay. However, dynamic load sharing continues to offer significant improvement in the response time and the maximum transaction rate that can be supported. Notice in Figure 4.6, that for static load balancing, the graph of the the fraction of class A transactions shipped versus transaction rate has a point of inflection. For small transaction rates, the larger communications delay leads to a large penalty for shipped transactions, resulting in a small shipping
fraction. The fraction shipped increases rapidly when the local site begins to get overloaded. Finally, the fraction shipped begins to saturate as the central site begins to become overloaded. The optimal thresholds for the queue length based heuristic is different from the previous case. In Figure 4.7, the threshold for cases A, B, and C are 0, 0.2, and 0.1 respectively. The optimal value of the threshold in this case is about -0.1 versus -0.2 for the previous case. This is because the larger communications delay puts a penalty on centrally run transactions, even though the central MIPS are larger. Hence the threshold in the queue length base heuristic must be carefully chosen based on the system parameters. The figure also shows the characteristic of the best dynamic strategy based on minimization of the average transaction response time. Notice by comparison to Figure 4.4 that the difference between the best dynamic strategy and the tuned queue length based scheme is more significant for the larger communications delay.

5. Conclusions

In this paper we examined the problem of load sharing in a hybrid system consisting of distributed computing systems connect to a central computing complex. We considered two types of transactions, class A transactions that can be run at either the local or the central site, and class B transactions that must be run at the central site. Typically, class A transactions are a larger fraction. We examined various schemes of deciding whether to run class A transactions at either the local or central site. Several factors enter into this decision, including the transaction rate, MIPS at the central and local site, communications delay to the central site, data contention and abort probabilities, commit protocol overhead and delay, and the current state of the system. First, we looked at static load sharing, assuming knowledge of the transaction rates, and found significant improvement in response time and the transaction rate that could be supported over that without load sharing, particularly for low communications delays. Next, we examined five dynamic load sharing strategies: a heuristic based on the measured queue lengths, two schemes that attempt to minimize the incoming transaction response time based on estimates of the response time from (a) CPU queue length, and (b) number of transactions at the local and central system, and two schemes that attempt to minimize the average response time of all running transactions when a new transaction arrives. We found that all the dynamic schemes can improve performance over optimal static load sharing. The overall performance of the schemes based on analytical estimates of the effect of routing on all transactions in the system, rather than that on the incoming transaction alone, were found to be the best. We examined tuning the queue length based heuristic so as to ship transactions to the central site if the difference in estimated utilization at the local and central sites is more than some (positive or negative) threshold. The optimal value of this threshold was found to depend on the communications delay, MIPS at local and central site, fraction of local transactions, and number of local systems. In particular, for large communications delay, a larger (positive) threshold was necessary, while for small communications delays, a small (negative) threshold was necessary since the processing time is smaller at the central site (due to its larger MIPS). The difference between the best dynamic strategy and the tuned queue length based scheme was found to be more significant for a larger communications delay.

Acknowledgement

We would like to thank Alberto Avritzer for his contribution in helping develop the simulation program for the hybrid system, and for his work in the early stages of this study.

References


