A Framework for Workload Allocation in Distributed Transaction Processing Systems

Erhard Rahm
Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany

Ever-increasing demands for high transaction rates, limitations of high-end processors, high availability, and modular growth considerations are all driving forces toward distributed architectures for transaction processing. However, a prerequisite to taking advantage of the capacity of a distributed transaction processing system is an effective strategy for workload allocation. The distribution of the workload should not only achieve load balancing, but also support an efficient transaction processing with a minimum of intersystem communication. To this end, adaptive schemes for transaction routing have to be employed that are highly responsive to workload fluctuations and configuration changes. Adaptive allocation schemes are also important for simplifying system administration, which is a major problem in distributed transaction processing systems. In this article we develop a taxonomic framework for workload allocation, in particular, transaction routing, in distributed transaction processing systems. This framework considers the influence of the underlying system architecture (e.g., shared nothing, shared disk) and transaction execution model as well as the major dependencies between workload, program, and data allocation. The main part of the framework covers structural (or architectural) and implementational alternatives for transaction routing to help identify key factors and basic tradeoffs in the design of appropriate allocation schemes. Finally, we show how existing schemes fit our taxonomy. The framework substantially facilitates a comparison of the different schemes and can guide the development of new, more effective protocols.

1. INTRODUCTION

Transaction processing (TP) systems provide online access to a shared data base for many concurrent users. They are used in a variety of business applications such as airline reservations, electronic banking, securities trading, and communicating switching to enable the online user to execute preplanned functions (canned transactions). These functions are implemented by transaction programs that access the data base. The essential software components of a TP system are the set of transaction programs, the data base management system (DBMS), and the so-called TP monitor. The TP monitor controls the execution of transaction programs and supports their interaction with the terminal and the DBMS. The TP system provides recovery and concurrency control functions guaranteeing that transactions are either completely executed or not at all (atomicity), that modifications of successful (committed) transactions survive system and media failures, and that users see a consistent state of the data base despite concurrent access by other users.

The workload of TP systems is commonly dominated by short interactive transactions that belong to a limited number of transaction types (e.g., the debit-credit transaction type [1]). A transaction request by a terminal operator is first processed by the TP monitor, which analyzes the corresponding input message, performs authorization functions, and starts the appropriate transaction program upon availability of resources. During execution of the transaction program, multiple data base operations are typically submitted to and processed by the DBMS. At the end of the transaction, a commit protocol guarantees repeatability of the transaction's updates (e.g., by writing redo log information to nonvolatile storage) before a response message is returned to the terminal. The typical resource consumption of such a transaction ranges between 50,000 and a few million machine instructions, up to 30 disk I/Os and 2-20 messages (depending on whether the application is distributed). Currently, the largest TP systems support more than 100,000 terminals and 1,000 disks, and process thousands of transactions per second [2].

Address correspondence to Dr. Erhard Rahm, Dept. of Computer Science, University of Kaiserslautern, P.O. Box 3049, D-6750 Kaiserslautern, Germany.

0164-1212/92/$5.00

The traditional approach to TP systems is the use of large mainframe computers with a centralized DBMS handling the common data base. However, the need for ever-increasing transaction rates, high availability, and modular expandability (horizontal growth) has rendered this approach inappropriate for many applications and resulted in the development of distributed TP systems. These distributed systems are based on a variety of different architectures, which will be classified in section 3. A common problem of distributed TP systems is how to effectively use all nodes of the system to achieve high transaction rates and short response times. This problem is closely related to the question of which strategy should be used for workload allocation, i.e., for the allocation of the transaction workload among the processing nodes. Ideally, the workload is assigned in such a way that transactions can be processed with a minimum of communication and I/O operations and little data contention (e.g., lock conflicts) and resource contention (e.g., CPU waits).

The major form of workload allocation in distributed TP systems is transaction routing, which determines at which system (node) an incoming transaction request is to be processed (i.e., where the corresponding transaction program should be started). Other types of workload allocation deal with the assignment of smaller units of work than transactions (e.g., data base operations; see section 3). Transaction routing in existing distributed TP systems is not well advanced, but relies heavily on static assignments of terminals and programs to nodes and manual interactions by the system administrator. Such approaches prevent effective use of the system and make administration more and more complex as the number of nodes, terminals, and programs increases. What is needed are automatic, self-tuning schemes for workload management, particularly for transaction routing, that support efficient transaction processing. Unfortunately, this task is much more complicated than load balancing, which has been the focus of related research in other distributed systems [3-6]. In contrast to typical assumptions in these studies, we have to consider a large number of different workload (transaction) types with different resource and performance requirements (throughput, response times). Furthermore, transaction execution not only requires CPU resources, but causes I/O and communication overhead and delays and is subject to data contention (lock conflicts, aborts, etc.). To take these factors into account, appropriate routing schemes must also consider the reference behavior of transaction types against the data bases.

The development of "good" routing schemes is difficult and complicated by many design alternatives and dependencies on the underlying system architecture. To guide the selection of promising routing schemes and permit a (qualitative) evaluation and comparison of different approaches, we have developed a general framework for transaction routing that takes the most important system dependencies and design criteria into account. The next section presents a more detailed discussion of the requirements for appropriate routing schemes and provides an overview of our framework. Section 3 classifies the major approaches for distributed transaction processing and discusses their implications for workload allocation. In addition, the interrelationships between workload, programs, and data allocation are outlined. The individual classification criteria and design alternatives for transaction routing are then described in the two subsequent sections. Finally, we outline and evaluate sample routing schemes in section 6.

2. REQUIREMENTS AND FRAMEWORK OVERVIEW

Appropriate schemes for transaction routing should be effective, efficient, automatic, adaptive, and stable:

- Effectiveness is the most important requirement. This demands routing strategies that improve the performance of the transaction processing systems, e.g., compared to static load assignments or a random distribution of the workload. Typically, the system should support high transaction rates while satisfying specified response time limits (a possible restriction could demand that 95% of type X transactions have subsecond response times).
- Efficiency means that the overhead for drawing routing decisions, getting the required information, or assigning transaction requests to the destination node should be small compared to the potential performance benefits. In general, there is a tradeoff between effectiveness and efficiency, since to improve effectiveness beyond a certain point, more complex algorithms requiring more information collection may have to be used.
- Automatic and adaptive schemes are required to reduce dependency on system personnel (simplify administration) and permit fast reactions to changed conditions in the load profile or system configuration. Adaptive schemes require the continuous monitoring of the system to gather dynamic information. This information has to be analyzed (e.g., periodically) to detect conditions that require corrective actions (performance problems, significant changes in the reference behavior, node failure, etc.). In part, monitoring and corrective actions may have to be

carried out by local load control components, which should be coordinated with the workload allocation strategy.

- Stability of the routing scheme under contingent conditions such as overload or after partial failures is another consideration for appropriate algorithms. This requirement is particularly important for adaptive schemes to avoid overreaction to minor system and workload changes [7]. In addition, it must be detected when automatic corrections do not prove useful so that permanent problems (e.g., massive lock contention on data base hot spots, insufficient hardware resources, etc.) are reported to the operator.

To be effective, a routing scheme should support transaction processing with a minimum of overhead and deactivations due to intersystem communication, I/Os, data contention (aborts, lock waits), or physical resource contention (CPU waits, paging, etc.). Of course, these points are influenced by many factors, such as hardware environment (number and capacity of processors, main memory sizes, disk subsystem, communication system, etc.), system architecture, quality of system and application software, data base design, administration, or load characteristics. Still, there are two basic measures for workload allocation, affinity-based routing and load balancing, which we consider essential for high-performance transaction processing.

Affinity-based transaction routing. Affinity-based routing uses information about the transactions' reference behavior to route transactions with an affinity to the same data base portions to the same node. In this way, it strives to achieve what we call node-specific locality of reference, which requires that transactions running at different nodes should mainly access disjoint portions of the data base. Thus, affinity-based routing can substantially improve performance for transaction processing.

- Node-specific locality can significantly reduce the frequency of internode communication by assigning transactions to a node where they are largely locally executed. This aspect depends on various factors, e.g., the underlying system architecture, and is discussed further in section 3. Reduced internode communication improves throughput (less overhead) and response times. Shorter response times also reduce data contention (lock-holding times).
- Improved locality of reference can be used by the data base buffer manager to reduce I/O overhead and delays (better hit ratios).
- Most concurrency control conflicts occur between transactions running on the same node. These local conflicts can be resolved faster than global conflicts (e.g., shorter lock waits).

Since it is not generally feasible to predict the data base references of individual transactions, affinity-based routing is based on transaction types or workload groups for which homogeneity is assumed (i.e., transactions of the same workload group exhibit similar reference characteristics and resource requirements). For canned transactions, information on the reference behavior and resource requirements is generally available a priori or can be collected via monitoring. Node-specific locality is hard to obtain if the references of a transaction type are spread over the entire data base, if some data base areas are referenced by most transactions, or if there are dominant transaction types that cannot be processed by a single node without overloading it. Additionally, the more nodes to be used, the less node-specific locality may be achievable.

Load balancing. Load balancing is the second key objective of effective routing strategies. It strives to find a load allocation that avoids overloading individual nodes, which would cause excessive queuing delays for the bottleneck resources (CPU, main memory, etc.). This is particularly difficult during peak load periods, when all nodes must be highly utilized to sustain the required transaction rate, but without coming into the thrashing region of resource contention.

While other studies of workload allocation in distributed systems [3-6] have concentrated solely on load balancing, in general we have to support load balancing as well as affinity-based routing. This is because transaction response times are often determined more by data base-related factors like I/O delays, lock waits, or remote requests than CPU waits or paging delays. Unfortunately, supporting node-specific locality and balancing the load are often contradictory goals (e.g., a dominant transaction type may have to be assigned to multiple nodes), making workload allocation additionally difficult.

Classification Criteria

Finding a strategy for transaction routing that satisfies the above requirements is difficult and is influenced by many factors. We present a framework for workload allocation in distributed TP systems that attempts to clarify the most important dependencies and illustrate the spectrum of possible solutions.

Figure 1 provides an overview of our classification criteria. We separate the schemes into two categories, namely affinity-based routing strategies and load-

balancing schemes. Load-balancing schemes only aim at balancing the load; most strategies proposed for general distributed systems (not considering transaction processing/data base aspects) belong to this class. Affinity-based approaches, on the other hand, strive to achieve load balancing as well as affinity-based routing by considering the reference behavior of transactions and other dependencies. The next four criteria in Figure 1 refer to implications for workload allocation from the underlying system architecture, the allocation of transaction programs and the data base, and the transaction execution model. These dependencies will be outlined in section 3. The remaining classification criteria deal with structural and implementational aspects and alternatives for transaction routing and will be discussed in sections 4 and 5, respectively. Naturally, some of our classification criteria overlap with the taxonomies of load-balancing schemes [3, 5]. Furthermore, transaction routing has similarities to message (packet) routing in communication networks, where the goal is to minimize packet transfer times and maximize network throughput. Tanenbaum [8] distinguishes between nonadaptive (static) and adaptive schemes as well as between centralized, isolated, and distributed policies for network routing.

3. INFLUENCE OF SYSTEM ARCHITECTURE

This section discusses the impact of different architectures for distributed transaction processing on workload allocation. In section 3.1, we present a brief classification of distributed TP systems and discuss the role of workload allocation in the various approaches. In section 3.2, we outline the relationships among workload, program, and data allocation. Section 3.3 discusses the influence of the transaction execution model and compares the data partitioning and data sharing approaches with respect to workload allocation.

1) Affinity-based routing schemes vs.load balancing schemes

2) System architecture and execution model
   * System architecture (shared disk, shared nothing, etc.)
   * Program allocation (distribution, partial replication, full replication)
   * Data allocation (partitioning, partial replication, full replication)
   * Transaction execution model

3) Structural aspects for transaction routing schemes
   * Location of the global scheduler (front-end or back-end approach)
   * Centralized or decentralized organization
   * Isolated or cooperative approach
   * Message-based or storage-based communication
   * Source- or server-initiated routing
   * Preemptive or non-preemptive transaction assignment

4) Implementation aspects for transaction routing
   * Static, dynamic or semidynamic schemes
   * Adaptive or non-adaptive
   * Individual or (workload) group-based routing
   * Deterministic or non-deterministic
   * Solution method for determining the destination processor
   * Amount of information used

Figure 1. Classification criteria for transaction routing schemes.

3.1 Classification of Distributed TP Systems

For a rough classification of distributed TP systems, we distinguish between homogeneous architectures, where every node has the same functionality with respect to transaction processing, and (functionally) heterogeneous architectures such as client-server models (Fig. 2). The first subclass is usually further subdivided into shared memory, shared disk (SD), and shared nothing (SN) architectures [9].

Shared memory (shared everything). This class refers to transaction systems running on a tightly coupled multiprocessor where all processors share a

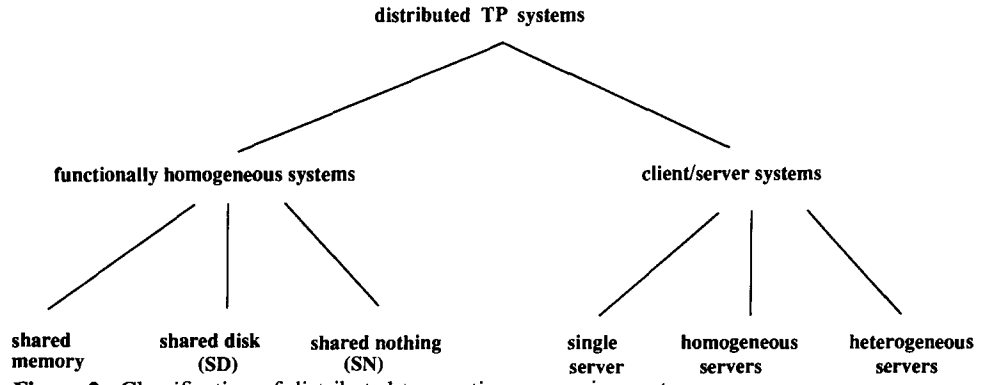

Figure 2. Classification of distributed transaction processing systems.

common main memory as well as peripheral devices (terminals, disks). Typically, there is only one copy of the application and system software, which is accessible to all processors via the shared memory. This approach is applicable to conventional (centralized) TP systems with few or no software changes. This approach has become increasingly popular for UNIX-based systems running on tightly coupled microprocessors (Sequent, Encore, Sequoia, Elxi, etc.).

Shared disk (data sharing). In data-sharing systems, the nodes are autonomous (no shared main memory; separate copies of operating system, TP monitor, DBMS, etc.) and share the common data base at the disk level. Every node in such a system may be a tightly coupled multiprocessor. Internode communication takes place by means of message passing (loose coupling) or via common semiconductor stores, which offer higher fault isolation than shared main memory and much faster access than disks (close coupling). Data (disk) sharing is only applicable if all nodes are in physical proximity. Existing SD systems include IBM's IMS data-sharing facility [10] and TPF [2] and DEC's DBMS for Vax Clusters [11].

Shared nothing (disk partitioning). SN systems also consist of a set of autonomous nodes which are loosely coupled, in general. Disks are partitioned among all nodes; the data base may be partitioned, too (data partitioning), or replicated. This approach is used in (geographically) distributed data base systems as well as in locally distributed TP systems. Fully or partially replicated data bases are of primary interest in geographically distributed systems to improve data availability and reduce the frequency of slow read operations at remote nodes. In local environments, the data partitioning approach (no replication) is more common. Existing SN systems include IBM's CICS [12] and Tandem's Encompass [13] and NonStop SQL [14].

The client/server approach is often used in workstation/server environments where the application programs are executed on workstations (or PCs) and data base services are provided by single or multiple homogeneous or heterogeneous server nodes. Data base machines can be considered as special data base servers where the clients (executing the application programs and submitting data base operations) are either workstations or general-purpose computers (hosts). Similarly, a distributed server may be based on any of the three homogeneous multisystem architectures above. Examples of client/server architectures for transaction processing include Camelot [15], Sybase, and SN data base machines like Teradata's DBC/1012 [16]. Remote data base access (RDA) is a SQL-based communication protocol currently under standardization, that allows a single program to submit data base operations to multiple heterogeneous servers with different hardware, operating systems, and DBMS [17].

For workload allocation, the SD and SN homogeneous approaches promise the highest flexibility. Workload allocation in shared memory architectures is usually handled by the operating system and thus is transparent to the TP system. In addition, SD and SN are more powerful than shared memory, since each node in these systems may be a tightly coupled multiprocessor. In client/server architectures, data base operations instead of transaction requests are usually submitted by the client (user, transaction program). However, this results in an increased number of work requests and thus increased communication overhead. Furthermore, some proposals for client/server cooperation (e.g., RDA) do not permit free selection of a server, but require the application programmer to specify the server(s) with which a connection should be established (fixed workload allocation, no location transparency). In the case of a single server, there is also no potential for workload allocation, since this node has to process all data base operations (potential bottleneck, limited expandability, single point of failure). With functionally specialized processors within a data base machine (e.g., sort processor, join processor, etc.), similar disadvantages exist unless these components are replicated.

In geographically distributed SN systems, terminals are usually directly assigned to the physically closest node to limit the communication delay for transaction requests. Although this fixed transaction allocation may be justified by strong geographic locality of reference, it does not guarantee high performance because load balancing may be poor (e.g., some nodes may be overloaded while others are idle). Instead of assigning a transaction to an already loaded system, it could be more effective to route it to a different node even it its execution would require more communication there.

Load balancing is generally more difficult to achieve in a geographically distributed system than in a local system. This is because communication overhead and delays are very high for wide-area networks, making it impossible to keep (almost) up-to-date information on resource utilization of remote nodes. In locally distributed systems with a high-speed interconnect, communication delays are almost negligible, so that routing decisions can be based on the current utilization of all nodes. In addition, slow communication is limited to the input from and response to the terminal, while remote requests during the execution of a transaction can be satisfied much faster than over a wide-area network. Furthermore, system administration is rela-

tively easy compared with geographically distributed systems.

Our discussion will therefore concentrate on locally coupled SD and SN systems offering the greatest potential for effective workload allocation. To compare these two approaches, we discuss the influence of program and data allocation on workload allocation for both architectures. In addition, differences in the transaction execution model and their implications for workload allocation are analyzed in section 3.3.

3.2 Influence of Program and Data Allocation

In contrast to centralized systems, distributed TP systems based on SD or SN architecture have to solve three allocation problems: workload, program, and data allocation (Figure 3). Workload allocation depends on program allocation, since a transaction should only be assigned to one of those nodes that can execute the corresponding application program. Data allocation is also important for workload allocation, since a transaction should preferably be assigned to that node where most of the required data is directly accessible (without communication delay). Thus, program and data allocation may limit the set of nodes where a transaction can or should be processed, resulting in a reduced potential for load balancing. As shown in Figure 4 and discussed below, the allocation of programs and data also depends on the system architecture (SD or SN).

Program allocation. Transaction programs may be allocated to nodes in one of the following ways:

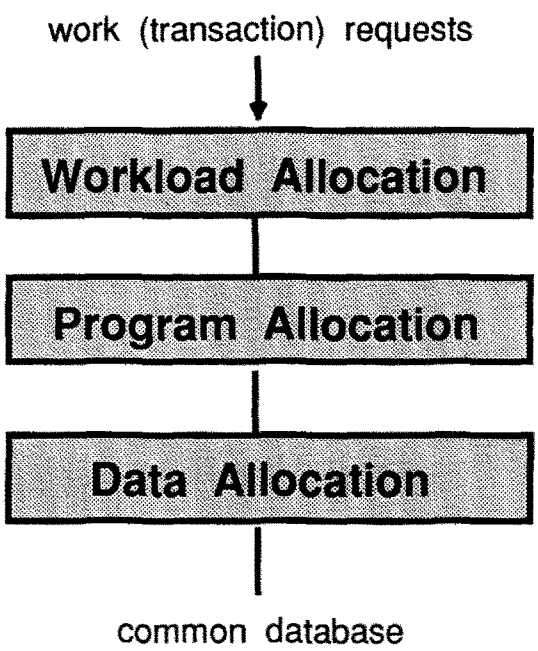

Figure 3. Allocation problems in distributed transaction processing systems.

- distribution
- partial replication
- full replication

Program distribution means that each application program (transaction type) is assigned to only one node. In this case, workload allocation is trivial and determined by the program allocation. A major problem of this approach is that dominant transaction types (e.g.,

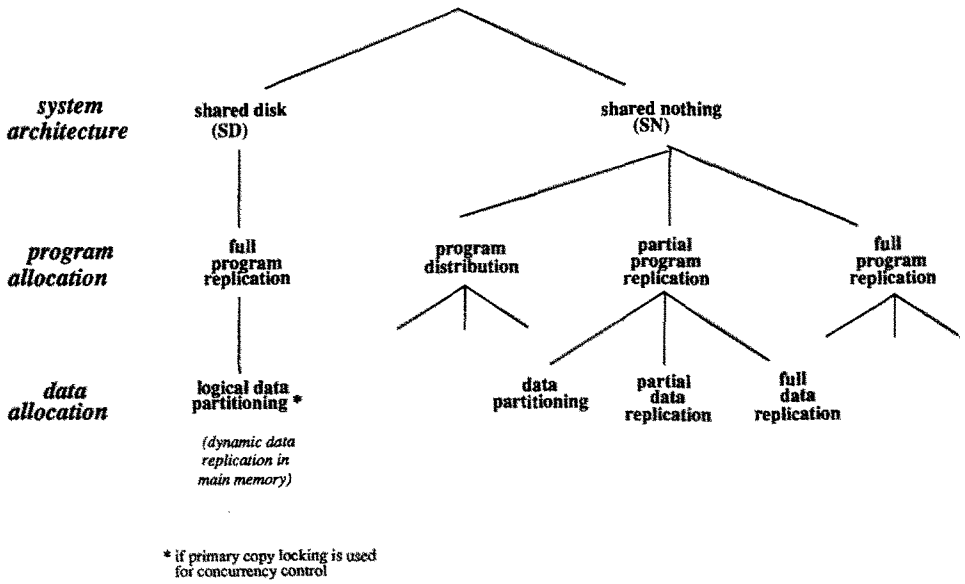

Figure 4. Program and data allocation in SD and SN systems.

debit-credit) can easily overload single nodes and thus prevent good utilization of all nodes. With full replication, each node can start all application programs. This leaves the greatest flexibility for workload allocation, since each node is eligible for assignment of any transaction request. With partial replication, some programs are replicated on multiple or all nodes.

Program allocations tend to be static, since relocations require manual interaction. Therefore, full program replication seems most promising from the perspective of workload allocation, since otherwise a (partially) static load distribution would be prescribed. Note that system software (DBMS, TP monitor) is also fully replicated in homogeneous architectures like SD and SN.

Full program replication is easily supported for data sharing by storing the programs on the shared disks. In SN systems, program replication requires the redundant storage of programs on the disks of different nodes and is thus more expensive than with SD. As a consequence, existing SN systems often apply partial program replication or program distribution, thus limiting the selection of destination nodes for automatic transaction assignment. Furthermore, administration becomes significantly more complicated since the program allocation (as well as the data allocation) has to be determined and adapted by system personnel.

Data allocation. In the case of data sharing, all nodes physically share the same disks and, thus, the data base. Data may be dynamically replicated in main memory due to caching of data base objects (which introduces the problem of buffer invalidations). Although the data base is physically shared, there may be a logical allocation of data base partitions with data sharing, e.g., for concurrency control purposes. This is the case in the primary copy-locking protocol, where the lock authority is distributed among all nodes [18].

In SN systems, data base partitions and replicas reside on disks and therefore constitute physical assignments to nodes. In contrast to logical data allocations represented by internal control information (e.g., in the case of data sharing with primary copy locking), physical data allocations in SN systems are relatively static, since cables and large data base portions are typically hard to move. The data allocation has to be determined by the system administrator and must be coordinated with the program allocation so that programs are assigned to the nodes where the data base portions they access are locally accessible. SN with fully replicated data bases permits a local processing of all read-only data base operations and thus has a high potential for load balancing. On the other hand, this approach requires n-fold disk capacity and makes update operations very expensive. As pointed out above, locally coupled SN systems are therefore usually based on the data partitioning approach (no replication).

The flexibility of work distribution is also influenced by the units of data allocation, or fragments. If only coarse fragments are supported, e.g., entire files or record types, the reference distribution against data base partitions may be highly skewed, making it difficult to achieve load balancing (dominant data base files, etc.). Data partitioning for nonrelational data bases is usually restricted to such coarse fragments and requires that every data base operation must not spawn multiple partitions. Relational data bases permit smaller fragments with horizontal or vertical partitioning of relations [19]. Data sharing does not require a physical data allocation of partitions; a logical data allocation can be based on arbitrarily small fragments, even for nonrelational data bases (e.g., page or record ranges).

In the past, much research has been devoted to the data allocation problem in distributed systems (see the survey in [20]). The heuristics proposed in [21] use prior knowledge of query processing strategies to determine a suitable fragmentation and allocation of the data base.

3.3 Distributed Transaction Execution Schemes

The main form of load distribution we have discussed so far is transaction routing, i.e., the assignment of entire transactions to nodes. Though it would be desirable from a performance point of view to completely process a transaction locally after its assignment to a node (to avoid communication overload and delays), this cannot generally be achieved if resources (applications, data) are to be shared between nodes. However, a distributed transaction execution introduces additional and smaller load distribution units than transactions. Given that a TP system mainly consists of transaction programs, TP monitor, and DBMS, we have basically three levels with different distribution units where communication/cooperation can take place. At the program level, a distributed transaction execution is possible by remote procedure calls of external program fragments. The next finer distribution granule are data base operations which may be submitted to the DBMS on the same or a remote machine. This shipping of data base operations can be done by the TP monitor or the DBMS. Finer distribution units than data base operations (e.g., suboperations or lock requests) require cooperation between the DBMS of different nodes. In the first two cases (distributed programs and TP monitors), the underlying system architecture is basically

SN, while DBMS-to-DBMS communication assumes either SN or SD.

Program-to-program communication (distributed programs). In this case, a transaction program can invoke other transaction programs which may reside on different machines (remote procedure call). This mechanism is to be supported by the TP monitor (and DBMS) since it requires a distributed commit protocol at the transaction's end. Applications based on the TP monitors CICS and UTM can make use of this approach, however without location transparency for the application programmer. Tandem, on the other hand, supports this approach with full location transparency for the programmer. Its TP monitor Pathway keeps track of the location of called programs (servers) and routes requests automatically to a node where the called function is available [22].

In principle, this approach permits a distributed transaction execution without support by the DBMS if each program fragment (function) is only permitted to access local data. In this case, however, we would have a collection of centralized DBMS requiring a data base and application design with multiple data bases and independent schemas. Apparently, this approach is very inflexible since, data base relocations generally introduce schema and program modifications. Furthermore, the data base splitting results in a distribution (no replication) of programs, leaving no potential for a dynamic workload allocation.

Shipping of data base operations. This cooperation level is supported by the function request shipping facility of CICS, which is also referred to as DB call shipping. Data base operations are "shipped" by the TP monitor to a local or remote DBMS. In the case of a remote request, the remote TP monitor starts a mirror transaction and submits the operation to the DBMS component at its node. CICS provides location transparency to the application program by maintaining appropriate directories.

In contrast to pure program-to-program communication, function request shipping permits a transaction program to access remote data bases. On the other hand, the smaller distribution unit may result in an increased number of remote requests. Furthermore, there is still no communication between the DBMS components, so that each data base operation must be completely processed by one DBMS. This is a severe restriction for relational DBMS, where operations such as joins may have to process multiple relations (all relations that may be accessed by a single operation would have to be stored at the same node.)

DBMS-to-DBMS communication. This approach offers the highest flexibility for workload, program, and data allocation (and thus for distributed transaction processing), albeit at the expense of the highest implementational complexity. Application programming is considerably simplified compared with the previous approaches, since the programmer sees only one logical data base (one data base schema). Intersystem communication is handled by the DBMS and is thus transparent to the program. Also, there are no a priori restrictions for data, program, and workload allocation, although full program (and data) replication is more expensive for SN than for SD (see above).

With data partitioning, communication takes place if the processing of a data base operation requires access to the data partition of another node. A simple approach to get the data is I/O request shipping, where each external object is explicitly requested (and returned in the case of a modification). This strategy is generally limited to simple data base operations that require few data accesses. For more complex operations, an execution plan has to be generated that determines a distribution of suboperations (e.g., selection, projection, join, etc.) that tries to limit the number of remote requests as well as the amount of data to be transferred. However, the execution location of these (sub)operations generally is independent of where a transaction is routed to, but is determined by the physical data base allocation. If a data base operation is started on a node not owning the required data, no load balancing advantage can be gained and only overhead is created for shipping the operation to the data owning system and returning the results. Since every node has to process all operations of local and remote transactions against its partition, there is very little potential for load balancing with data partitioning. Thus, imbalanced system utilization and overloaded nodes are possible. On the other hand, the known data allocation makes affinity-based routing comparatively easy if the reference distribution of transaction types is known. In this case, transaction types are simply assigned to the node that controls the data partition to which most accesses are directed. This assignment is relatively static, given that the data allocation cannot be changed frequently and the reference distribution is also stable, and may even be achieved with program distribution.

With data sharing, the entire processing of data base operations is local (and does not require development of distributed execution plans), since all data is directly accessible. Communication is required for concurrency and coherence control (treatment of buffer invalidations). Our previous investigations have shown that the primary copy scheme permits a very efficient concurrency and coherence control for data sharing [23, 24].

It uses a logic partitioning of the data base such that each node is assigned the synchronization responsibility or primary copy authority (PCA) for one partition. By coordinating load and PCA allocation, a transaction can often be assigned to a node where most of its references can be locally synchronized. In this way, the primary copy scheme can use node-specific locality of reference to limit the number of remote lock requests. Coherence control can be fully integrated into the concurrency control protocol to avoid extra messages [18, 24].

The determination of the preferable destination node is as simple as for data partitioning if the reference distribution of transaction types and the current data (PCA) allocation are known. In contrast to the physical data allocation in data partitioning systems, however, the PCA allocation can be dynamically adapted together with the routing strategy, e.g., when the load profile changes significantly or the number of systems is changed. Also, compared to data partitioning, the primary copy scheme still preserves a high potential for load balancing since only lock-request processing is affected by the PCA allocation. However, the largest part of a transaction can be processed at the node to which the transaction has been routed. Thus the primary copy approach combines the load balancing advantages of data sharing with the ease of affinity-based routing of data partitioning.

Both architectures, data sharing and data partitioning, may employ parallel execution strategies for complex queries and assign independent (nonconflicting) suboperations to different nodes in parallel. With data partitioning, the physical data distribution determines whether a parallel execution is applicable for a given operation. This limitation is removed for data sharing; the same objects can be concurrently read in different nodes, thus increasing the flexibility for parallel processing models. Khosafian and Valduriez [25] discuss parallel execution strategies for SN systems with horizontally partitioned data bases. In [26], the pros and cons of SN and SD for parallel query processing are discussed in more detail.

Summary. The discussion in this section has shown that the homogeneous SN and SD architectures offer the highest potential for effective load distribution. Furthermore, only distributed TP systems with DBMS-to-DBMS communication support flexible load, program, and data allocations. Data sharing offers a higher potential for load balancing than data partitioning, since each program and data base operation can largely be locally processed (full program replication, directly accessible data base). System administration is also simplified, since no program or physical data allocation have to be determined and adapted. In addition, data sharing simplifies query processing (no distributed execution plans) and may offer increased flexibility for parallel execution strategies.

4. STRUCTURAL CLASSIFICATION ASPECTS FOR TRANSACTION ROUTING

Four alternatives characterize the physical architecture of the transaction routing component or global scheduler: front-end (FE) or back-end (BE) approach, centralized or decentralized realization, isolated or cooperative policy, and message- versus storage-based communication. The resulting 12 approaches (the differentiation between isolated and cooperative schemes is only applicable for decentralized routing) together with examples of existing implementations are shown in Figure 5. Two other organizational aspects are orthogonal to these alternatives and deal with the distinction between source- and server-initiated and preemptive and nonpreemptive load distribution.

4.1 Location of the Global Scheduler (BE vs. FE Approach)

The location of the global scheduler influences where the terminals are allocated. In the BE approach usually found in existing TP systems, terminals are directly attached to the transaction processing nodes. In this case, the global scheduler can be seen as part of every local (BE) TP monitor that determines to which system an incoming transaction request should be routed for processing. Typically, the output message for the terminal user has to be returned via the node to which the terminal is attached. The transfers of transaction requests and output messages between BE nodes can incur a substantial communication overhead.

The BE approach is employed by IMS MSC (multiple systems coupling) [27], CICS ISC (intersystem communication) [12], and Tandem. IMS MSC and CICS are based on a distribution of programs (no replication), so that the routing destination is determined by the program allocation. Tandem supports a partial program replication, although it is unclear from the available documents how the destination node is selected when more than one node can process a given transaction type. IMS MSC has the additional restriction that every transaction program may access only local data. CICS and Tandem are based on the data partitioning approach and support a shipping of data base operations (in the case of CICS "function request shipping") or suboperations (Tandem NonStop SQL) to access remote data.

The FE approach allows a dynamic allocation of terminals/transactions to transaction processing nodes

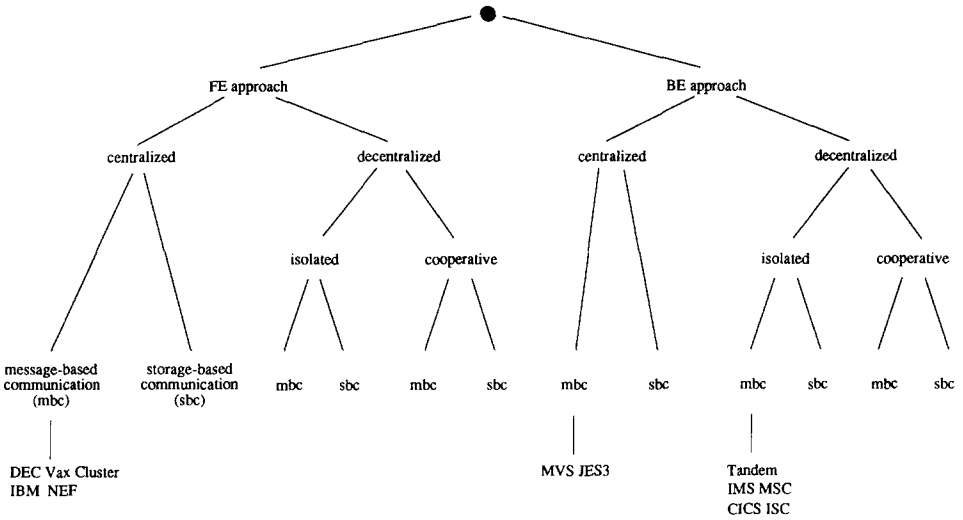

Figure 5. Workload allocation—structural aspects.

that considers the current load situation at the BE nodes. Here the terminals are basically attached to the FE system from where connections to any BE node can be established. The FE processors are typically communication controllers, which may provide optimized communication primitives (compared to the communication among BE nodes) to route transactions to the BE systems and to receive response messages. Another advantage of the FE approach is that message processing (routing overhead, logging of input messages, etc.) can be partially offloaded from the BE nodes to the FE nodes. Furthermore, failure of a BE node no longer requires that connections to a large number of terminals be reestablished.

Some existing TP systems already use the FE approach for transaction routing. For instance, a large U.S. bank is using IBM TPF (transaction processing facility) as a FE message switcher to multiple BE IMS systems [28]. The routing decisions are based on the account number found in the input messages. The IBM Network Extension Facility (NEF) for communication controllers supports transaction routing by user-selected criteria as well as fast session starts and restarts [28]. The DEC Vax Clusters have a special terminal server as a FE that tries to route user requests to the least utilized node to achieve load balancing [11].

4.2 Centralized vs. Decentralized Organization

With the BE approach, transaction routing is usually decentralized (distributed), since it makes little sense to allocate all terminals to a single transaction processing node. This would leave little capacity at this node for transaction processing and result in a high communication overhead for routing transactions to and receiving response messages from the other systems. IBM's JES3 [29], however, uses the centralized BE approach for allocating batch jobs, typically submitted by operators instead of terminal users. It supports workload allocation among up to eight nodes, including the global processor that controls load distribution and provides a single system image to the operator. Since batch jobs have to specify the resources (e.g., files) they want to access, JES3 can prevent scheduling jobs simultaneously that are going to update the same data.

Running the global transaction scheduler on a single node may be more appropriate with the FE approach, provided that no bottleneck is created. The main advantage of the centralized approach is simplicity: one instance controls the entire system (BE nodes, terminals, etc.), thus facilitating the provision of a single administration interface. In addition, it guarantees a single node image to the network [30]. While availability problems may be resolved by a (passive) standby system, a single FE system could become a bottleneck with growing transaction rates. This is especially a problem if the FE node is used not only for message switching but also for other tasks such as monitoring the BE systems and arrival rates or logging input messages.

4.3 Isolated vs. Cooperative Strategies

Decentralized routing schemes may be cooperative or isolated depending on whether the components of the global scheduler coordinate their routing decisions. In isolated decentralized routing schemes, there is no communication between the load distribution components for exchanging status information or adapting the

routing policy. This uncoordinated approach is used in existing TP systems such as IMS MSC, CICS ISC, and Tandem. Note, however, that in the case of a decentralized FE approach, status information may be exchanged between the transaction processing nodes and the FE nodes performing transaction routing. In addition, every node taking part in transaction routing may adapt its own routing strategy based on its local state information.

In cooperative decentralized schemes, there is a common routing policy and explicit cooperation to exchange status information or adapt the routing strategy. One coordination approach would be to store global control information in a shared memory segment accessible by all processors participating in transaction routing (close coupling). This may be more efficient than explicit message exchange and avoids the problems associated with using global information at different levels of actualization. Cooperation may also be facilitated by using a central coordinator for global control decisions in addition to the load distribution components [31]. The common routing policy is then determined and adapted by the global coordinator, which communicates with the load distribution components, e.g., to bring a new routing strategy into effect. The global control component can provide a single administration interface to the outside and cooperate with local load control components to get monitoring information and coordinate local and global scheduling decisions [31].

4.4 Message- vs. Storage-Based Communication

Communication for workload allocation purposes is needed for transferring input and output messages (assigning work requests to the destination node and returning results) and for exchanging control information. Control information may be exchanged between local and global load control components (e.g., monitoring data), or between global scheduler components in the case of cooperative decentralized transaction routing (e.g., to adapt the routing policy).

If all processors are loosely coupled, internode communication takes place by means of message passing, i.e., sending and receiving messages over communication lines. This approach does not rely on shared memories, and thus provides good failure isolation and is not susceptible to memory bottlenecks. On the other hand, message passing often causes a high CPU overhead—even in local systems—and global information needed in multiple systems has to be stored and updated redundantly, thus introducing additional messages. Ferguson et al. [4] distinguish between broadcast, polling, and diffusion schemes for propagating control information between processors.

The disadvantages of message passing can be overcome by a close coupling using shared (semiconductor) stores. Such a shared memory segment can be used for maintaining shared message queues, which allow an efficient propagation of messages (transaction requests, response messages) between systems. If the shared store is nonvolatile, logging of messages can also be achieved with little or no extra overhead. In a decentralized organization of the global scheduler, the shared store can be further used for keeping global state and routing information.

Of course, these uses require a store with appropriate access and synchronization primitives that provide better failure isolation than is typically found for shared main memory in tightly coupled multiprocessors. Also, access times of a few microseconds should be possible to enable synchronous memory accesses without process switches. Finally, the number of nodes that can be connected to the shared store(s) should be high enough. If, for instance, only four systems can be connected to the store, a distributed FE approach would be restricted to two FE and two BE nodes. An alternative in this case would be to use the common store only within the FE system (for holding global state and routing information), and relying on message passing for communication with the BE nodes.

4.5 Source- vs. Server-Initiated Routing

The distinction between source (sender)- and server (receiver)-initiated routing stems from work on global load-sharing schemes [5]. In our context, the source is that (FE or BE) processor to which the terminal issuing a transaction request is attached, while the server is the (BE) node where the transaction program is started. With the source-initiated strategy, usually found in existing transaction systems, the source node determines where a transaction is routed. The routing decision is typically made at transaction arrival time (immediate routing of a transaction) so that queues tend to form at the server nodes. In a server-initiated algorithm, on the other hand, the servers go looking for work at the source node(s), e.g., when they have completed another transaction or the available resource capacity allows for additional work. Accordingly, queues tend to form at the source node(s).

Shared message queues in a common store (storage-based communication) can be used by source- as well as server-initiated routing schemes. In the former case, there may be separate input and output queues for every node in the shared store. To assign a transaction, the source writes the input message to the (input) queue of the destination node. These input queues are either periodically checked by the transaction processing nodes

to pick up and schedule waiting requests, or the source sends a short notification interrupt to the server node indicating that a transaction request has been inserted in its input queue. Output messages are written to the (output) queue of the node, which has to send it to the terminal. In the BE approach for routing, a node may have an input as well as an output queue, whereas otherwise output messages are associated with FE nodes and input messages with BE nodes.

In the case of a server-initiated scheme, the source writes input messages either into a single input queue for all server nodes or into a specific queue if there are separate input queues for different transaction types or workload groups. The server nodes pick up requests from the input queue(s) when they are willing to process a new transaction.

The basic advantage of server-initiated routing is that overloaded nodes are easily avoided, thus giving advantages for load balancing. Furthermore, every server can directly use information on local conditions, e.g., data base buffer contents or lock ownerships, to pick a transaction request that can be processed with little I/O and communication overhead. On the other hand, there may be long waiting times for transactions at the source until a server is willing to process them. This could lead to "starvation" for some transactions, or at least to missed response time limits. In addition, selection of waiting transactions cannot be based solely on local conditions but must be globally coordinated by the global scheduler. In this respect, the source-initiated approach seems superior, though the server-initiated scheme can be extended to overcome its problems (e.g., by limiting the selection of waiting transactions according to global priorities or by forcing servers to process long-waiting transactions).

All system examples in Figure 5, except JES3, support a source-initiated workload allocation. JES3 is based on the server-initiated approach, where server nodes send job requests to the global processor. The global processor maintains job information for waiting batch jobs on a shared (spool) disk device and specifies in a response message to the requester which job from the spool queue can be processed [29].

4.6 Preemptive or Nonpreemptive Transaction Assignment

Another aspect of load distribution is whether migration of transactions is allowed, i.e., that a transaction can be rerouted from one processor to another in mid-execution for load balancing reasons. The distinction between migration (preemptive) and placement (nonpreemptive) strategies has been made in related work on load balancing [32]. Transaction migration seems difficult to realize since locks, data base pages, or terminal control blocks associated with the transaction would have to be moved, too. The migration overhead is not likely to pay off for short transactions, but may be acceptable for complex queries to improve load balancing. Currently, migration is not supported in distributed transaction and data base systems.

5. ADDITIONAL CLASSIFICATION ASPECTS

Six additional classification aspects coping with implementation alternatives and the used information complete our framework for transaction routing protocols. As shown in Figure 6, four of these implementation criteria allow us to separate ten families of algorithms for which examples from the literature and existing systems are specified. In addition to these four criteria, we discuss which solution methods are applicable to determine a transaction's destination node and which types of information can be used by different routing schemes. Individual algorithms will be discussed in section 6.

5.1 Static, Dynamic, or Semidynamic Routing Schemes

There is no general consensus about the definition of static and dynamic workload allocation schemes. A simple approach would be to consider a routing scheme as dynamic if it uses dynamic information for routing decisions, i.e., information about the current system state, and static if the current system state is not taken into account. For instance, Eager, Lazowska, and Zahorjan [33] consider a scheme as static if it uses only information about the average behavior of the system, ignoring the current state. Casavant and Kuhl [3] use the time at which the assignment decisions are made to separate static and dynamic schemes (predetermined destination vs. routing decisions at job arrival time). In this article, we will adhere to definitions in [4] that allow us to distinguish between three categories: static, dynamic, and semidynamic policies.

As in previous work, we consider a scheme as static if it does not consider the current system state for routing decisions but employs a predetermined allocation strategy. Such an approach is not capable of automatically adapting to changing situations but requires manual interactions to change the static load assignment. We separate routing policies that take the current system state into account into dynamic and semidynamic approaches. Dynamic schemes take the pessimistic view that the system state is constantly fluctuating and cannot be adequately estimated a priori. Therefore, they consider the current system state for

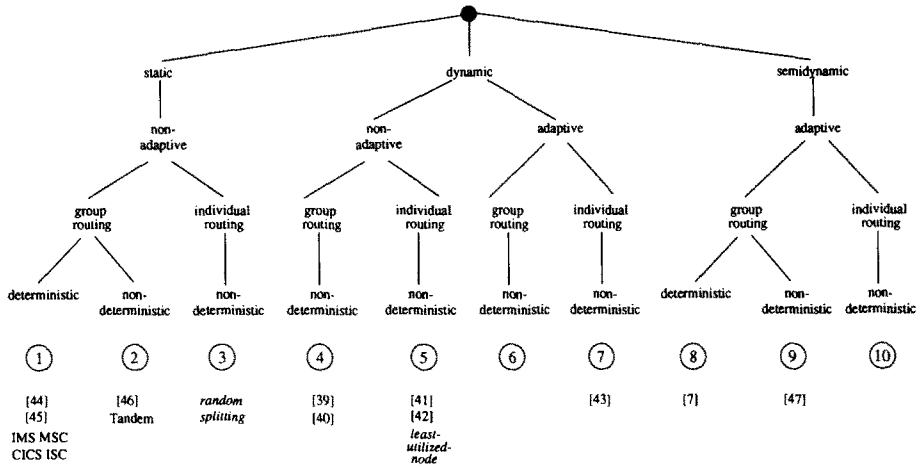

Figure 6. Workload allocation—implementational aspects.

every transaction assignment. Semidynamic policies are a compromise between static and dynamic schemes. They have a static phase in which a common routing policy is applied to all transactions, and a dynamic phase in which the current system state is considered to adapt the routing policy. In the dynamic phase, the current system state is examined and, if deemed necessary, the routing policy is changed. Typical reasons for such an adaptation are performance problems (e.g., missed response time limits) and changes in the load profile or configuration (e.g., node failure).

The main advantage of static schemes is efficiency (low overhead), since no dynamic information has to be gathered and analyzed. On the other hand, effectiveness may be poor since no automatic adaptation to changing conditions is possible. Administration is difficult since system personnel must determine and adapt the workload allocation. On the other hand, dynamic routing schemes may introduce significant overhead, depending on how much dynamic information is to be provided and analyzed. Semidynamic policies may thus be a reasonable compromise if there are stable phases where the load profile and system state do not change significantly.

5.2 Adaptive vs. Nonadaptive Policies

Changes in the load profile and system configuration (changed number of nodes, new applications) or tuning measures make adaptations of the workload allocation necessary. Adaptive workload allocation schemes are capable of an automatic adaptation to changing conditions [34], while nonadaptive schemes require manual interactions for any allocation change. By definition, semidynamic schemes are always adaptive, and static schemes are nonadaptive (Figure 6). Dynamic policies may be adaptive or nonadaptive depending on whether the policy itself is modified according to changed conditions. For instance, a strategy that always assigns a transaction to the least used node is dynamic (since it considers the current system state for every transaction), but not adaptive.

Adaptive policies require dynamic information, i.e., a continuous monitoring of the system state. To control the system, they typically apply feedback loops, which consist of analyzing the current system state, determining whether performance problems exist or are likely to occur in the near future (e.g., due to changes in the reference behavior), determining the underlying causes of problems (bottlenecks), and initiating corrective actions. To limit the control overhead, monitoring may be done on a sampling basis and the feedback loop may be executed only periodically or after special events such as node failures. The average time interval between successive executions of the control loop is a tuning parameter that determines the overhead as well as the responsiveness of the control scheme.

The main corrective action is an adaptation of the workload allocation policy by changing the algorithms and/or adjusting control parameters used to implement the routing policy (e.g., routing table) [32, 34]. With data sharing and primary copy locking, the data allocation can also be adapted automatically. Another option for data sharing (proposed in [35]) is to adapt the number of transaction processing nodes to current performance requirements (this is typically not possible for data partitioning since all data base partitions must stay accessible). This idea seems appealing since the maximal capacity is generally only needed during periods of peak transaction rates, so that available resources can be assigned to other applications (e.g., numeric com-

putations) during off-peak periods. If automatic corrections do not prove effective, the operator can be informed of the problem as a last resort.

5.3 Individual or (Workload) Group-Based Routing

Group-based routing schemes categorize all transactions into workload groups and use the same assignment policy for all transactions in the same group. Typically, a workload group consists of transactions with similar reference behavior and resource requirements that can thus be considered collectively. A transaction's workload group is derived from information associated with the transaction request, i.e., transaction type, input parameters, and terminal identification. This information allows a flexible definition of workload groups (by the administrator). For instance, a workload group can consist of

- all transactions of one or more transaction type
- a fraction of a dominant transaction type determined by a value range of input parameters (e.g., account ranges for debit-credit transactions)
- transactions from a specified set of terminals
- combinations of the above alternatives.

In the simplest case, all transactions of a workload group are assigned to the same node. If this is not possible for load-balancing reasons, a workload group can also be allocated to multiple nodes according to a specified rule (deterministic vs. nondeterministic schemes; see below). Table-driven routing schemes are an example of group-based allocation policies where a routing table determines the determination for every workload group. This approach is considered adaptive if the routing table can automatically adapt to changing conditions (e.g., relocation of a workload group from an overloaded node to a less utilized one).

Individual routing schemes do not aim at a common routing strategy for workload groups but treat transactions independently of one another. Since these individual routing decisions are usually more expensive than just a table look-up, they may introduce substantially more overhead than group-based schemes.

5.4 Deterministic vs. Nondeterministic Routing Schemes

For group-based routing schemes, this aspect separates the cases in which a workload group is assigned to exactly one node (deterministic) or multiple nodes (nondeterministic). Assignment to a single node is desirable to support node-specific locality of reference (affinity-based routing) and simplify the routing decisions. Unfortunately, this restriction leads to overused nodes if the processing requirements of a single workload group exceed the capacity of one node (dominant transaction type, unexpectedly high arrival rates, etc.). Therefore, load-balancing considerations often prescribe a nondeterministic scheme in which the destination node may be different for transactions of the same workload group. Probabilistic routing schemes are a typical example of nondeterministic policies in which the transaction assignment is controlled by some probabilistic distribution (e.g., 80% of group X transactions are routed to node 1, 20% to node 2).

Individual routing schemes are generally nondeterministic since the workload is to be distributed among multiple nodes.

5.5 Solution Method for Determining Processor Allocation

According to [3], we can here distinguish between optimal and suboptimal solutions, and in the latter case approximative and heuristic approaches. Optimal and approximative approaches formalize the transaction routing problem within a computational model and try to find solutions by one of the following methods [3]:

- queuing theory
- graph theory
- mathematical programming
- solution space enumeration and search.

Optimal schemes assume complete knowledge about the state of the system and resource needs and determine a processor allocation that optimizes a certain objective function such as minimal communication overhead or highest throughput. Their main disadvantages are the typically enormous computation overhead and the fact that they are necessarily based on a number of simplifying assumptions. Approximative schemes only try to find a good instead of the best solution in order to avoid searching the entire solution space. Although they are more efficient than optimal schemes, they generally also use an (over)simplified computational model of the transaction system. Heuristic approaches, on the other hand, depend to a lesser degree on the accuracy of the underlying models. They try to find a processor allocation even more efficiently than approximative solutions by avoiding computationally expensive methods and using some intuitive optimizations. For instance, clustering group of processes that interfere with or communicate heavily to the same processor [36] generally helps to decrease the communication overhead; even so, its effect on response times may not be directly predictable.

5.6 Amount of Information Used for Transaction Routing

The information which may be used for load distribution can be characterized along two lines:

Static versus dynamic information. None of the information relevant for transaction routing is completely static, but is subject to change. Nevertheless, for simplicity and efficiency, some information is assumed to be static or to change infrequently. Typical examples are the number of nodes and their respective CPU capacity or the number of transaction types (workload groups) and their average resource requirements. Dynamic information is collected while the load is being executed (via monitoring) and used for subsequent routing decisions. This kind of information may change quite frequently. Examples are the current CPU utilization, response time information per transaction type derived from recent executions, or information about which data base pages are currently held in the data base buffers. Static routing schemes use only static information, while dynamic and semidynamic schemes may use dynamic and static information. Use of static information complicates system administration since this information must generally be provided by the administrator.

Other information. This includes

- load characteristics (e.g., arrival rates, resource requirements, performance goals)
- data base characteristics (e.g., fragmentation)
- processor characteristics (e.g., number of nodes, resource capacity, utilization).

Some information refers to more than one of these three categories, e.g., the reference pattern (load and data base), the current distribution of transaction requests to nodes (load and processors), or the data/PCA allocation (data base and processors).

The reference pattern for different workload groups can be characterized by means of a reference matrix. Such a matrix shows for every workload group the frequency of object references with respect to different data base fragments over a characteristic period of time [7].

Resource (CPU, memory, etc.) utilization is dynamic and can be expressed by different load indices [37, 38]. Examples of load indices include CPU utilization as observed by a sampling monitor, the length of the ready queue or average CPU waiting times in the case of CPU resources, and the paging rate, paging delays, or hit ratios in the case of memory resources. Routing schemes that do not use information on resource utilization are sometimes referred to as load independent; otherwise they are considered load dependent. Note that all static schemes are load independent, and that dynamic and semidynamic schemes may or may not rely on utilization-based information.

Clearly, many quantities may be useful for workload allocation. However, practical routing schemes typically have to base their decisions on comparatively few parameters because complexity and overhead generally increase with the amount of information used, though more effective strategies may result. For the sake of efficiency, only selected quantities (e.g., CPU utilization) can be dynamically monitored and analyzed.

6. SAMPLE APPROACHES TO TRANSACTION ROUTING

The previous sections showed that the design of a routing scheme depends on several factors and that numerous design alternatives can be chosen. The presented framework permits us to classify and compare the various approaches used in existing transaction systems or proposed in the literature. In section 6.1, we briefly summarize the characteristics of some allocation schemes in existing transaction systems. The following two subsections discuss proposals from the literature for (pure) load balancing (6.2) and affinity-based routing (6.3) Due to space limitations, most of these schemes can only briefly be characterized here; however, in section 6.3, one class of algorithms (table-driven, affinity-based schemes) will be discussed in more detail. At the end of this section, we discuss some observations from a trace-driven simulation study on purely table-driven schemes and argue that dynamic, group-based schemes may be a better approach for transaction routing.

6.1 Workload Allocation in Existing TP Systems

Workload allocation in current distributed TP systems has already been characterized in previous sections. Using our classification criteria, we can summarize the characteristic features of the various systems as shown in Figure 7. The criteria are ordered according to Figure 1 (groups 1-4), although some criteria do not apply or their value cannot be determined from the available documentation. Point 1 (load balancing vs. affinity-based routing) has not been specified for static schemes that rely on manual assignments. In these systems, it is up to the administrator and his or her allocations whether load balancing or an affinity-based routing can be achieved.

A static terminal allocation with no further load distribution by the transaction processing (BE) nodes

DEC Vax-Clusters:
1) load balancing (send to least utilized node)
2) system architecture, execution model: SD with distributed locking protocol [11]
program allocation: full replication
3) centralized FE approach, message-based communication, source-initiated routing, non-preemptive
4) dynamic, non-adaptive, individual routing information used: CPU utilization

IMS/MSC:
2) system architecture: SN
program allocation: distribution (no replication)
data allocation: data partitioning (at database/file level)
execution model: local transaction execution
3) isolated decentralized BE approach, message-based communication, source-initiated routing, non-preemptive
4) static, non-adaptive (manual), group-based routing (transaction type), deterministic (destination node determined by program location)

CICS ISC (Intersystem Communication):
2) system architecture: SN
program allocation: distribution (no replication)
data allocation: data partitioning (at database/file level)
execution model: program-to-program communication or DB call shipping
3) - 4) as for IMS/MSC

Tandem:
2) system architecture: SN
program allocation: partial replication
data allocation: data partitioning (horizontal fragmentation of relations for NonStop SQL)
execution model: program-to-program and DBMS-to-DBMS communication (distribution of suboperations)
3) isolated decentralized BE approach, message-based communication, source-initiated routing, non-preemptive
4) static, non-adaptive (manual), group-based routing (transaction type), non-deterministic (program replication)

Figure 7. Characteristics of load allocation schemes in existing TP systems.

can also be classified with our framework. It embodies a static, nonadaptive, group-based, and deterministic approach in which the destination node is solely determined by the terminal identification. This allocation can easily be established with routing tables in communication controllers. An adaptive version would result if the terminals of a crashed node were automatically reassigned to one or more of the surviving processors.

6.2 Load-Balancing Schemes

Many load balancing schemes have been proposed in the literature. Two of them, random splitting [5] and "send to least utilized node" (according to some load index) are classified in Figure 6. These simple schemes are not affinity based and therefore not appropriate for transaction processing. They implicitly assume completely homogeneous workloads consisting of jobs (transactions) with similar resource requirements. They further assume that a transaction can be executed on any node with no communication, I/Os, or lock contention. Random splitting is a static scheme in which the destination node is determined at random according to a prespecified probability function.

Load-balancing schemes for replicated data bases are discussed in [39–41]. They consider only queries (read-only operations) to eliminate lock contention as well as the overhead for keeping all replicas up to date. Fully replicated data bases are assumed in [39, 41] so that every node can execute a query without any communication. The main objective in [39, 40] is to balance the number of CPU-bound and I/O-bound queries per node, assuming that queries can be grouped in one of these two categories before their execution. Thomasian [41] uses queuing models to estimate query response times and to find the node with the shortest execution time for a query.

6.3 Affinity-Based Routing Schemes

Affinity-based routing schemes aim at load balancing as well as supporting node-specific locality to reduce the amount of communication, I/O, or global lock conflicts (see section 2). It is most easily achievable for data partitioning or data sharing with primary copy locking. In both cases, a transaction is preferably assigned to that node controlling most of the data base portions the transactions is likely to access. This is basically a static decision (permitting a group-based or table-driven routing scheme) that only needs to be adapted if the data or PCA allocation or the reference pattern of transaction types change.

Before discussing the table-driven approaches, we briefly look at proposals for individual routing [42, 43]. They compare various dynamic routing schemes for data partitioning systems with a central FE node for load distribution. The key problem of these schemes appears to be the overhead associated with the determination of the destination node. For every incoming transaction, they calculate a response-time estimate for every node, and assign the transaction to the processor with the best estimate. These calculations are fairly complex (no cost analysis is provided), although a number of significant simplifications (e.g., no lock contention) are applied to keep the formulas analytically tractable. In [43], the nonadaptive algorithms from [42] are extended by feedback mechanisms to reduce the dependencies on accurate information about resource requirements.

Table-driven affinity-based routing schemes. The overhead of individual affinity-based routing can be significantly reduced by using a table-driven (group-based) workload allocation. On arrival of a transaction, the destination processor can simply be determined by a "lookup" in the routing table. On the other hand, the computation of the routing table itself may be expensive and typically requires extensive a priori information (or estimates based on prior measurement data), e.g., resource requirements or reference behavior. Typically, table-driven routing schemes are either semidynamic or static, depending on whether the computation method permits an automatic adaptation of the routing table to changing conditions. However, we will consider a group-based approach as dynamic if the destination node is not solely determined by a routing table (or equivalent data structure), but the current system state at transaction arrival time is also taken into account.

Several methods for calculating a routing table have already been proposed in the literature. In Figure 8, the characteristic features of these schemes are summarized according to the four groups outlined in Figure 1. They generally assume either a data partitioning system or data sharing with primary copy locking, since both approaches allow a similarly good prediction of the communication frequency (if the transaction reference behavior is known). The objective function in most of these schemes is to minimize the number of remote requests by means of an affinity-based routing while achieving roughly the same CPU utilization for all nodes. Main input parameters of the algorithms are the number of nodes, mean arrival rate per transaction type (workload group), CPU requirements per transaction type (average path length), and a reference matrix (or equivalent). In addition to the routing table, the schemes generally also determine a suitable data/PCA allocation. Recall, however, that a dynamic adaptation of the data/PCA allocation (together with the routing table) is only possible for data sharing, in general.

Cornell, Dias, and Yu [44] transform the quadratic optimization problem into a linear one and apply optimal solutions to it. To make this possible, a deterministic (nonprobabilistic) routing is necessitated in which every transaction type has to be assigned in its entirety to one node. Apart from this disadvantage, the computation method is too expensive to allow a dynamic adaptation of the routing table during transaction processing. This method has therefore been classified as static in Figures 6 and 8. In addition, their computation method determines the data allocation for which a frequent adaptation is not possible in data partitioning systems. The methodology of [44] has also been applied in [45], but for data sharing and a token ring–based

Cornell, Dias, and Yu [44]:
1) affinity-based routing
2) system architecture: SN
   program allocation: full replication
   data allocation: data partitioning (data allocation at database/file level)
   execution model: DB call shipping
3) centralized FE approach, message-based communication, source-initiated routing, non-preemptive
4) static, non-adaptive, group-based routing (transaction type), deterministic
   computation method: optimal solution, mathematical programming
   information used: number of nodes, reference matrix, CPU requirements

Yu, Cornell, Dias, and Iyer [45]:
1) affinity-based routing
2) SD with 'pass-the-buck' protocol for concurrency control, full program replication
3) - 4) as in [44]

Yu, Cornell, Dias, and Thomasian [46]:
1) affinity-based routing
2) system architecture: SN
   program allocation: full replication
   data allocation: data partitioning (data allocation at database/file level)
   execution model: DB call shipping or I/O request shipping
3) centralized FE approach, message-based communication, source-initiated routing, non-preemptive
4) static, non-adaptive, group-based routing (transaction type), non-deterministic (probabilistic)
   computation method: approximative solution, mathematical programming + iterative search
   information used: number of nodes, reference matrix, CPU requirements, arrival rates, CPU capacity, lock conflict probability, I/O delay

Reuter [7]:
1) affinity-based routing
2) SD with primary copy locking, full program replication
3) message-based communication, source-initiated routing, non-preemptive
4) semidynamic, group-based routing (transaction type), deterministic
   computation method: simple, two-step heuristic (routing table is determined in step 1, PCA allocation in step 2)
   information used: number of nodes, reference matrix, CPU requirements, arrival rates, CPU capacity

Rahm [47]:
1) affinity-based routing
2) data partitioning (SN) or (SD) with primary copy locking, full program replication
3) message-based communication, source-initiated routing, non-preemptive
4) semidynamic, group-based routing (transaction type), non-deterministic (probabilistic)
   computation method: iterative search heuristic (coordinated calculation of routing table and data/PCA allocation)
   information used: number of nodes, reference matrix, CPU requirements

Figure 8. Characteristics of some table-driven transaction routing schemes.

synchronization protocol ("pass-the-buck") used in IMS data sharing.

To achieve a probabilistic routing, an even more expensive, approximate determination of the routing table has been proposed [46]. The scheme works in two steps. The first step uses the method from [44] to calculate an optimal nonprobabilistic routing table and the data allocation. In the second step, an iterative search procedure has been applied that uses the data allocation of step 1 and strives to find a probabilistic routing table that minimizes the average response time. For this purpose, an analytical model is used during the search procedure to calculate an estimate for the average response times.

The computational overhead of both schemes seems prohibitive for use in an adaptive (semidynamic) scheduling policy. Also, given that the input data for the algorithms is based on averages or estimates, and many important dynamic aspects cannot be considered a priori (hit ratios, concurrency control conflicts, etc.), it seems superfluous to try to derive optimal solutions or to predict the exact response time. Much more efficient heuristic algorithms to determine the routing table and data/PCA allocation have been proposed [7, 47].The proposal in [7] is restricted to a deterministic load assignment and works in two steps. The first step calculates a routing table such that each node has to process about the same amount of work (load balancing). The second step determines a PCA allocation that minimizes the number of remote requests for the load allocation of step 1. The scheme in [47] permits a probabilistic routing and determines the load and data/PCA allocation in a coordinated way to achieve a lower number of remote requests. In each step of this iterative heuristic, a transaction type (or some part of it) is assigned to one node such that this node is not overloaded. This assignment starts with the largest transaction type and is continued until all workload groups are allocated. The data/PCA distribution is adapted in each step of the assignment procedure such that the load distributed so far can be processed with a minimum of internode communication.

Routing tables determined with the latter method were used for load allocation in an empirical simulation study of data-sharing systems [23, 24]. The simulation system was driven by traces of six real-life transaction processing applications and included different algorithms for concurrency control (among them PCL) and buffer management. Though substantial performance benefits of affinity-based over random routing schemes could be quantified, the problems of a purely table-driven routing scheme were also revealed. It turned out that with the predetermined transaction allocation, load balancing could not generally be achieved (routing table and PCA allocation remained unchanged during a simulation run). Not only great differences in the CPU utilization of different nodes and overleaded CPUs were observed, but also significant differences with respect to I/O, communication, and concurrency control conflict frequencies. However the algorithms for calculating a routing table usually (have to) assume that the frequency of these events and the resulting CPU overhead and response-time delays are nearly the same in all nodes (in fact, even the communication overhead for remote requests is generally ignored in the calculation of the routing table). As a consequence, the table-driven routing often failed to achieve a balanced system utilization. This was particularly a problem for higher transaction rates, where not only increased CPU requirements but also increased data contention had to be dealt with.

In part, these problems may be resolved by dynamically adapting the routing table (semidynamic approach) and additionally by applying local workload management techniques. Also, not all performance problems can be solved by load control techniques but are influenced by many other factors (data base and application design, algorithms used for concurrency control and buffer management, etc.). Still, our experience indicates that a purely table-driven approach is not appropriate for achieving load balancing, although it is an efficient way to support affinity-based routing decisions. This is because only the average CPU utilization may be predictable and can thus be balanced by a predetermined routing table, but not the actual utilization. Even a prediction of the average utilization is difficult and based on many simplifications and estimates.

A possible solution may be dynamic, group-based approaches that draw a routing decision from a routing table and also consider the current system state (e.g., CPU utilization) to avoid assignments to already loaded systems. Such schemes would normally assign transaction requests to the node where they can be processed with minimal communication overhead. Only if the preferred destination is already loaded would a different routing decision be drawn. In this way the number of nonpreferable assignments can be reduced, particularly if compared with a nondeterministic routing in which a transaction type is allocated according to a predetermined probability function. The details and quantitative evaluation of such an approach, however, are subjects for future research.

7. CONCLUSIONS

Effective workload allocation is a key factor for high performance in distributed transaction systems. Apart from load balancing, transaction routing should support

transaction processing with a minimum of internode communication, I/O, and lock delays. To this end, affinity-based transaction routing that tries to achieve node-specific locality by assigning transactions that access the same data base portions to the same destination node(s) should be used. In addition, load distribution should be adaptive to permit a flexible and automatic adaptation to changing conditions and to simplify system administration. Current TP systems fail to provide such load allocation strategies. They generally apply static workload allocation schemes controlled by system personnel (e.g., static terminal assignment to nodes).

Developing effective and efficient transaction routing policies is difficult. One complication arises from the fact that load balancing alone is not sufficient for efficient transaction execution, but that data base-related aspects (communication, I/O, data contention) are generally more important factors. In addition, there are numerous workload and system dependencies and realization alternatives to consider, resulting in a huge design space. The framework presented here is a first attempt to summarize the major dependencies and to structure the solution space. The discussion was based on a classification of alternative system architectures and execution models for distributed transaction processing. We found that the homogeneous, locally coupled SN and SD architectures offer the highest potential for workload allocation. We then described the major structural and implementation aspects to be considered and revealed basic design tradeoffs. Finally, the classification and evaluation of existing implementations and proposals for transaction routing demonstrated the usefulness and power of our framework.

So far, implementations and proposals for transaction routing are mostly based on a source-initiated approach and message-oriented communication. The discussion in section 4, however, shows that server-initiated policies and storage-based communication are viable alternatives that deserve serious investigation. While we have concentrated here on workload allocation for canned transactions, separate strategies may be necessary for batch transactions and complex ad-hoc queries because of their large resource requirements, which may penalize the concurrent execution of short transactions. Local load control measures and their coordination with global control decisions is another important area for future research.

REFERENCES

1. A Measure of Transaction Processing Power *Datamation* 112-118 (April 15, 1985).
2. T. W. Scrutchin, Jr., TPF: performance, capacity, availability, in *Proceedings of the IEEE Spring CompCon*, San Francisco, IEEE Computer Society Press, 1987, pp. 158-160.
3. T. L. Casavant and J. G. Kuhl, A taxonomy of scheduling in General-Purpose Distributed Computing Systems, *IEEE Trans. Software Eng.* 14, 141-154 (1988).
4. D. Ferguson, G. Leitner, G. Wasilkowski, Y. Yemini, C. Nikolaou, Dynamic Load Balancing in Distributed Computer Systems, Technical Report, Columbia University, New York, 1986.
5. Y. Wang, and R. J. T. Morris, Load Sharing in Distributed Systems, *IEEE Trans. Comp.* 34, 204-217 (1985).
6. S. Zhou, Performance Studies of Dynamic Load Balancing in Distributed Systems, Ph.D. Thesis, University of California at Berkeley, 1987.
7. A. Reuter, Load control and load balancing in a shared database management system, in *Proceedings of the 2nd IEEE International Conference on Data Engineering*, Los Angeles, IEEE Computer Society Press, 1986, pp. 188-197.
8. A. S. Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
9. M. Stonebraker, The Case for Shared Nothing, *IEEE Database Eng.* 9, 4-9 (1986).
10. J. P. Strickland, P. P. Uhrowczik, and V. L. Watts, IMS/VS: An Evolving System, *IBM Syst. J.* 21, 490-510 (1982).
11. N. P. Kronenberg, H. M. Levy, and W. D. Strecker, VAX Clusters: A Closely Coupled Distributed System, *ACM Trans. Comp. Syst.* 4, 130-146 (1986).
12. Customer Information Control System (CICS), General Information, IBM Manual GC33-1055-3, 1987.
13. A. Borr, Transaction monitoring in Encompass: reliable distributed transaction processing, in *Proceedings of the 7th International Conference on Very Large Data Bases*, 1981, pp. 155-165.
14. The Tandem Database Group; NonStop SQL: a distributed, high-performance, high-availability implementation of SQL, in *Proceedings of the 2nd International Workshop on High Performance Transaction Systems,* Asilomar, CA, IEEE Computer Society Press, 1987, Lecture Notes in Computer Science 359, Springer-Verlag, 1989, pp. 60-104.
15. A. Z. Spector, R. F. Pausch, and G. Bruell, Camelot: a flexible, distributed transaction processing system, in *Proceedings of the IEEE Spring CompCon*, San Francisco, IEEE Computer Society Press, 1988, pp. 432-437.
16. DBC/1012 Data Base Computer Concepts and Facilities, Teradata Manual C02-0001, 1983.
17. J. Moad, The Database Dimension, *Datamation* 59-63 (May 15, 1989).
18. E. Rahm, Primary Copy Synchronization for DB-sharing, *Infor. Syst.* 11, 275-286 (1986).
19. D. Sacca and G. Wiederhold, Database Partitioning in a Cluster of Processors, *ACM Trans. Database Syst.* 10, 29-56 (1985).

20. A. R. Hevner and A. Rao, Distributed Data Allocation Strategies, *Adv. Comp.* 27, 121-155 (1988).
21. P. M. G. Apers, Database Allocation in Distributed Database Systems, *ACM Trans. Database Syst.* 13, 263-304 (1988).
22. G. Arceneaux, D. Binger, J. Collins, J. Day, L. Girard, M. Hughes, R. Kubitz, K. Madsen, M. Noonan, S. Pelt, R. Wayman, A Close Look at PATHWAY, Technical Report SEDS-003, Tandem Software Education and Design Support, 1982.
23. E. Rahm, Concurrency Control in Multiprocessor Database Systems: Concepts, Implementation and Quantitative Evaluation, Ph.D. Thesis (in German), Informatik-Fachberichte 186, Springer-Verlag, 1988.
24. E. Rahm, Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Data Sharing, IBM Research Report 14325, Yorktown Heights, New York, 1988, to appear in ACM Trans. Database Syst.
25. S. Khosafian, P. and P. Valduriez, Parallel execution strategies for declustered databases, in *Database Machines and Knowledge Base Machines, Proceedings of the 5th International Workshop on Database Machines, 1987* M. Kitsuregawa, H. Tanaka, eds., 1988, pp. 458-471.
26. H. Pirahest, C. Mohan, J. Cheng, T. S. Liu, P. Selinger, Parallelism in relational data bases systems: architectural issues and design approaches, in *Proceedings of the 2nd International Symposium on Databases in Parallel and Distributed Systems*, Dublin, IEEE Computer Society Press, 1990.
27. IMS/VS Version 2: Administering the System, IBM Manual SC26-4176-1, 1987.
28. IBM's Transaction Processing Facility: Another SNA Operating System, *SNA Perspective*, 8, 2-8 (1987).
29. MVS/Extended Architecture, JES3 Introduction, IBM Manual GC23-0049-3, 1987.
30. C. N. Nikolaou, F. N. Eskesen, D. F. Ferguson, N. Halim, J. A. Pershing, A. Stagg, Issues in the Design of a Highly Available Multiple Processor Network Attachment, IBM Research Report RC 12594, Yorktown Heights, New York, 1987.
31. E. Rahm, D. Ferguson, L. Georgiadis, C. Nikolaou, G. Su, G. Wang, Goal-Oriented Workload Management in Locally Distributed Transaction Systems, IBM Research Report 14712, Yorktown Heights, New York, 1989.
32. P. Krueger and M. Livny, A comparison of preemptive and non-preemptive load distributing, in *Proceedings of the 8th IEEE International Conference on Distributed Computing Systems*, 1988, IEEE Computer Society Press, San Jose, pp. 123-130.
33. D. L. Eager, E. D. Lazowska, and J. Zahorjan, Adaptive Load Sharing in Homogeneous Distributed Systems, *IEEE Trans. Software Eng.* 12, 662-675 (1986).
34. S. Zhou, A Trace-Driven Simulation Study of Dynamic Load Balancing, *IEEE Trans. Software Eng.* 14, 1327-1341 (1981).
35. K. Shoens, Data Sharing vs. Partitioning for Capacity and Availability, *IEEE Database Eng.* 9, 10-16 (1986).
36. N. S. Bowen, C. Nikolaou, and A. Chafoor, Hierarchical workload allocation for distributed systems, in *Proceedings Parallel Processing*, vol. II, 1988, pp. 102-109.
37. D. Ferrari, A study of load indices for load balancing schemes, in *Workload Characterization of Computer Systems and Computer Networks* (G. Serazzi, ed.), North Holland, 1986.
38. D. Ferrari and S. Zhou, An empirical investigation of load indices for load balancing applications, in *Proceedings of Performance 87*, North-Holland, 1987, pp. 515-528.
39. M. J. Carey, M. Livny, H. Lu, Dynamic task allocation in a distributed database system, in *Proceedings of the 5th IEEE International Conference on Distributed Computing Systems*, Denver, IEEE Computer Society Press, 1985, pp. 282-291.
40. M. J. Carey and H. Lu, Load balancing in a locally distributed database system, in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, D.C., 1986, pp. 108-119.
41. A. Thomasian, A performance study of dynamic load balancing in distributed systems, in *Proceedings of the 7th IEEE International Conference on Distributed Computing Systems*, West Berlin, IEEE Computer Society Press, 1987, pp. 178-184.
42. P. S. Yu, S. Balsamo, and Y. Lee, Dynamic Transaction Routing in Distributed Database Systems, *IEEE Trans. Software Eng.* 14, 1307-1318 (1988).
43. P. S. Yu and A. Leff, On Robust Transaction Routing and Load Sharing, *ACM Trans. Database Syst. 16*, 476-512 (1991).
44. D. W. Cornell, D. M. Dias, and P. S. Yu, On Multi-System Coupling Through Function Request Shipping, *IEEE Trans. Software Eng.* 12, 1006-1017 (1986).
45. P. S. Yu, D. W. Cornell, D. M. Dias, and B. R. Iyer, Analysis of Affinity Based Routing in Multi-System Data Sharing, *Perform. Eval.* 7, 87-109 (1987).
46. P. S. Yu, D. W. Cornell, D. M. Dias, and A. Thomasian, Performance Comparison of I/O Shipping and Database Call Shipping Schemes in Multisystem Partitioned Databases, *Perform. Eval.* 9, 15-34, (1989).
47. E. Rahm, Algorithms for efficient load control in multiprocessor database systems, *Angewandte Informatik* 28, 161-169 (1986) (in German).