1995

Adapting Distributed Database Systems for High Availability

Xiangning Liu

Evaggelia Pitoura

Bharat Bhargava
Purdue University, bb@cs.purdue.edu

Report Number:
95-013

ADAPTING DISTRIBUTED DATABASE SYSTEMS FOR HIGH AVAILABILITY

Xiangning Liu
Evaggelia Pitoura
Bharat Bhargava

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD-TR-95-013
February 1995

Adapting Distributed Database Systems for High Availability

Xiangning Liu, Evaggelia Pitoura and *Bharat Bhargava*

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
E-mail: {xl,pitoura,bb}@cs.purdue.edu

Abstract

The availability offered by current data replication and update algorithms varies with dynamically changing conditions which include the network configuration and system load. With dynamic adaptability, systems can switch to an appropriate mechanism to improve performance and availability. In this paper, we present an algorithm to estimate the overall availability of transaction processing in a distributed database system. Our algorithm estimates the system availability of different replication schemas based on transaction access patterns and the availability of system components. A flow graph is used to represent and simplify the computation and the combinatorial model. We illustrate the incorporation of our algorithm into RAID, an adaptable distributed database system, thus permitting the RAID software to dynamically decide to redistribute data or adopt a more suitable update algorithm. The proposed technique for the measurement of availability can also be used during the design phase of a system.

Keywords: Adaptability, Availability, Reliability, Distributed Database, Computer Network, Performance, Data Replication, Transaction Processing,

1 Introduction

Availability and reliability have become central to the performance characterization of a computing system [Shr85, GR93]. A common approach to increasing system availability is the appropriate distribution of data over remote sites. Significant research effort has been expended in the search for new algorithms to achieve this goal, and numerous mechanisms for replica distribution and update have been proposed. Examples include voting [Gif79], dynamic voting [JM87, BGMS89], virtual partition [ASC85, AT89], $\sqrt{N}$ algorithm [Mae85], tree quorums [AA91], and multidimensional voting [CAA91]. [DGMS85] and [HBH95] survey many algorithms and provide guidelines on algorithm selection.

Each of the proposed approaches for replica control performs well under a particular set of circumstances. These varying strengths can best be exploited through the concept of adaptability. RAID, a Robust Adaptable Interoperable Distributed Database System developed at Purdue University, implemented this concept by dynamically switching algorithms and execution mechanisms [BR89b, BR89a, BB90]. In this paper, we describe a method to quantitatively evaluate the availability of a distributed database transaction processing system. This method serves as the mechanism underlying the adaptability feature of RAID, allowing RAID to dynamically choose the most suitable updating algorithm and distribution schema so that high availability is achieved.

Related Work

Previous evaluation methods for availability include the k-out-of-n [BD84, GMK87], event-based reliability [LL86], and multi-state combinatorial [VT93] models. [BP65] also presents a profound mathematical theory of the reliability of general systems. These models estimate the availability of computing systems in general and are not tailored to database transactional systems in particular. Transaction patterns are not taken into consideration and the presented results are therefore insufficiently precise to aid in the achievement of adaptability in transaction processing systems.

A technique that takes transactions into consideration is described in [MRS81]. It presents a method which uses structure vectors to analyze the Markov process of transition between the states corresponding to each vector. System availability may be viewed as the probability that the system is in states which correspond to the successful execution of a transaction. A serious disadvantage of analyses based on Markov chains is that they involve a large state space and result in formidably complicated computations. To reduce this space, many excessive assumptions are made, resulting in inaccurate conclusions. Related work also addressed the quantitative analysis of specific replication algorithms. [JM87] estimates the availability of the dynamic voting algorithm. In [LCS89], the availability of regeneration-based replica control protocols [PNP86] is evaluated. All these approaches use Markov process analysis.

In [BHF92, Hel91] we provided a model for the availability of distributed databases. The analysis was based on the assumption that the availability of all data is independent. In most cases, different data items from a distributed database are placed on a single site, the failure of which renders all data copies at that site inaccessible. Thus, the availability of different data items is not mutually independent. In this paper, we extend previous work by

dropping this assumption, and in addition, we present a method of availability evaluation which we then use to achieve high availability.

Outline of Our Approach

We propose a method to quantitatively calculate the availability of distributed database transaction systems in terms of the availability of component sites and computer network links. The method is based on a combinatorial analytical model. The model and the computation of availability is simplified by using a flow graph to eliminate repeated elements. With this method, different replication schemes and updating algorithms can be analyzed and compared on the basis of availability. This evaluation method is incorporated within RAID to support algorithmic adaptability to changes in the transaction patterns or loads. The evaluation method considers arrival sites and patterns of transactions and is therefore more accurate than previously introduced approaches. The algorithm thus obtained is both simple and efficient.

The remainder of this paper is organized as follows. In Section 2, we introduce the idea of adaptability. In Section 3, we first introduce some related concepts and then present our model and its mathematical foundation. In Section 4, building upon an example illustrative of the basic ideas of the method, the details of the full algorithm are presented. Section 5 provides examples which demonstrate the ability of the algorithm to support adaptability among different replication methods. Finally, in Section 6, we offer conclusions and directions for future work.

2 Adaptability in Distributed Database Systems

In [BR89a], adaptability is classified into four broad categories: structural static, structural dynamic, algorithmic, and heterogeneous. *Structural static adaptability* encompasses software engineering techniques for the development of systems that can be easily maintained and adapted to changing requirements throughout their life cycles. *Structural dynamic adaptability*, also called reconfiguration, refers to the restructuring of a running system in response to failures or performance requirements. *Algorithmic adaptability*, supported as a feature in our prototype RAID system, involves a set of techniques for dynamically shifting between the execution of several algorithms for a module. For instance, a transaction system can change to a new concurrency controller, or a distributed system can adopt a new site failure algorithm. The fourth category is *adaptability for heterogeneity*, which enables computing systems of different types to work together.

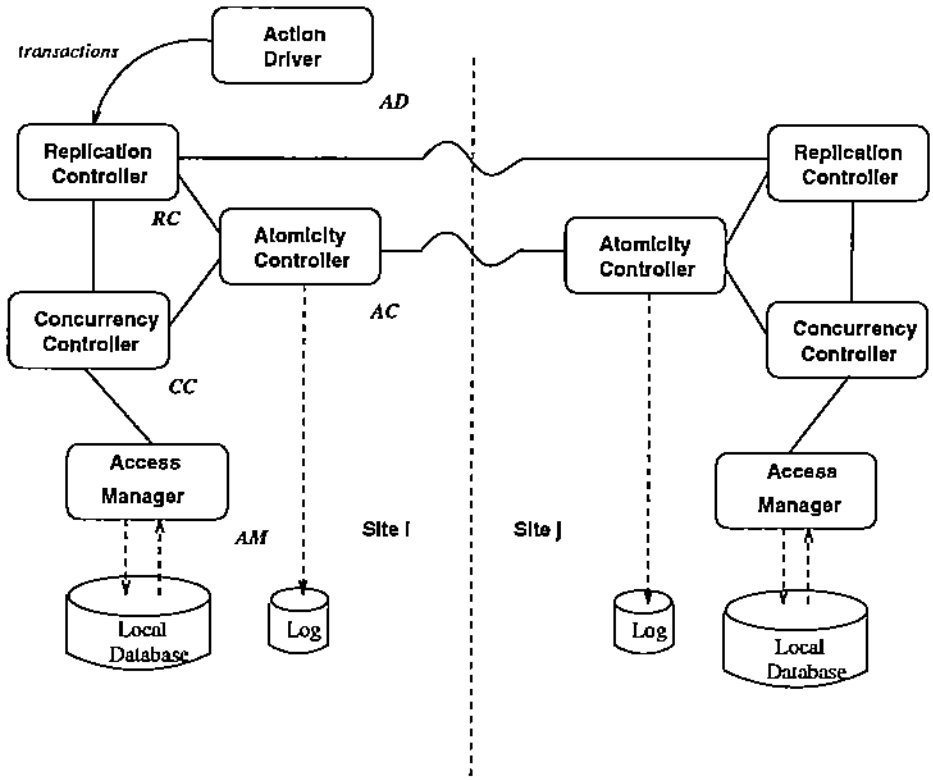

Figure 1: RAID - a Robust Adaptable Interoperable Distributed System

In the present research, we will focus on algorithmic adaptability. This capability is of prime importance in meeting the challenges posed by the great variety of networking conditions and system loads associated with modern distributed systems. Adaptability is particularly crucial in an environment which includes wireless connections, since the network configuration is dynamic and the network connectivity ranges from total disconnection to full connection [PB94]. The same need holds for Wide Area Networks (WANs), where the system load varies substantially depending on such parameters as the time of day and the type of network [ZB93].

Figure 1 presents a schematic illustration of the architecture of RAID. Users submit transactions to the *Action Driver (AD)*, which parses logical actions to procedural actions and submits them to the *Replication Controller (RC)*. The *RC* consults the replication directory and forwards the read or write operations to the local *Concurrency Controller (CC)*, the local *Atomicity Controller (AC)*, or to remote *RCs*. The *CC* controls the consistency of transactions, while the *AC* coordinates update commitment. Finally, the *Access Manager*

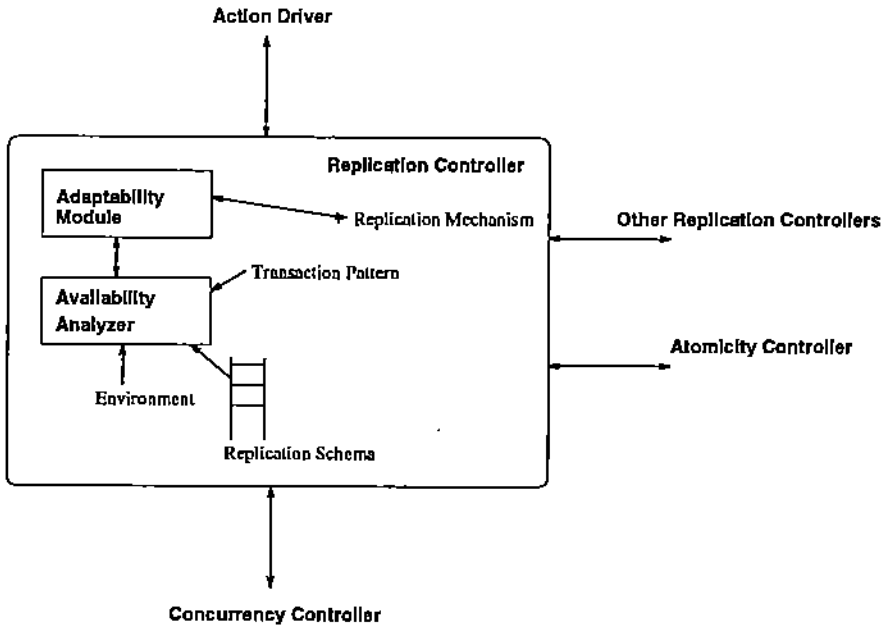

Figure 2: Replication Controller

manages physical access to data.

High availability can be achieved through a multi-step procedure. System availability is periodically evaluated, and on that basis a selection is made at the time of execution from among a repertoire of replication mechanisms. This functionality is incorporated within the RC (Replication Controller), which is illustrated in Figure 2. Inputs to the evaluation process include transaction patterns, the replication mechanisms, and the availability of system components. The output is then used by the adaptability module in the decision to switch to a more appropriate replication schema. In the current RAID configuration, changes between replication mechanisms occur when availability drops below a predefined threshold value.

As an alternate method, the availability of various algorithms under different conditions can be pre-computed and then stored as part of the system. The system can then use this stored information about algorithmic availability in conjunction with the measurement of current condition to intelligently select the most appropriate algorithm.

3 Availability of Distributed Transaction Processing Systems

In this section, we first introduce the concepts of reliability and availability and discuss the subtle differences between them. We then present our model for estimating the availability of a distributed transaction system based on the availability of its components.

3.1 Related Concepts

A system may fail or may inadequately process the working load either because of a component failure or due to a slow response which prevents the system from meeting the required response time. Two concepts are used to describe the frequency of failure occurrences in a system: system availability and reliability. *System availability* is the fraction of the offered load that is processed within an acceptable response time [GR93]. The concept of *system reliability* refers to the probability that the system is not down during a given period and is usually measured by mean time to failure (MTTF). A system may have low reliability but high availability. For example, consider a system that crashes once within a period which averages 10 hours and does not exceed 20 hours and which can be repaired and returns to work in one minute. In this case, the reliability for a one-day period is $1-$*probability of crash in one day* $= 0$ because there is always a daily crash, while the availability is $1 - (1min/600min) \approx 99.83\%$.

Availability may also be described through *availability classes*, each of which is defined by the number of leading nines in the availability figure for a system or model; i.e.:

$$\lfloor log_{10}(\frac{1}{1-A}) \rfloor$$

where A is the system availability. The real number $log_{10}(\frac{1}{1-A})$ is called *availability degree.*

We can further make a differentiation between algorithmic availability and operational availability [BHF92, Hel91]. If a system responds later than the required response time, the system is algorithmically available but not operationally available. *Algorithmic availability* is determined by the nature of the computing algorithm and the status of the involved sites and does not depend on the implementation and processing speed. It is intended to measure the degree of fault-tolerance for component failures provided by replication methods. It defines a measure of merit through which replication control methods can be theoretically evaluated, regardless of any specific implementation of these methods or their system counterparts. *Operational availability* defines the range and conditions under

which a replicated database system is not operable, even in the absence of failures and is performance failure-dependent.

Since operational availability is implementation-dependent, usually it is measured experimentally and not evaluated theoretically [BFHR90]. It remains as part of our future work to implement adaptability based on operational availability. At this point, we will focus on a theoretical evaluation of algorithmic availability. Possibilities for and obstacles to the evaluation of operational availability are also briefly described. The model used here, which has been implemented in the RAID environment, collects statistical data and assumes that the transaction pattern in the near future will remain similar to that of the most recently completed transactions.

Reliability and Availability of Systems

The availability of a system depends on the status of its components, which should be reliable or available during the processing period. Those components which should be reliable are called *continuous components*. For example, the site at which the transaction arrives must be continuously operational during the transaction processing period. Any crash, no matter how short, will cause the transaction to abort. Those components which need only be available (rather than reliable) are called *instantaneous* components. For example, consider a data item at a remote site which crashes after the first read operation arrives but before the transaction sends any additional requests to that site. Later, when the transaction sends a write operation, the site is up again, leaving the transaction completely unaware of the remote crash. Therefore, we require only that a remote site be available when needed.

In this paper, we assume that the home site of a transaction is reliable and that the other sites are available. In other words, all read and write actions are atomic, and there are no interrupts during these actions. To facilitate presentation, only availability will be referred below.

In summary, the assumptions made in this paper are as follows:

1. All transactions are in themselves correct. Deadlock, access conflicts and other concurrency control problems are not considered here. Every transaction can therefore commit if no machine or network crashes.

2. The failure of each site and communication link is independent of any other failure.

3. If a system component is available at the beginning of a transaction, it will be available during the entire lifetime of the transaction.

Assumption 1 is necessary for all the discussion in this paper. In section 4.1, we will discuss the implications of dropping assumptions 2 and 3.

3.2 System Availability in terms of System Components

The availability of a database transaction system is defined as the fraction of the number of transactions that commit within an acceptable time. Let $\Omega = \{t_1, t_2, ..., t_\omega\}$ be the set of all transactions under consideration. Let $\mathcal{A} = (a_{ij})_{n\times n}$ be the matrix of availability of sites $s_1, s_2, ..., s_n$ and the links between them. Here a_{ii} is the availability of site s_i and a_{ij} is the availability of the link between site s_i and site s_j when $i \neq j$. We use $\mathcal{R} = (r_{ij})_{n\times m}$ to describe the data replication scheme; r_{ij} is 1 if and only if data object i is replicated at site s_j; otherwise, it is 0. $\mathcal{P} = (P_{t_1}, ..., P_{t_\omega})$ describes the probability of the arrival of the transactions of Ω. P_{t_i} is the percentage of arrivals of transaction t_i in all possible transactions. U is the access and update mechanism, including such possibilities as ROWA (Read One Write all) and QC (quorum consensus). Our goal is to develop an efficient algorithm to compute the function f_U of the availability provided by U and, on that basis, the transaction system availability of Ω:

$$\begin{aligned} A(\Omega) &= A(\mathcal{A}, \mathcal{R}, U, \mathcal{P}) \\ &= f_U(a_{11}, ..., a_{nn}, r_{11}, ...r_{mn}, p_{t_1}, ..., p_{t_\omega}) \end{aligned}$$

We assume all transactions to be correct and be able to commit if there are no failures in the component systems. Therefore:

$$\begin{aligned} A(\Omega) &= \sum_{t\in\Omega} P_t P(t \textit{ is commited}) \\ &= \sum_{t\in\Omega} P_t P(\textit{all required objects } d_i \textit{ are available for } t) \end{aligned}$$

Since several data items may be located at the same site, they are all either available or unavailable at a given time. The accessibility of these data items is therefore not independent, and produces:

$$A(\Omega) \neq \sum_{t\in\Omega} P_t P_{d_1} P_{d_2} ... P_{d_l}$$

where P_{d_i} is the availability of the data item d_i.

In the following analysis, we assume that the failure of each site is independent. We further assume that, during the lifetime of a transaction, a component that is operational upon

the first access remains so for all successive accesses. In section 4.1, we will discuss the scenario in which the second access may be denied. Each transaction involves the reading and writing of some data. For each read or write operation under access mechanism U, several alternative groups of sites may satisfy a successful execution. Assume σ is the total number of such alternative groups. If group $i, 1 \leq i \leq \sigma$, consists of sites $s_{i1}, s_{i2}, ..., s_{i3}, ..., s_{in_i}$, then the probability that all sites are all available is $P_{G_i} = P(s_{i1})P(s_{i2})...P(s_{in_i})$. Let $A_t(\mathcal{A}, U, \mathcal{R})$ be the system availability for transaction t. As used in the following recursive expression, P_i is the availability calculated from the first i alternative groups of sites. In order to consider network failures, suppose that the transaction t arrives at site k. Let P_{ki} be the probability that all sites in group i are reachable from site k. Then we have

$$\begin{aligned} P_{i+1} &= P_i + (1 - P_i)P_{ki}P(s_{i1})P(s_{i2})...P(s_{i+1n_{i+1}}) \\ P_1 &= P_{k1}P(s_{11})P(s_{12})...P(s_{1n_1}) \end{aligned}$$

If all links are independent, we can write $P_{ki} = a_{ki_1}a_{ki_2}...a_{ki_{n_i}}$ where a_{ki_j} is the availability of a link between k and i_j. Then we have:

$$\begin{aligned} P_{i+1} &= P_i + (1 - P_i)P(s_{i1})P(s_{i2})...P(s_{i+1n_{i+1}})a_{ki_1}a_{ki_2}...a_{ki_{n_i}} \\ P_1 &= P(s_{11})P(s_{12})...P(s_{1n_1})a_{k1_1}a_{k1_2}...a_{k1_{n_1}} \quad for\ 1 \leq i \leq \sigma \\ A_t(\mathcal{A}, U, \mathcal{R}) &= P_\sigma \end{aligned}$$

and

$$\begin{aligned} A(\Omega) &= A(\mathcal{A}, \mathcal{R}, U, \mathcal{P}) \\ &= \sum_{t \in \Omega} P(t)A_t(\mathcal{A}, U, \mathcal{R}) \qquad (3.1) \\ &= \sum_{t \in \Omega} P(t)P_\sigma \end{aligned}$$

We will not discuss here the case of dependent network links. This situation is similar to the discussion to be presented below regarding the dropping of assumption 3, i.e., allowing components to fail in the middle of a transaction.

In the next section, we will present a method to efficiently calculate $A_t(\mathcal{A}, U, \mathcal{R})$ and from that point, the calculation of $A(\Omega)$ is straightforward from formula (3.1).

4 An Efficient Availability Calculation Method

In this section, we present a method for calculating availability which is based on the construction of a flow graph for the availability of each transaction. We present an algorithm

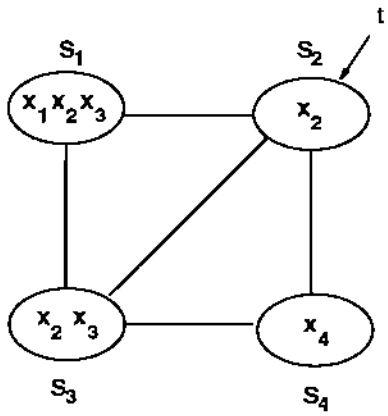

Figure 3: Data items are at sites s_1, s_2, s_3, and s_4.

for traversing this graph and computing availability and then discuss its complexity.

4.1 Flow Graph

Our algorithm can best be illustrated through the presentation of a simple concrete example. We use an example initially introduced in [MRS81].

Consider four data objects x_1, x_2, x_3, and x_4 on four sites s_1, s_2, s_3, and s_4 as shown in Figure 3 and assume they have the following distribution:

- site s_1 contains $\{x_1, x_2, x_3\}$;
- site s_2 contains $\{x_2\}$;
- site s_3 contains $\{x_2, x_3\}$;
- site s_4 contains $\{x_4\}$.

Let us consider a transaction

$$t: \ r(x_3)r(x_4)w(x_2)$$

which arrives at site s_2 which, as the home site, must be operable for t. Operation $r(x_3)$ requires either site s_1 or s_3 to be accessible; i.e., it requires that the link $2 \rightarrow 1$ and site s_1 or the link $2 \rightarrow 3$ and site s_3 be available. Operation $r(x_4)$ requires that site s_4 be available. For operation $w(x_2)$, since data item x_2 is replicated on sites s_1, s_2, and s_3, an update protocol which requires two copies as a majority, will thus demand that at least two links and sites from the set $\{s_1, s_2, s_3\}$ be available. These requirements are all reflected by the flow graph in Figure 4.

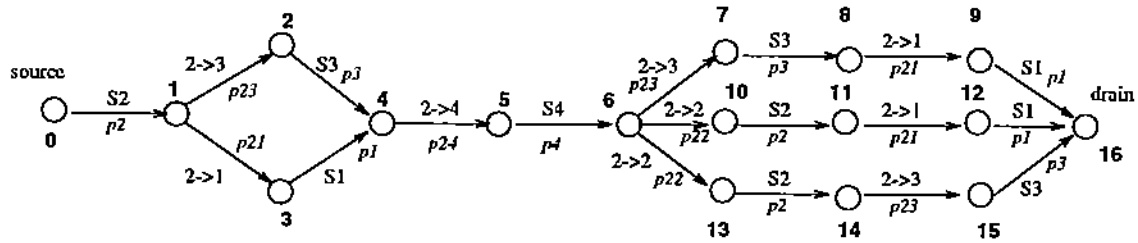

Figure 4: Flow graph of transaction t: $r(x_3)r(x_4)w(x_2)$

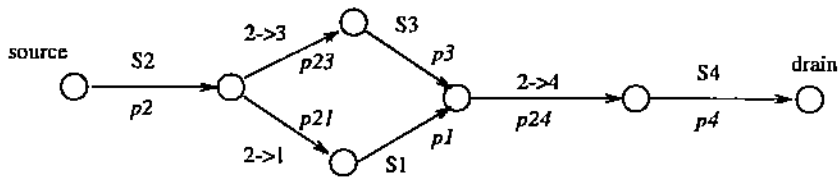

Figure 5: Simplified flow graph of Figure 4

In [MRS81], an up-and-down model was used to construct a structure vector space which has 128 states. Therefore, the sufficient and necessary condition for the availability of the system to transaction t is the existence of a path from the source to the drain of the search graph. The system was then analyzed as a Markov process with transitions between each of these 128 states. While this idea is simple, the large number of states renders a computation of availability prohibitive. In practice, this up/down structure vector approach is difficult to apply.

We have observed that, if we associate the availability of sites or network links with the edges of the search graph, we can produce a new *flow graph*. Intuitively, while the search graph of [MRS81] used 0/1 value logic to construct a structure vector to search for an all-1 path from the source to the drain, our flow graph employs fuzzy logic [Zad71] to describe the process and to express availability as the probability of moving from the source to the drain. Such a flow graph effectively and simply depicts the mathematical formula given in section 3. To provide a physical metaphor, if a flow of one unit of water leaves the source, each edge on a path lets pass an amount equal to the availability of the corresponding component; the amount of water that finally arrives at the drain is the system availability.

If we remove assumption 3, for each component i, we assume that the availability for the second access is p_i' if the first access succeeds and p_i'' if the first access fails. In this instance, we cannot simplify the flow graph of Figure 4, and all paths must be considered.

Given a path, such as $0 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 16$ in Figure 4, we can arrive at the probability that the system can pass through this path by multiplying the availability of all edges of the path. We have

$$P_1 = p_2 p_{23} p_3 p_{24} p_4 p'_{23} p'_3 p_{21} p_1$$

In general, we can express the availability of the first i+1 paths recursively as

$$\begin{aligned} P_{i+1} &= P_i + (1 - P_i) \prod_{path\ i+1} p_j^{(i+1)} \\ P_1 &= \prod_{path\ 1} p_j^{(1)} \\ A_t(\mathcal{A}, U, \mathcal{R}) &= P_\omega \end{aligned}$$

where ω is the total number of paths, $p_j^{(i+1)}$ is p_j if it is the first access, p'_j if it is the second access and the first access is on the path, and p''_j if there is an access to that edge in the first $j-1$ paths.

Given assumptions 2 and 3 above, the search graph in Figure 4 can be simplified to Figure 5 using algorithm 4.1 to be presented below.

From Figure 5, we can easily compute system availability with regard to transaction t. If all components are of class 2; i.e., $p_{ij} = 0.99$ for all i, j, we get

$$\begin{aligned} A &= p_2(p_{23}p_3 + (1 - p_{23}p_3)p_{12}p_1)p_{24}p_4 \\ &= 0.99(0.99 * 0.99 + (1 - 0.99 * 0.99) * 0.99 * 0.99) * 0.99 * 0.99 \\ &= 0.9699. \end{aligned}$$

While this approach is effective in determining algorithmic availability, it cannot be applied to operational availability. Within the processing of any transaction, different rates of speed of processing component operations may result in the timely commitment of the entire transaction despite the slow processing of some operations. This amortization of processing time cannot be expressed in the flow graph.

4.2 Description of the Algorithm

In the previous section, we used an example to elucidate the basic ideas underlying the availability evaluation algorithm. In this section, we shall describe this algorithm in general

terms. For a given transaction t, component availability matrix $\mathcal{A}$, update method U, and replication schema $\mathcal{R}$, a flow graph can be constructed in a straightforward manner in which each edge is associated with the availability of the corresponding component. For each read or write operation, one out of a set of site and link groups must be operational, the edges of which are connected into a path. The paths for the set of the site and link groups of an operation start and end at the same nodes, forming a subgraph corresponding to that operation. To describe the entire transaction, all the subgraphs are connected to form the flow graph.

The flow graph can be simplified if assumptions 1, 2, and 3 are all enforced. We start a search from the source, with each step involving a subgraph for an operation and any repeated edges consolidated by merging subgraphs. We can then calculate the system availability $A_t(\mathcal{A}, U, \mathcal{R})$ for a given transaction t from this simplified graph. Using formula (3.1) in section 3.2, we arrive at the availability $A(\Omega)$ of the system. It can be proved that the flow graph can always be simplified to one without repeated edges. An acyclic subgraph which starts at one node and ends at one node is called a *sub-flow graph*. Algorithm 1 outlines the flow graph simplification process. Figure 6 provides two examples of sub-flow graph merging. The subgraphs on the left are merged to become the subgraphs on the right. Algorithm 2 recursively computes the availability of a sub-flow graph. On line 4 or 6, any A_j may be computed by the algorithm 2 recursively.

Algorithm 1 : Simplify flow graph
Input: A flow graph
Output: Simplified flow graph

```
1   begin
2       loop through the subgraphs of all operations
3           if an edge of the subgraph appears earlier in one of the paths then
4               merge the two subgraphs;
5           end
6       end
7   end
```

Algorithm 2 : Calculate availability for a transaction
Input: Simplified sub-flow graph
Output: System availability

```
1    begin
2        A=1;
```

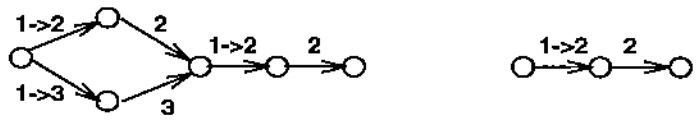

Figure 6: The graphs on the left are simplified to the graphs on the right

```
3       loop through the subgraphs of the input graph
4           initialize  B_1 = A_1 A_2 ... A_n of the first path of the subgraph
5           loop for every path i of the subgraph
6              B_i+1 = B_i + (1-B_i)* A_i1 A_i2 ... A_in_i
7           end
8           A = A * Bi
9       end
10  end
```

The algorithms presented above are obviously simple. More complex algorithms can be developed on the basis of the ideas presented in section 4.1 to address a scenario in which assumptions 2 and 3 are dropped. Due to space limitations this development will not be pursued here.

4.3 Complexity

Time complexity can be simply expressed as $C * l * n^2$ for the ROWA protocol, where C is a constant, l is the length of the transaction, and n is the number of sites. The time complexity is proportional to the square of the number e of edges of the flow graph; i.e.,

$$T = C * e^2$$

In comparison, the structure vector [MRS81] approach involves a time complexity of $C * 2^e$.

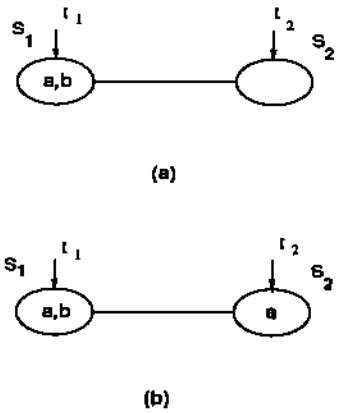

Figure 7: Data items at two sites S_1 and S_2

In theory, the highest possible cost could be $C_1 * l * n^{C_2 n}$, since for any operation there can be as many as $n^{C_2 n}$ alternative groups of sites satisfying the site operation. In practice, many network links and sites often have identical properties and, by symmetry, the computation can be greatly reduced.

5 Adapting for High Availability

In this section, we illustrate the use of availability estimation to facilitate the selection among schemes for replica distribution and control. These examples are meant to be illustrative of the applicability of our measurements in adapting to appropriate schemes and should not be considered to be exhaustive.

5.1 Dynamic Data Replication

Figure 7 depicts an illustrative case with two sites, S_1 and S_2. Two transactions $t_1 = \{r(b)w(a)\}$ and $t_2 = \{r(a)\}$ have a probability of arrival p_1 at site S_1 and p_2 at site S_2. In Figure 7(a), data items a and b are both at site S_1, with no data items at site S_2. In Figure 7(b), a is also replicated at site S_2. Configuration (a) has availability

$$A_1 = p_1 a_1 + p_2(a_2 a_{12} a_1)$$

Configuration (b) has availability

$$A_2 = p_1(a_1 a_{12} a_2) + p_2 a_2$$

Suppose site S_1 has availability degree 4, and site S_2 and the link have availability degree 3, i.e. $a_1 = 0.9999$ and $a_2 = a_{12} = 0.999$. The alteration $p_1 = 1 - p_2$ produces the graph

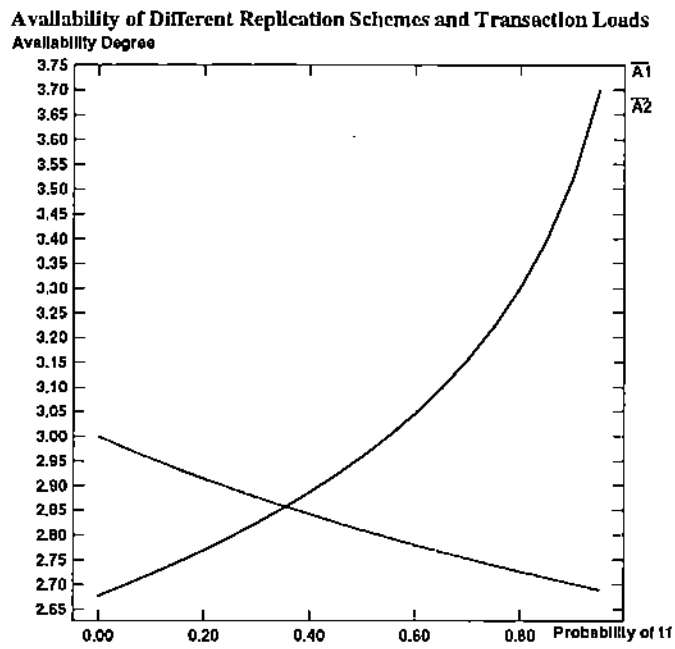

Figure 8: Availability for different transaction arrival probabilities

in Figure 8, which indicates that when p_1 is above 35%, better availability is arrived at by not replicating a. When p_2 is above 65%, higher availability is obtained by replicating a to site S_2.

5.2 Dynamic Selection of Updating Algorithms

Figure 9 shows a configuration of a distributed database system. Data item d is distributed on all sites $\{S_1, S_2, S_3\}$. Suppose there are transactions $t_1 = \{r(d)w(d)\}$ and $t_2 = \{r(d)\}$, with t_1 arriving only at site S_1 at rate p_1, t_2 arriving at S_2 and S_3 at rates p_2 and p_3.

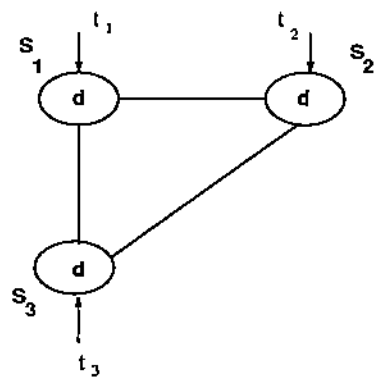

Figure 9: Distribution of data item d on sites S_1, S_2, and S_3

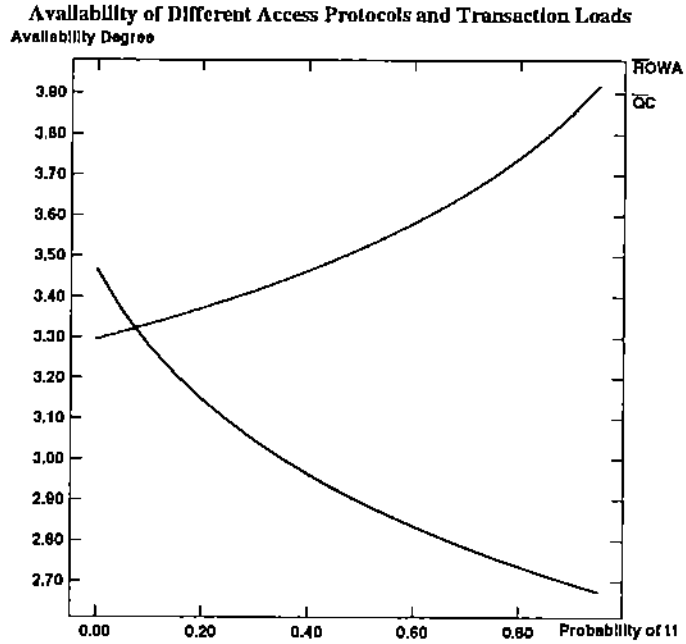

Figure 10: Availability of ROWA and Quorum Consensus Algorithms

By the Read-One-Write-All (ROWA) algorithm, a read operation needs only one data copy to be available, while a write operation requires that all data copies be available. We then have availability:

$$A_1 = p_1 a_1 a_2 a_3 a_{12} a_{13} + p_2 a_2 + p_3 a_3.$$

By the quorum consensus algorithm, the number of available data copies must exceed the quorum or operation threshold. If both read and write quorums are two, we have

$$\begin{aligned} A_2 = {} & p_1 a_1 (a_2 a_{12} + (1 - a_2 a_{12}) a_3 a_{13}) + p_2 a_2 (a_1 a_{21} + (1 - a_1 a_{21}) a_3 a_{23}) \\ & + p_3 a_3 (a_1 a_{31} + (1 - a_1 a_{31}) a_2 a_{32}) \end{aligned}$$

Suppose all $a_1 = a_{13} = 0.9999$, $a_3 = 0.99999$, and $a_2 = a_{ij} = 0.999$ for every i and j. In other words, S_3 is highly reliable and S_2 is not reliable. Let $p_3 = 2 * p_2$ indicating that the majority of access is by S_3. We then arrive at the graph in Figure 10, from which we conclude that, when $p_1 > 7\%$, the quorum consensus algorithm produces better availability than ROWA.

5.3 Voting and Dynamic Voting Algorithms

Suppose there are five sites $S_1, S_2, ..., S_5$ with data item d replicated at all sites. Assume further that all sites have the same availability a and that all links have availability b..

The static voting algorithm requires that a majority of total sites be available. In this case, there must be 3, 4, or 5 sites available. Let X_i denote system availability with at least i sites. With 5 sites available, we have $X_5 = a^4 * b^4$, with at least 4 sites, we have $X_4 = X_5 + \frac{4}{5}(1 - X_5)a^4b^3$, and with at least 3 sites, we have the following system availability:

$$\begin{aligned} A_1 &= p_1 X_3 + p_2 a \\ &= p_1(X_4 + \frac{3}{5}(1 - X_4)a^3b^2) + p_2 a \\ &= p_1(a^4 * b^4 + \frac{4}{5}(1 - a^4 * b^4)a^4b^3 + \frac{3}{5}(1 - (a^4 * b^4 + \frac{4}{5}(1 - a^4 * b^4)a^4b^3))a^3b^2)) + p_2 a \end{aligned}$$

The dynamic voting algorithm requires that a majority of sites in the last majority partition be available. Even if this partition formed a majority by joining repaired sites an update is still not allowed in this partition unless all sites are available. Let X_i denote system availability with at least i sites in a majority partition; if the repair rate is much higher than failure rate, then $X_5 = a^4 * b^4$, $X_4 = X_5 + \frac{4}{7}\frac{4}{5}(1 - X_5)a^4b^3$, $X_3 = X_4 + \frac{1}{2}\frac{3}{5}(1 - X_4)a^3b^2$ $X_2 = X_3 + \frac{1}{3}\frac{2}{5}(1 - X_3)a^2b$. If, in addition to the probability p_1 of the update's transaction arrival, there is another read transaction which arrives at all sites with a total probability p_2, then

$$A_2 = p_1 X_2 + p_2 a$$

Let sites have availability degree 5 and links have 4, i.e. a=0.99999, b=0.9999; then we arrive at the graph in Figure 11. From this graph, it is evident, that the dynamic voting algorithm offers better availability that the static voting algorithm.

6 Conclusions

In this paper, we have presented a flow graph method to simplify the evaluation of algorithmic availability. Our algorithm is tailored to transaction processing systems. It can be used to theoretically compare the respective merits of different replication protocols. We find that using the flow graph approach to evaluate transaction system availability is simple, efficient and effective.

A number of examples have been presented to illustrate the role played by the proposed algorithm in our prototype distributed database system to support dynamic adaptability. Such adaptability enchances system availability by switching to a more appropriate replica update algorithm or choosing a better data distribution schema. The proposed technique

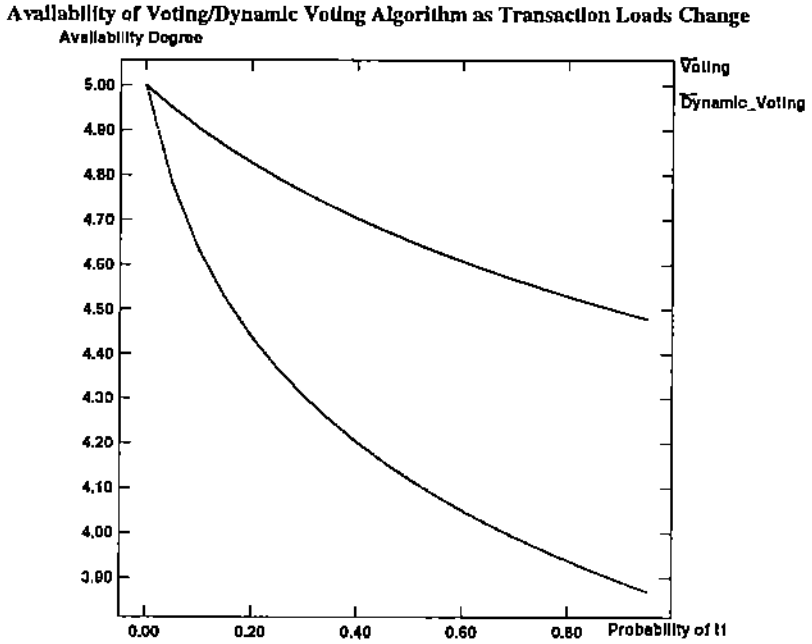

Figure 11: Static and dynamic voting algorithms

can also be used during the design phase of a system to choose the most appropriate algorithm.

Future studies will address the following issues.

1. Analytic models about evaluation of operational availability and their integration to RAID will be considered.

2. Failures of network links are often dependent and we must account for that in our model.

3. Experimental work will be conducted to verify our theoretical estimations.

References

[AA91] R. Agrawal and A. El Abbadi. An Efficient and Fault-Tolerant Solution for Distributed Multual Exclusion. *ACM Transaction on Computer Systems*, 9(1):1–20, February 1991.

[ASC85] A. El Abbadi, D. Skeen, and F. Christian. An efficient fault tolerant protocol for replicated data management. In *Proc of the 4th ACM Symposium on Principles of Database Systems*, pages 215–229, Portland, Oregon, March 1985.

[AT89] A. El Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. *ACM Transaction on Database Systems*, 14(2):264–290, June 1989.

[BB90] B. Bhargava and S. Browne. Adaptable recovery using dynamic quorum assignments. In *Proc. 16th Int'l Conf. on Very Large Data Bases*, Brisbane, Australia, August 1990.

[BD84] R. E. Barlow and J. D. Day. Computing k-out-of-n System Reliability. *IEEE Transaction on Reliability*, 33(4):322–323, 1984.

[BFHR90] B. Bhargava, K. Friesen, A. Helal, and J. Riedl. Adaptability experiments in the RAID distributed database system. In *Proceedings of the Ninth IEEE Symposium on Reliable Distributed Systems*, Huntsville, AL, October 1990.

[BGMS89] D. Barbara, H. Garcia-Molina, and A. Spauster. Increasing Availability under Mutual Exclusion Constraints with Dynamic Voting Reassignment. *ACM Transaction on Computer Systems*, 7(4):394–426, November 1989.

[BHF92] B. Bhargava, A. Helal, and K. Friesen. Analyzing availability of replicated database systems. *International Journal of Computer Simulation*, 1:393–418, 1992.

[BP65] R. E. Barlow and F. Proschan. *Mathematical Theory of Reliability.* Wiley, New York, 1965.

[BR89a] B. Bhargava and J. Riedl. A Model for Adaptable Systems for Transaction Processing. *IEEE Transactions on Knowledge and Data Engineering*, 1(4), December 1989.

[BR89b] B. Bhargava and J. Riedl. The RAID Distributed Database System. *IEEE Transactions on Software Engineering*, 15(6), June 1989.

[CAA91] S. Y. Chueng, M. H. Ammar, and A. Ahamad. Multidimensional Voting. *ACM Transaction on Computer Systems*, 9(4):399–431, November 1991.

[DGMS85] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3), September 1985.

[Gif79] D. K. Gifford. Weighted Voting for Replicated Data. In *Proc of the 7th Symposium on Operating Systems Principles*, pages 150–162, Asilomar, California, December 1979.

[GMK87] H. Garcia-Molina and J. Kent. Performance Evaluation of Reliable Distributed Systems. In B. Bhargava, editor, *Concurrency Control and Reliability in Distributed Systems*, chapter 16, pages 454–488. Van Nortrand Reinhold, 1987.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, Inc., 1993.

[HBH95] A. Helal, B. Bhargava, and A. Heddaya, editors. *Advanced Replication Techniques in Distributed Computing Systems.* IEEE Computer Science Press, 1995. (to appear).

[Hel91] A. Helal. Experimental Analysis of Replication in Distributed Systems. *Ph. D Thesis, Purdue University*, May 1991.

[JM87] S. Jajodia and D. Mutchler. Dynamic Voting. In *Proc. of the 16th SIGMOD Conference*, San Francisco, May 1987.

[LCS89] D. D. E. Long, J. L. Carroll, and K. Steward. The Reliability of Regeneration-Based Replica Control Protocols. *IEEE Transactions on Computers*, 38(12), Dec 1989.

[LL86] Y. F. Lam and V. O. K. Li. Reliability Modeling and Analysis of Communication Networks with Dependent Failures. *IEEE Transactions on Communications*, 34(1), January 1986.

[Mae85] M. Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transaction on Computer Systems*, 3(2):145–159, May 1985.

[MRS81] G. Martell, B. Ronchetti, and F. Schreiber. Availability Evaluation in Distributed Database Systems. *Performance Evaluation*, 1(3):201–211, 1981.

[PB94] E. Pitoura and B. Bhargava. Building Information Systems for Mobile Environments. In *Proc of 3rd Int. Conf. on Information and Knowledge Management*, Maryland, November 1994.

[PNP86] C. Pu, J. D. Noe, and A. Proudfoot. Regenaration of Replicated Objects: A Technique and its Eden Implementation. In *Proc of 2nd Int. Conf. Data Eng.*, pages 175–187, Los Angeles, CA, 1986.

[Shr85] S. K. Shrivastava, editor. *Reliable Computer Systems*. Springer-Verlag, 1985.

[VT93] M. Veeraraghavan and K. Trivedi. An Approach for Combinatorial Performance and Availability Analysis. In *Proceedings of 12th Symposium on Reliable Distributed Systems*, pages 24–33, Princeton, New Jersey, October 1993.

[Zad71] L. A. Zadeh. Similarity Relations and Fuzzy Orderings. *Information Sciences*, pages 177–200, March 1971.

[ZB93] Yongguang Zhang and Bharat Bhargava. WANCE: A Wide Area Network Communication Emulation System. In *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 40–45, Princeton, New Jersey, October 1993.