TELEMATICS RESEARCH AT PRINCETON - 1988

Robert Abbott, Rafael Alonso, Daniel Barbara,
Brad Barber, Matt Blaze, Chris Clifton,
Luis Cova, Hector Garcia-Molina, Boris Kogan,
Kriton Kyrimis, Christos Polyzois, Kenneth Salem,
Patricia Simpson, and Annemarie Spauster

CS-TR-219-89

April 1989

TELEMATICS RESEARCH AT PRINCETON — 1988

Robert Abbott, Rafael Alonso, Daniel Barbara, Brad Barber, Matt Blaze,
Chris Clifton, Luis Cova, Hector Garcia-Molina, Boris Kogan,
Kriton Kyrimis, Christos Polyzois, Kenneth Salem, Patricia Simpson,
Annemarie Spauster

Department of Computer Science
Princeton University
Princeton, N.J. 08544

April 1989

TELEMATICS RESEARCH AT PRINCETON — 1988

Robert Abbott, Rafael Alonso, Daniel Barbara, Brad Barber, Matt Blaze,
Chris Clifton, Luis Cova, Hector Garcia-Molina, Boris Kogan,
Kriton Kyrimis, Christos Polyzois, Kenneth Salem, Patricia Simpson,
Annemarie Spauster

Department of Computer Science
Princeton University
Princeton, N.J. 08544

1. Introduction

In this note we briefly summarize the database and distributed computing research we performed in the year 1987. In general terms, our emphasis was on studying and implementing mechanisms for *efficient and reliable* computing and data management. Our work can be roughly divided into 12 categories: load balancing, process migration, information exchange networks, negotiation in federated systems, a stashing file system implementation, probabilistic databases, distributed file comparison, remote backups, document databases, real time database systems, multicast protocols, and a new file system design.

Due to space limitations, we concentrate on describing our own work and we do not survey the work of other researchers in the field. For survey information and references, we refer readers to some of our reports.

2. Load Sharing in Distributed Systems

In many distributed systems it is possible to share the processing capabilities among the nodes. To accomplish this goal, a number of load distribution algorithms have been proposed in the literature. In a network of *independently owned processors* (e.g., a network of workstations), load distribution schemes cannot consider the whole network as one unit and thus cannot try to optimize the overall performance [Cova1988a]. That is to say that *load balancing* schemes are not appropriate for this type of environment. Instead, the needs of each resource owner have to be considered. Therefore *load sharing* is more appropriate.

Informally, there have been other load sharing schemes discussed and used in the load sharing literature. First, is the extensively used **All-or-Nothing** scheme where idle processors are used to service remote jobs until they are claimed by their owners. Second, is the **Priority** schemes where remote jobs are accepted at a node with lower scheduling priority than local jobs. These methods clearly guarantee ownership of resources to the machine's owners. In [Cova1988b], we empirically compared these methods against our own scheme by using a synthetic workload.

This work has been supported by NSF Grants DMC-8351616 and DMC-8505194, New Jersey Governor's Commission on Science and Technology Contract 85-990660-6, and grants from DEC, IBM, NCR, and Concurrent Computer corporations.

The sharing scheme we developed, called High-Low [Alonso1988c], replaces the notion of stealing CPU cycles with the notions of lending and borrowing CPU cycles. Workstation owners do not have to be concern with their resources being abused. The degree of participation of each computer in this load sharing scheme is completely distributed. It does not depend on any global information or central controller, just on the node's local use and purpose.

In our work, we determined that the All-or-Nothing and Priority schemes of load sharing are too restrictive in an environment where most resources are underutilized. Also, they do not scale well as the system load increases. Instead, by allowing a moderated sharing among the nodes, good performance is guaranteed to the node owners and to the remote jobs.

Our results suggest that All-or-Nothing's performance gains can always be improved, no matter the system load. This scheme does not have any feature of merit except that it is the obvious way (and simplest to implement) to guarantee ownership of resources.

High-Low has the desirable characteristic that an All-or-Nothing scheme has, i.e., it guarantees to the owners of lightly loaded machines that their local resources will not be abused by remote jobs. This is achieved by fixing the maximum degradation that a user of a lightly loaded node might perceive. At the same time, it avoids the anomaly of having poorer performance for the heavily loaded machines as the load of the network increases (when it should have an opposite behavior since it is in this situation that improvement is most needed). The High-Low scheme improves the performance of heavily loaded nodes in a greater amount than All-or-Nothing or Priority, and is close to the performance of the load balancing schemes tested (lsh and random).

A further point is that the capability to migrate jobs [Alonso1988b] after they have started executing in a machine may be desirable for our system. For example, a machine may receive too many remote jobs which can suddenly start to demand a large number of its cycles, degrading the performance perceived by local users. To correct this possible anomaly, the machine could move some of these remote jobs back to their originating nodes or to a new host. In this way control of the local resources could still be maintained. Of course, now the research question becomes how can the load sharing system effectively and efficiently monitor remote jobs. A second new interesting research problem arises due to the recent introduction of multiprocessor workstations to the market. In a network of such workstations, load distribution has to be done at two levels: within the local processors of a workstation and among the workstations.

Finally, we realize that the performance of a computer does not depend solely on its CPU utilization. We have just used this resource to illustrate our ideas. The notions behind Low-mark and High-mark could also be applied to whatever local resource becomes the system bottleneck, such as physical memory or disk space.

3. Process Migration

In a distributed computing environment it is often desirable to balance the system-wide load in such a way that we do not have a situation where some machines are busy while others are relatively idle. In a distributed system, a *load balancing* strategy is used to achieve an even distribution of the work. Such strategies may be preemptive or not.

A preemptive implementation is usually said to be able to carry out *process migration*, while non-preemptive policies are referred to as *initial placement*. With initial placement, when a job arrives at a machine, the system determines which is the best machine to run the new job so that the load balance is maintained, and starts that job there. With process migration, each job is run on the machine where it is initiated and, at an arbitrary point in time, the system can decide to move that job to another machine so that the system-wide load can remain balanced.

Determining which of the two methods is the one that should be preferred for an actual implementation is a very controversial subject, as there are arguments in the literature in favour of both. In our past research we have implemented a load balancing mechanism using initial placement [Alon86] and a process migration mechanism [Alon88b] that could be easily used to create a load balancing mechanism. Since we had both methods available to us, we set out to compare the two by running a set of load balancing experiments.

To make a load balancing experiment, a workload must be used that is as realistic as possible. The ideal situation would be to run the experiment on a working system and measure the changes in performance. Unfortunately, this was not possible in our case, because the system that was available to us (seven Sun 2 workstations running version 3.2 of the Sun operating system) is used by very few people, thus having a very low actual workload, so we had to use a synthetic workload. We used distributions that have been derived from traces of actual systems, describing the service demand of a job, and the time between job arrivals. Using these distributions gave us the added advantage that by changing their parameters, we could affect the role that implementation overhead plays in these experiments, which is something that we would not have been able to do if we had used a real workload: a high mean service demand would make the implementation overhead negligible, while a low mean service demand would make the implementation overhead significant. By changing the parameters of the interarrival time, we were also able to study the effect of the CPU utilisation on our experiments: if jobs arrive at long intervals, CPU utilisation is low while, if jobs arrive at short intervals, CPU utilisation is high. To simulate the effect of having both idle and loaded machines in the network, we ran the workload on five machines, while keeping two to act as load servers. We ran our experiments for low, medium and high significance of the implementation overhead and for low, medium and high CPU utilisations.

Our experiments showed that if the implementation overhead is considerable, instead of improving the average job response, initial placement actually makes it worse. On the other hand, the process migration policy produces results similar to the original workload, even producing a marginal improvement when the CPU utilisation is high.

If there is an implementation overhead, but it has a value comparable to the mean service demand, we get similar results as before, except for the case where CPU utilisation is high. In this case, initial placement produces similar results to doing load balancing. Process migration, however, produces a noticeable improvement.

Finally, when the implementation overhead becomes of small significance, both initial placement and process migration improve the average job response significantly, but this time there is little difference between initial placement and process migration. (Actually, our results show that initial placement performs *better* than process migration, due to the high overhead of our migration implementation.)

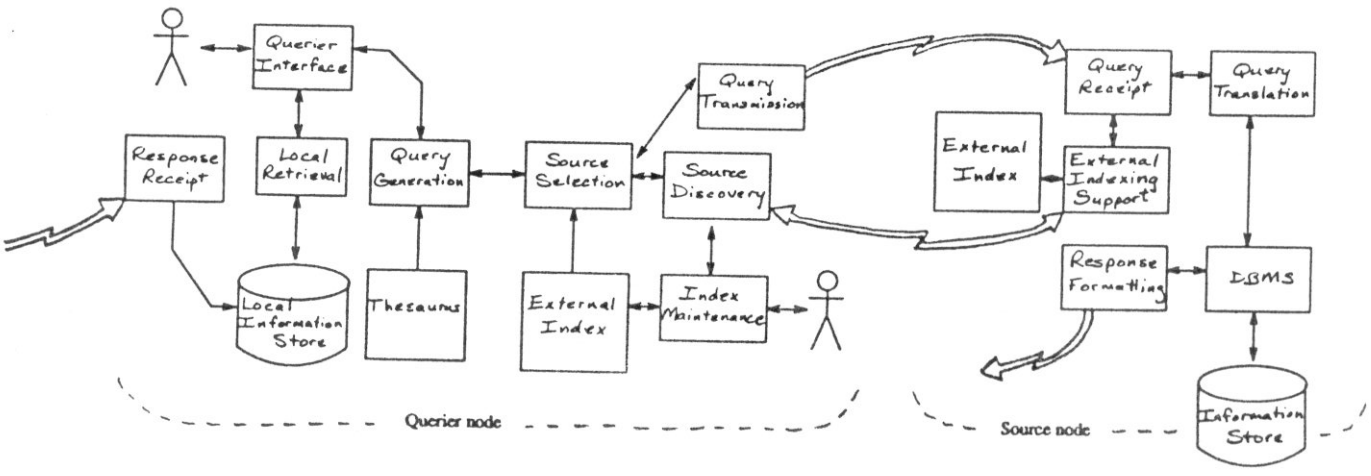
Since any realistic implementation would have some overhead, our experiments suggest that with a reasonably good implementation process migration can help improve job performance when a machine is heavily utilised. The recommended policy is to do nothing while the system load is low to medium, and run the load balancing algorithm that uses process migration when the system load gets high. Our results also show that even if this algorithm is run all the time (i.e., regardless of the system load), there is not going to be any significant degradation. Finally, even when the implementation overhead is very high, our results suggest that when the system load is high, there may be some small improvement in performance if load balancing is used in conjunction with process migration.

4. Information Networks

In 1988 we completed the logical design of an *information exchange network* architecture. Information exchange (IE) is the extension of information retrieval (IR) to a network environment; it enables topic-based searching and retrieval of information located in many independent database systems throughout the network. IE is one of many applications suitable for the network that will naturally arise as individuals and organizations with computer systems gain access to the Integrated Services Digital Network (ISDN) now under construction.

Such a network has three distinguishing characteristics — *universal access*, resulting in potentially millions of application participants; *heterogeneity* of hardware, software, communication protocols (at ISO/OSI layers 4 and above), and user needs; and *autonomy* of all network users, entailing the freedom to enter or leave applications at any time, to found new applications, and to behave inconsistently and unreliably. As part of the design process we analyzed the implications of these characteristics and derived general principles for network application design. These principles include decentralized control, stateless interaction, bottom-up evolution, application independence, and simple communication standards.

Guided by these principles, we have developed an architecture for the IE application. The figure below, taken from [Simp89], illustrates the software modules and data structures necessary to implement IE. The functions performed by querier nodes and information source nodes are shown separately for clarity, but it should be noted that any network participant may act as either querier or source, or both, at any time.



The *query generation* function is performed by the querier. With the help of local query interface software, keywords are extracted from natural language input to produce a transmissible query in query normal form (QNF), which is a simple extension of the term-vector query format widely used in IR systems. The assignment of all querier interface responsibilities to the querier's own processor ensures that each querier can customize his/her view of the network and its resources to suit individual needs and preferences.

The *external indexing*, or source selection, function is also performed by the querier, using a locally constructed partial index of information sources. This external index is built up gradually as needed; it consists of a network address for each known source together with a descriptive summary of the information held there. A vector similarity measure is used to identify sources whose descriptions are similar to the proposed query. The QNF query is then transmitted to these candidate sources.

Each source performs *query translation* of an incoming QNF query into its own internal retrieval language. It executes the query in this form, then transmits the response — a sequence of documents ranked by their similarity to the query — to the querier. In this way queriers need not learn the many different retrieval languages used by sources, and sources retain their autonomy and privacy by hiding internal storage structures from queriers. Translation procedures can be defined for general classes of retrieval languages. In [Simp88] we presented translation algorithms for common types of document-based IR systems. In [Simp89] we sketched a translation procedure for relational database systems.

The source also plays a role in the external indexing function. It must generate, via some summarization function, a description of its information which queriers may store in their indexes. It may announce this descriptor to potential queriers or send it to them in response to their requests for it. Nodes may also request portions of other nodes' external indexes; in this way they enlarge their pool of neighbors to whom both queries and index-building requests may be directed.

5. Negotiating Data Access in Federated Database Systems

The ever growing need for information is putting pressure on organizations to share data with their partners. However, although the different entities would like to share information, it is clear that each individual system administrator would like to preserve his or her control over the system. This concern with each system's autonomy has led to the notion of **federated databases**. Previous work in this area has touched upon the topic of negotiating access in a federated database (i.e., determining what local information may be access by any particular remote user), but we feel that the topic has not been studied in depth.

In [Alon89b] we propose a new scheme, based on the notion of **quasi-copies** [Alon88a], which may be used for interaction among autonomous databases. In the past, the notion of data sharing has been synonymous with letting importers query the local database. We feel, however, that there is a broad spectrum of choices in this area. For example, for performance reasons, one may consider giving the importer a copy of the data. If the importer has very stringent requirements for response time, a local copy might be the answer to efficient sharing. Placing a copy in the importer's site has the added benefit that it also provides a way of lowering the query traffic to the owner site, thus reducing the competition for the owner's resources.

Heterogeneity is a second important reason for using copies. If the systems do not share a common query language, and no copies are kept on the importer's site, all importer queries will have to be made using the owner's query language (which implies the overhead of a query translation mechanism). Of course, once data sharing has been agreed upon, the importer can always keep a copy of the data items without notifying the owner (say, by storing the result of its previous queries). But in reality this is not a feasible solution in many applications since in this case the importer can never be sure of the consistency of the data.

However, when the owner grants a copy of the data, the responsibility for data consistency is an issue that can (and must) be resolved beforehand. The choices range from giving a copy without any commitment (i.e., providing the importer with a **hint** about the data) and leaving the responsibility of the consistency to the importer, to keeping the importer posted of every update. It is clear that the choice has a tremendous impact over the amount of autonomy that the owner has over its data (where by autonomy we mean the degree of freedom that an owner has to update local information). For instance, in the second case above, the owner cannot proceed with an update without informing all the importers first (possibly requiring the expense of two-phase commit or some other sort of concurrency control protocol).

As can be appreciated from the comments above, the interests of the owner and the importer of data may often conflict. The needs of both types of entities must be compared, and an agreement (if possible) should be reached. The process of reaching an agreement is called a **negotiation**, and is a key element to the federated system. If such negotiation is to take place automatically, the need for a protocol is evident. Such a protocol should specify the following two items:

- A way to precisely express the needs of the importers and the degree of sharing that the owner is willing to offer.
- A way to estimate the cost of a specific agreement, both for the owner and for the importer of the data, as a means for justifying such an agreement.

We also provide protocols for access negotiation, as well as simple cost models for estimating the expense involved in allowing remote access to information. One of the main advantages of our new scheme is that it provides a very precise way of establishing how much autonomy is given up by the owner of the information when he or she decides to share data. Finally, our approach may be used in both the case where the databases systems in question have a common query language (or there exist facilities for query translation), and when they do not.

6. A Stashing File System

One type of distributed computing environment consists of a federation of autonomous computers that cooperate with each other. We study networks where the nodes have administrative and functional autonomy relative to their association to the distributed system, i.e., nodes belong to different users, or organizations, and participate in the system to extend their capabilities.

As distributed systems become larger, the appropriate system architectures are those that provide a great deal of support for local node autonomy. Since in such autonomous computing environments we expect that servers will have the freedom to deny service to

any user, we believe that the proper view of a server is not that it is the only place in which to obtain a service or resource, but rather a server is the best place to do so. We presented this idea at the 1988 ACM SIGOPS European workshop on Autonomy or Inter-dependence in Distributed Systems [Alonso1988d]. In a distributed file system based on the client-server model, a possible approach consists of storing at the server the latest copy of all user files, while keeping at the clients copies of older versions of the most crucial information. Thus, even after a failure, users may be able to proceed with their work, although in a degraded manner.

The idea of keeping local copies of key information has been called **stashing** in the literature We augment the usefulness of stashing by combining it with the idea of **quasi-copies** [Alonso1988a] (remote data copies that are allowed to diverge from the primary data but in a controlled, application-dependent fashion), which eases the cost of maintaining replicated data.

At the present we are implementing the first prototype of FACE [Alonso1989a], a distributed file system which allows users the option of using stashed files that are kept sufficiently consistent to fulfill user needs. This prototype is being implemented on the facilities of the Princeton Distributed Computing Laboratory. The system is being developed on a Sun 3/50 computer running Sun's UNIX 4.2 release 3.3 and based on Sun's Network File System

Our design goals for the FACE prototype are: fast prototyping, transparency at the user level, portability to a variety of computer systems, and minimal modifications to the underlying implementation. The prototype provides file stashing to increase the availability of important information when file servers are not reachable. This feature enhances both the autonomy of the local nodes as well as the degree of fault tolerance of the overall system.

We deal with the issue of stashed copy consistency, by incorporating into our design quasi-copy techniques. Thus, users may decide the level of consistency that stashed copies must have in order to meet the demands of their particular applications. We feel that this is an important feature with broad usefulness. It also liberates the system from the burden of keeping perfectly consistent copies in cases where perfect consistency is not required.

7. A Probabilistic Relational Model

It is often desirable to represent in a database entities whose properties cannot be deterministically classified. For example, customers living in a certain area are likely to purchase certain model of car with probability 0.7, and its sports version with probability 0.3. An oil company might have a database of potential oil sites with probabilistic projections as to the type and quantity of oil at each site. Medical records describing a the causes of a disease could be stochastic in nature.

We have developed the Probabilistic Data Model (PDM) [Barb89a], an extension of the relational model that lets one represent probabilities associated with the values of attributes. In this model, relations have deterministic keys. That is, each tuple represents a known real entity. The non-key attributes describe the properties of the entities and may be deterministic or stochastic in nature. To illustrate, consider the following probabilistic relation:

Key	Independent Deterministic	Interdependent Stochastic	Independent Stochastic
EMPLOYEE	DEPARTMENT	QUALITY BONUS	SALES
Jon Smith	Toy	0.4 [Great Yes] 0.5 [Good Yes] 0.1 [Fair No]	0.3 [$30,000] 0.5 [$35,000] 0.2 [*]
Fred Jones	Houseware	1.0 [Good Yes]	0.5 [$20,000] 0.5 [$25,000]

The relation describes two entities, "Jon Smith" and "Fred Jones." Attribute DEPARTMENT is deterministic. e.g., it is certain that Jon Smith works in the Toy department. In this example, QUALITY and BONUS are probabilistic and jointly distributed. The interpretation is that QUALITY and BONUS are random variables whose outcome depends on the EMPLOYEE in consideration. For instance,

Prob[QUALITY = "Great" AND BONUS = "Yes" | EMPLOYEE = "Jon Smith"] = 0.4

It makes sense to consider QUALITY and BONUS jointly distributed if, as the example implies, QUALITY functionally determines BONUS. The last attribute, SALES, describes the expected sales in the coming year by the employee. It is probabilistic but independent of the other non-key attributes. For instance,

Prob[SALES = "$35,000" | EMPLOYEE = "Jon Smith"] = 0.5

In the attribute SALES, probability 0.2 has not been assigned to a particular sales value. It is assumed that this missing probability is distributed over all domain values, but we make *no assumptions* as to how it is distributed. This situation could arise if, for instance, 10 people are polled to estimate Jon Smith's sales for the upcoming year. Three people have estimated sales of 30,000 (e.g., last years sales); five of 35,000 (e.g., last year plus inflation). Two people have not been reached and give rise to the 0.2 missing probability. Since the missing probability could or could not go to the value 30,000, the probability that the sales will be 30,000 next year is actually between 0.3 and 0.3 + 0.2. In this sense, the probability 0.3 associated with the sales value $30,000 is a lower bound.

We believe that missing probability is a powerful concept. It allows the model to capture uncertainty in data values as well as in the probabilities. It facilitates inserting data into a probabilistic relation, i.e., it is not necessary to have all information before some tuple can be entered.

In [Barb89a] we formalize our probabilistic model along with the conventional operators for the relational model. We also investigate the ''correctness'' of the operators on missing probability, and introduce a new set of probabilistic relational operators.

8. Randomized Techniques for Remote File Comparison

Files are replicated in a distributed system in order to improve reliability and performance. However, due to human errors or hardwqre failures, copies may diverge. It then becomes necessary to compare the remotely located files and identify the differences. A simple approach would be to transmit the entire file and do a compare at a single site. However, this approach is not feasible when the files are large.

In [Barb89b], we present a file comparison technique that uses randomized signatures. That is, before any comparison takes place, a set of random signatures is agreed upon by the two sites. This can be achieved by making one of the sites apply the algorithm that selects the random signatures and later send the set to the other site or by making the two sets use the same pseudo-random algorithm, starting with the same seed. Once the set is agreed upon by the two sites, the file comparisons can take place. Notice that the selection of the signatures takes place only **once**. From there on, the same set of signatures shall be used for every comparison. This strategy can be tuned up to correctly diagnose up to an arbitrary number of differing pages f, with a probability that can be made as close to 1 as desired. We also use this technique to introduce a metric of complexity for one-message strategies, that meassures the number of bits needed to be sent to diagnose up to f differing pages, keeping the probability of false diagnose bounded by a given amount. In previous work, the analysis of the techniques was made only in terms of number of signatures sent, only mentioning that the number of bits per signature had to be sufficiently large to make the likelyhood of the collision of two different pages very small. This metric is defined as follows:

Definition: For a given one message strategy, $\beta(n, f, \delta)$ is the number of bits sent to diagnose up to f differing pages in a file with n pages keeping the probability of false diagnose bounded by δ.

The technique developed in this paper, called **FIND_INNOCENTS**, discards pages that are in signatures that agree and form the set of suspicious pages with the rest. (The same way the police would discard somebody from the list of suspects if this person has a strong alibi.) It achieves a bound

$$\beta(n, f, \delta) = O(f(log(n) + log(\frac{1}{\delta}))$$

$$log(log(n)) + log(f) + log(\frac{1}{\delta})))$$

and proves to be very competitive in practice with previously published algorithms.

9. Management of a Remote Backup Copy for Disaster Recovery

In critical database applications, the halting of the computer system in case of failure is considered unacceptable. Instead, it is desirable to keep an up-to-date backup copy at a remote site, so that the backup site can take over transaction processing until the primary site recovers. Such a remote backup database (or *hot standby*), normally of the same capacity as the primary database, should be able to take over processing immediately. Furthermore, when the primary site recovers, the backup should provide it with a valid version of the database that will reflect the changes made by transactions processed while the primary was not operational; this will enable the primary site to resume normal processing. Finally, the overhead at the primary site for the maintenance of the remote backup copy should be kept low.

Systems for maintaining a hot standby are commercially available. They are typically built around a central process that records the logs of actions performed by transactions running at the primary site, transmits the logs to the backup site and replays them

against the backup copy. The correctness of this approach relies heavily on the existence of a *single* logging process, which renders the method non-scalable and therefore non-applicable to high performance processing systems. Such systems consist of multiple processors, so that a single logging process would form an bottleneck.

In [King88] we studied the maintenance of a hot standby in the case of multiple processors. We explain why the simple algorithm used in commercial systems does not generalize easily. We presented a *fully decentralized* mechanism for keeping a remote backup copy up-to-date. Our mechanism, which is also based on the transmission of logs, applies to a wide range of parallel systems without sacrificing much of the potential for parallelism, while imposing a minimal overhead.

Our method guarantees that the remote site will always contain a consistent version of the database. Take-over time is kept low, but a few transactions that were executing just prior to the occurrence of a disaster may be lost, although they may have committed at the primary. This is the price for avoiding a global two-phase commit protocol, which would decrease system performance significantly. While both sites are operational, our scheme ensures that the two copies will also be *mutually* consistent; inconsistencies are avoided because transactions commit in the same relative order at both sites.

We have also developed a method for initializing a remote copy (this can also be used when a site recovers from disaster). It is similar to a fuzzy dump, but more flexible: it allows the logs to be replayed even before the dump completes. This reduces start-up time and allows initialization to take place without affecting the processing rate at the site processing incoming transactions.

10. A Document Database

Traditional database management systems are designed for information which consists of large numbers of identically structured *records*. Much of the data which can be stored and accessed using a computer does not fit cleanly into such a structure. In particular many documents, involving text as well as pictures, graphs, and other media, are now created and stored on computers. Currently such information is stored in file systems and does not benefit from the access and reliability technology which has been developed in the Database Management field.

One model for electronic documents which is growing in popularity is *hypertext*. In this model, nodes containing information are connected by links, and the user ''reads'' a document by following these links. This model has a problem scaling to large collections of documents. Although it is nice to use if one knows how to get to a document, it can be difficult to find information if one does not know which links to follow. We have developed a query language which supports the hypertext-style browsing interface, and integrates this with non-navigational queries. We view this query language as an interface between an application which presents the document, and a ''document server'' which provides storage, reliability, sharing, and other facilities offered by traditional databases. An overview of this approach is given in [Clif88].

We are looking at a number of issues that arise in such a system. One example is *indexing*. In our system queries may only operate on a portion of the database determined by the links between documents. Indexes must take into account these links when defining the *scope* of the index. This is in contrast to a relational system where the scope

of an index is a relation or view, which is defined independent of the database content. We have developed and analyzed a number of approaches for indexing in this environment [Clif89]. These involve "hierarchies" of indexes associated with different locations in the database. Each index is responsible for documents in a subset of the database, and queries which access areas of the database containing many indexes make use of all of them in a coherent manner. This work may have applications in other hypertext systems as well as in object-oriented databases.

We are currently building a prototype database manager based on these ideas. The system is capable of processing queries, and should be able to operate as a full-fledged document server in the near future. We are investigating user interfaces which provide a hypertext-style browsing "look and feel" for composing non-navigational queries.

11. Real Time Database Processing

Existing database management systems do not provide real-time services. They process transactions as quickly as they can, but they do not guarantee that a transaction will meet its deadline. Furthermore, most users cannot even tell the system what priority their request has. Hence, all transactions are treated as equal.

Many applications do have at the same time real-time constraints and large data needs (e.g., aircraft tracking, hospital monitoring, reservations systems). Since database systems do not provide the required real-time response, users have had to code their own special purpose data management systems. Although such systems seem to work, they are difficult to debug and to expand. Thus we believe it is time to investigate a general purpose real-time database system.

Such a system may very well be distributed, but we have decided to focus initially on a centralized one. The first problem we have addressed in this area is that of transaction scheduling. In the future we plan to study additional issues like the general architecture, how to trigger events efficiently, and the appropriate user interface.

Real time transaction scheduling differs from conventional scheduling in that the transactions (or tasks) make *unpredictable* resource requests, mainly requests to read or write the database. In our case, the scheduling algorithm must be combined with the concurrency control algorithm (which guarantees that executions are serializable). To illustrate the interaction, consider a transaction T_1 that is being executed because its deadline is the nearest. Now assume that T_1 requests a lock that is held by transaction T_2. What should the system do? Abort T_2 so that the lock is released and T_1 can proceed? Or maybe suspend T_1 so that T_2 can complete and release its lock? Or maybe it is best to let T_1 proceed without aborting T_2, hoping that the schedule will still be serializable (optimistic control)?

In a first study we developed a family of locking-based scheduling algorithms for memory resident real-time database systems [Abbo88a]. Each algorithm has three components: a policy to determine which tasks are eligible for service, a policy for assigning priorities to tasks, and a concurrency control mechanism. The eligibility policy determines whether a transaction that has already missed its deadline (or is about to) can be aborted and not performed at all. The priority policy orders the ready transactions according to their deadline, the slack time, or the arrival time. The concurrency control policy specifies the action to take when lock conflicts occur.

We studied the performance of the scheduling algorithms via a detailed event-driven simulation [Abbo88b]. The parameters in the model include the arrival rate of transactions, the average number of objects updated, the available slack for meeting the transaction's deadline, and the possible error in the transaction's running time estimate. The performance metrics we studied were the number of missed deadlines, the transaction throughput, and the number of aborted (and restarted) transactions.

Very briefly, our results indicated that one priority policy (Earliest Deadline) and one concurrency control mechanism (Conditional Restart) are superior in most cases. They also indicate that screening for ineligible transactions can significantly reduce the number of missed deadlines (as long as the application allows this type of screening).

In a second study we expanded our model to include disk resident database systems and both shared and exclusive locks [Abbo89]. The IO system is now an important component for achieving real-time response. We developed two different ways in which to schedule IO requests at the disk. We also examined two additional concurrency control algorithms including one which promotes the priority of a transaction which is blocking a higher priority transaction from executing.

The algorithms were evaluated via a detailed event-driven simulation. Some of the new parameters in the model were size of the database, average number of objects locked exclusively and the speed of the disk server. The performance metrics used to rate the different algorithms were number of missed deadlines, average tardy time of transactions, and number of aborted (and restarted) transactions. We also studied the algorithms under a simulated load spike, or the introduction of a flurry of jobs in an otherwise lightly loaded system.

Very briefly, our results indicated that the priority policy Earliest Deadline worked best under low load settings but the policy Least Slack performed better when the load was high. The concurrency control policy Wait-Promote was generally the best when the load was steady and continuous but the policy High Priority performed better under the spike load simulation. We learned that scheduling IO requests according to transaction priority can benefit performance.

12. Multicast Protocols

A multicast group is a collection of processes that are the destinations of the same sequence of messages. These messages may originate at one or more source sites and the destination processes may run on one or more sites, not necessarily distinct. A multicast protocol ensures that the messages are delivered to the appropriate processes. There are two types of properties that multicast protocols may ensure: ordering and reliability. We have studied how these properties can be *efficiently* guaranteed.

We have distinguished three ordering properties that a multicast protocol may provide. The first is *single source ordering* which guarantees that if messages m_1 and m_2 originate at the same source site, and if they are addressed to the same multicast group, then all destination processes get them in the same relative order. The second, *multiple source ordering*, lets m_1 and m_2 originate from different sources. Finally, *multiple group ordering* not only lets m_1 and m_2 come from different sources, but lets the destination processes be members of different, but overlapping, multicast groups.

We have developed a new multicast protocol that guarantees these three properties [Spau88,Spau89]. The basic idea is to organize the nodes into a logical tree structure, called a message propagation graph. Messages flow down the tree and are ordered as they move along. The number of messages that must be transmitted is much smaller than in other fully distributed solutions. In many cases, our broadcast tree will have small depth and the multicast delay may also be smaller than in conventional solutions. The disadvantage with our approach is that the tree structure must be set up, and hence it is not advisable for applications where the multicast groups are rapidly changing.

Along with providing ordering properties, multicast protocols must often be *reliable*, i.e., they must make guarantees about the receipt of messages at the destinations. In our investigation of multicast ordering we have distinguished three types of reliability. The first two alternatives provide for atomic delivery of messages, i.e., the messages are delivered at the destinations in a consistent order and cannot be "undelivered" and redelivered later. Under atomic delivery there are two options, operational delivery and insistent delivery. Operational delivery merely guarantees that message delivery will be atomic at "up" sites. Insistent delivery guarantees that *all* sites eventually deliver each message atomically, even those sites that were down when a message was delivered everywhere else and have since recovered. The third option is non-atomic delivery. Non-atomic delivery is insistent, but for performance reasons delivery can be undone and redone. Deciding which alternative is best depends on the distributed system application and the performance requirements.

Guaranteeing reliability for the message propagation graph is discussed in [Spau88] and there we argue that we can guarantee the same reliability as other approaches in an efficient way. The most important feature of our approach is that we do not let the reliability guarantee significantly affect the performance of the ordered multicast delivery during non-failure operation (the most common situation). In addition, the message propagation graph algorithm can guarantee all three of the reliability alternatives outlined above.

13. The iPcress File System

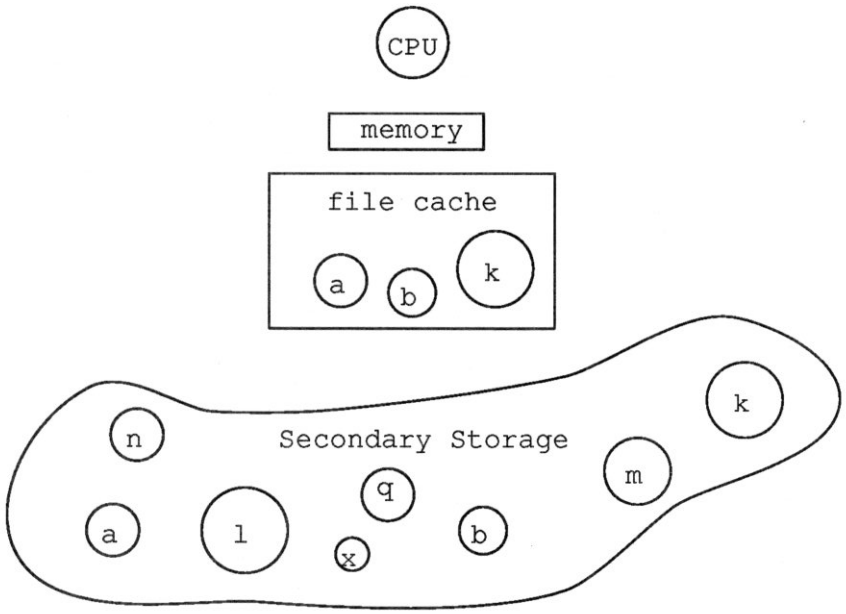
The iPcress File System was designed for general purpose computing in response to several shortcomings in the existing UNIX file systems. Primarily, the UNIX file system is too slow, but it also suffers from a variety of other ills, such as poor reliability. However, the UNIX file system has a basic interface, and its view of files as a stream of bytes is a very fundamental and useful abstraction. Consequently, the iPcress file system keeps the simple UNIX interface, but it completely discards the UNIX implementation of the file system.

The iPcress file system is being implemented on a VAX 11/785 with 128MBytes of core memory running MACH. It uses MACH threads, ports, and shared memory facilities. The file system is run as a user process and it communicates with its clients using ports. File data is transferred between client and server via shared memory. The system was written in GNU C++, and uses the GNU C++ library and generic facilities.

There were several goals in the design of the iPcress file system. First of all, we wanted to utilize existing hardware and I/O bandwidth more effectively. Secondly, we wanted to provide better reliability and recovery than that afforded by UNIX. Additionally, we wanted it to be able to integrate all secondary storage devices for a host, such as

disks, electronic disks and video jukeboxes, into a single file system. Finally, we wanted to add facilities for the collection of file system statistics, in order to facilitate future analysis of file system use. In an earlier study performed jointly between Amdahl and Princeton [Stae88], we discovered that certain statistics could help in predicting future file usage. Thus, we feel it is desirable to keep and then use statistics for caching decisions.

First of all, we decided that each host should have a single file system which includes all secondary storage for the machine. This is a very fundamental change from the UNIX file system, which places a separate file system on each device. It also allows us to add several important features, such as mirrored files, in the natural way. In addition, it implies that the system should be insensitive to the loss of any given device. Finally, it allows the system to monitor and control all file operations and secondary storage for the whole system. A diagram of the conceptual structure of the system is shown in the system level block diagram.



System Level Block Diagram

Secondly, we use a file cache, rather than a block cache. A file cache will stage and flush complete files, unless the file is very large. If files are usually contained in a single extent, then the available bandwidth to disk is utilized as effectively as possible. In addition, if the file system were to keep statistics for each file regarding past file access patterns, then the file cache (and the secondary storage manager) could attempt to optimize its behavior. Most heavily accessed files should live within the cache, and most read requests should be fulfilled from the cache rather than disk.

In order to optimize performance, we use a variable size blocking scheme, so that most files could be contained within a single contiguous block of storage (extent). We use the buddy system for both memory and disk management because of its simplicity and performance. The buddy system allocates blocks in sizes which are a power of two, which can result in large internal fragmentation costs. Therefore we decided to use a variant on the buddy system similar to that used in the DTS file system. which reduces internal fragmentation by splitting some files into several extents.

One technique unique to iPcress is the online internal collection of file access statistics by the file system. The file system keeps a set of statistics for each file regarding that file's access patterns. This is useful for a variety of optimization techniques. First of all, it may be used to balance disk loads, or to influence the layout of files within a disk. It may also be used to detect which files may be split into several extents without greatly impacting system performance. The question of exactly which statistics will be kept by the system is an interesting open question.

We decided to improve file system reliability by adding a transaction mechanism and various file reliability modes. We wanted the file system to use transactions internally when accessing internal data structures such as directories or free lists. In addition, we also wanted to provide these features to the user without adding a second transaction facility. Consequently, we store all file system data structures in files, and all reliability and recovery mechanisms operate on files. This decision has several consequences, the primary one being that the design of the file system must be layered, with the lowest level providing simple files, and with each additional layer adding more functionality.

14. References

[Abbo88a]
R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions," *ACM SIGMOD Record*, Vol. 17, No. 1, March 1988, pp. 71-81.

[Abbo88b]
R. Abbott and H. Garcia-Molina, "Scheduling Real Time Transactions: A Performance Evaluation," *Proc. 14th VLDB Conference*, Long Beach, California, August 1988.

[Abbo88a]
R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data," Technical Report CS-TR-207-89, Department of Computer Science, Princeton University, February 1989.

[Alon86]
Rafael Alonso, Phillip Goldman, and Peter Potrebic, "A Load Balancing Implementation for a Local Area Network of Workstations," *Proceedings of the IEEE Workstation Technology and Systems Conference*, March 1986, pp. 118-124, Atlantic City, NJ.

[Alon88a]
Rafael Alonso, Daniel Barbara, Hector Garcia-Molina, and Soraya Abad, "Quasi-Copies: Efficient Data Sharing for Information Retrieval Systems," *Proceedings of the 1988 International Conference on Extending Data Base Technology*, March 14-18, 1988, Venice, Italy.

[Alon88b]
Rafael Alonso and Kriton Kyrimis, "A Process Migration Implementation for a UNIX System," *Proceedings of the Winter 1988 USENIX Conference*, February 9-12, 1988, Dallas, Texas.

[Alon88c]
Rafael Alonso and Luis Cova, "Sharing Jobs Among Independently Owned

Processors,'' *Proceedings of the 8th International Conference on Distributed Computing Systems*, June 13-17, 1988, San Jose, California.

[Alon88d]
Rafael Alonso and Luis Cova, ''Resource Sharing in a Distributed Environment,'' roceedings of the 1988 ACM SIGOPS European Workshop, Cambridge, England, September 1988.

[Alon89a]
Rafael Alonso, Daniel Barbara, and Luis L. Cova, ''FACE: Enhancing Distributed File Systems for Autonomous Computing Environments,'' Technical Report, Department of Computer Science, Princeton University, March 1989.

[Alon89b]
R. Alonso, and D. Barbara, ''Negotiating Data Access in Federated Database Systems'', *Proceedings of the Fifth Conference on Data Engineering*. Los Angeles, Ca, Feb. 1989.

[Barb89a]
D. Barbara, H. Garcia-Molina and D. Porter, ''A Probabilistic Relational Model'', Technical Report CS-TR-215-89

[Barb89b]
D. Barbara, and R.J. Lipton, ''A Randomized Technique for Remote File Comparison'', *Proceedings of the Ninth International Conference on Distributed Computing Systems*. Newport Beach, CA, June 1989.

[Clif88]
Chris Clifton, Hector Garcia-Molina, and Robert Hagmann, ''The Design of a Document Database,'' in *Proceedings of the Conference on Document Processing Systems,* pp. 125-134, ACM, Santa Fe, New Mexico, December 5-9, 1988.

[Clif89]
Chris Clifton and Hector Garcia-Molina, ''Indexing in a Hypertext Database,'' Technical Report CS-TR-206-89, Princeton University, Princeton, NJ, February 1989.

[Cova88a]
Luis Cova, ''Load Balancing in Two Types of Computing Environments,'' Technical Report, Department of Computer Science, Princeton University, June 1989.

[Cova88b]
Luis Cova, and Rafael Alonso, ''Distributing Workload Among Independently Owned Processors,'' Technical Report, Department of Computer Science, Princeton University, December 1988.

[King88]
R. P. King, N. Halim, H. Garcia-Molina, C. A. Polyzois, "Management of a Remote Backup Copy for Disaster Recovery", Technical Report.

[Simp88]
Patricia Simpson, ''Query Processing in a Heterogeneous Retrieval Network,'' *Proc. Eleventh Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, June 1988, Grenoble, France.

[Simp89]
Patricia Simpson and Rafael Alonso, ''Querying a Network of Autonomous Databases,'' Technical Report CS-TR-202-89, Department of Computer Science, Princeton University, January 1989.

[Spau89]
H. Garcia-Molina, A. Spauster, "Message Ordering in a Multicast Environment," *Proceedings of the 9th International Conference on Distributed Computing Systems*, June 1989, Newport Beach, California, to appear.

[Spau89]
"Ordered and Reliable Multicast Communication," Technical Report CS-TR-184-88, Department of Computer Science, Princeton University, October 1988, submitted for publication.

[Stae88]
Carl Staelin, File Access Patterns, Technical Report CS-TR-179-88, Department of Computer Science, Princeton University, September 1988.