PM-1 3½”x4” PHOTOGRAPHIC MICROCOPY TARGET

NBS 1070a ANSI/ISO #2 EQUIVALENT

1.0

1.1

1.25

1.4

1.6

1.8

2.0

2.2

2.5

2.8

PRECISION™ RESOLUTION TARGETS
NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

Canada
1.2.2 Assumptions

In this thesis a distributed system refers to a collection of nodes interconnected by a LAN. We view faults as node crashes, and we assume that node failures are detectable. A file is viewed as a sequence of blocks (pages) of data. Blocks constituting a file are stored at some nodes. Blocks stored at a node are a consecutive subset of the file. Also in order maintain full distribution of the data, we assume that live nodes hold at least one block of data.

1.2.3 The Results

The main contribution of this thesis is the development of efficient solutions to the problem of continuously guaranteeing the availability of the files under four different system requirements. Specifically we study the following cases:

1) *Basic* case: no restrictions imposed on the system.
2) *Balanced* case: balanced load requirement.
3) *Ordered* case: ordered data requirement.
4) *Balanced Ordered* case: balanced load and ordered data requirements.

For each of these cases, we develop a mechanism that keeps a $K$-tolerant system continuously $K$-tolerant after the occurrence of up to $K$ faults in the system and the recovery of an arbitrary number of nodes. Each mechanism consists of a Replication Strategy, a Failure Protocol, and a Recovery Protocol.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWI</td>
<td>Set of Failed Working Indices</td>
</tr>
<tr>
<td>STM</td>
<td>Send to manager</td>
</tr>
<tr>
<td>STN</td>
<td>Send to node</td>
</tr>
<tr>
<td>RFM</td>
<td>Receive from manager</td>
</tr>
</tbody>
</table>

x
<table>
<thead>
<tr>
<th>Subject Categories</th>
<th>Subject Term</th>
<th>Subject Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>THE HUMANITIES AND SOCIAL SCIENCES</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communications and the Arts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>0729</td>
<td></td>
</tr>
<tr>
<td>Art History</td>
<td>0737</td>
<td></td>
</tr>
<tr>
<td>Cinema</td>
<td>0700</td>
<td></td>
</tr>
<tr>
<td>Dance</td>
<td>0738</td>
<td></td>
</tr>
<tr>
<td>Fine Arts</td>
<td>0737</td>
<td></td>
</tr>
<tr>
<td>Information Science</td>
<td>0733</td>
<td></td>
</tr>
<tr>
<td>Journalism</td>
<td>0791</td>
<td></td>
</tr>
<tr>
<td>Library Science</td>
<td>0799</td>
<td></td>
</tr>
<tr>
<td>Mass Communications</td>
<td>0708</td>
<td></td>
</tr>
<tr>
<td>Music</td>
<td>0743</td>
<td></td>
</tr>
<tr>
<td>Speech Communication</td>
<td>0459</td>
<td></td>
</tr>
<tr>
<td>Theater</td>
<td>0465</td>
<td></td>
</tr>
<tr>
<td><strong>EDUCATION</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General</td>
<td>0515</td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>0514</td>
<td></td>
</tr>
<tr>
<td>Adult and Continuing</td>
<td>0516</td>
<td></td>
</tr>
<tr>
<td>Agricultural</td>
<td>0517</td>
<td></td>
</tr>
<tr>
<td>Art</td>
<td>0273</td>
<td></td>
</tr>
<tr>
<td>Bilingual and Multicultural</td>
<td>0282</td>
<td></td>
</tr>
<tr>
<td>Business</td>
<td>0279</td>
<td></td>
</tr>
<tr>
<td>Community College</td>
<td>0275</td>
<td></td>
</tr>
<tr>
<td>Curriculum and Instruction</td>
<td>0275</td>
<td></td>
</tr>
<tr>
<td>Early Childhood</td>
<td>0518</td>
<td></td>
</tr>
<tr>
<td>Elementary</td>
<td>0524</td>
<td></td>
</tr>
<tr>
<td>Finance</td>
<td>0279</td>
<td></td>
</tr>
<tr>
<td>Guidance and Counseling</td>
<td>0527</td>
<td></td>
</tr>
<tr>
<td>Health</td>
<td>0680</td>
<td></td>
</tr>
<tr>
<td>Higher</td>
<td>0279</td>
<td></td>
</tr>
<tr>
<td>History of</td>
<td>0279</td>
<td></td>
</tr>
<tr>
<td>Home Economics</td>
<td>0278</td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td>0521</td>
<td></td>
</tr>
<tr>
<td>Language and Literature</td>
<td>0280</td>
<td></td>
</tr>
<tr>
<td>Mathematics</td>
<td>0522</td>
<td></td>
</tr>
<tr>
<td>Music</td>
<td>0279</td>
<td></td>
</tr>
<tr>
<td>Philosophy of</td>
<td>0968</td>
<td></td>
</tr>
<tr>
<td>Physical</td>
<td>0523</td>
<td></td>
</tr>
<tr>
<td>Psychology</td>
<td>0525</td>
<td></td>
</tr>
<tr>
<td>Reading</td>
<td>0535</td>
<td></td>
</tr>
<tr>
<td>Religious</td>
<td>0527</td>
<td></td>
</tr>
<tr>
<td>Sciences</td>
<td>0714</td>
<td></td>
</tr>
<tr>
<td>Secondary</td>
<td>0533</td>
<td></td>
</tr>
<tr>
<td>Social Sciences</td>
<td>0554</td>
<td></td>
</tr>
<tr>
<td>Sociology</td>
<td>0540</td>
<td></td>
</tr>
<tr>
<td>Special</td>
<td>0529</td>
<td></td>
</tr>
<tr>
<td>Teacher Education</td>
<td>0530</td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td>0710</td>
<td></td>
</tr>
<tr>
<td>Tests and Measurements</td>
<td>0288</td>
<td></td>
</tr>
<tr>
<td>Vocational</td>
<td>0747</td>
<td></td>
</tr>
<tr>
<td><strong>LANGUAGES, LITERATURE AND LINGUISTICS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>0679</td>
<td></td>
</tr>
<tr>
<td>Literary</td>
<td>0294</td>
<td></td>
</tr>
<tr>
<td>Linguistics</td>
<td>0295</td>
<td></td>
</tr>
<tr>
<td>Medieval</td>
<td>0297</td>
<td></td>
</tr>
<tr>
<td>Modern</td>
<td>0298</td>
<td></td>
</tr>
<tr>
<td>Modern</td>
<td>0298</td>
<td></td>
</tr>
<tr>
<td>Nunan</td>
<td>0316</td>
<td></td>
</tr>
<tr>
<td>American</td>
<td>0591</td>
<td></td>
</tr>
<tr>
<td>Asian</td>
<td>0305</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>0593</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>0593</td>
<td></td>
</tr>
<tr>
<td>Germanic</td>
<td>0311</td>
<td></td>
</tr>
<tr>
<td>Latin American</td>
<td>0312</td>
<td></td>
</tr>
<tr>
<td>Roman</td>
<td>0313</td>
<td></td>
</tr>
<tr>
<td>Slavic and East European</td>
<td>0314</td>
<td></td>
</tr>
<tr>
<td><strong>PHILOSOPHY, RELIGION AND THEOLOGY</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Philosophy</td>
<td>0422</td>
<td></td>
</tr>
<tr>
<td>Religious</td>
<td>0527</td>
<td></td>
</tr>
<tr>
<td>General</td>
<td>0318</td>
<td></td>
</tr>
<tr>
<td>Biblical Studies</td>
<td>0321</td>
<td></td>
</tr>
<tr>
<td>Clergy</td>
<td>0319</td>
<td></td>
</tr>
<tr>
<td>History of Religion</td>
<td>0321</td>
<td></td>
</tr>
<tr>
<td>Philosophy of Religion</td>
<td>0322</td>
<td></td>
</tr>
<tr>
<td>Theology</td>
<td>0469</td>
<td></td>
</tr>
<tr>
<td><strong>SOCIAL SCIENCES</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anthropology</td>
<td>0323</td>
<td></td>
</tr>
<tr>
<td>Archaeology</td>
<td>0324</td>
<td></td>
</tr>
<tr>
<td>Cultural</td>
<td>0326</td>
<td></td>
</tr>
<tr>
<td>Religious</td>
<td>0527</td>
<td></td>
</tr>
<tr>
<td>Business Administration</td>
<td>0310</td>
<td></td>
</tr>
<tr>
<td>Accounting</td>
<td>0272</td>
<td></td>
</tr>
<tr>
<td>Banking</td>
<td>0770</td>
<td></td>
</tr>
<tr>
<td>Management</td>
<td>0454</td>
<td></td>
</tr>
<tr>
<td>Marketing</td>
<td>0338</td>
<td></td>
</tr>
<tr>
<td>Canadian Studies</td>
<td>0385</td>
<td></td>
</tr>
<tr>
<td>Economics</td>
<td>0501</td>
<td></td>
</tr>
<tr>
<td>Agricultural</td>
<td>0502</td>
<td></td>
</tr>
<tr>
<td>American</td>
<td>0503</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>0504</td>
<td></td>
</tr>
<tr>
<td>French</td>
<td>0505</td>
<td></td>
</tr>
<tr>
<td>German</td>
<td>0506</td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>0507</td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>0508</td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>0509</td>
<td></td>
</tr>
<tr>
<td>Labor</td>
<td>0510</td>
<td></td>
</tr>
<tr>
<td>Theory</td>
<td>0511</td>
<td></td>
</tr>
<tr>
<td>Folklore</td>
<td>0358</td>
<td></td>
</tr>
<tr>
<td>Geography</td>
<td>0351</td>
<td></td>
</tr>
<tr>
<td>Geology</td>
<td>0351</td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>0358</td>
<td></td>
</tr>
<tr>
<td>History</td>
<td>0578</td>
<td></td>
</tr>
</tbody>
</table>

| **THE SCIENCES AND ENGINEERING** | | |
| Biological Sciences | | |
| Agriculture | 0473 | |
| Agronomy | 0285 | |
| Animal and Plant | 0475 | |
| Animal | 0476 | |
| Animal Science and Technology | 0397 | |
| Forestry | 0478 | |
| Plant Pathology | 0480 | |
| Plant Physiology | 0817 | |
| Range Management | 0777 | |
| Wood Technology | 0746 | |
| **Biology** | | |
| General | 0306 | |
| Anatomy | 0787 | |
| Biochemistry | 0308 | |
| Botany | 0309 | |
| Cell | 0370 | |
| Ecology | 0329 | |
| Entomology | 0353 | |
| Genetics | 0359 | |
| Immunology | 0793 | |
| Microbiology | 0410 | |
| Molecular | 0307 | |
| Neuroscience | 0317 | |
| Oceanography | 0416 | |
| Physiology | 0433 | |
| Radiation | 0821 | |
| Veterinary Science | 0778 | |
| Zoology | 0477 | |
| **Biophysics** | | |
| General | 0786 | |
| Medical | 0760 | |
| **EARTH SCIENCES** | | |
| Biogeochemistry | 045 | |
| Geology | 0986 | |
The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,
Continuously K-Tolerant Systems: Schemes for Maintaining full Availability in
Distributed File Systems.

submitted by Georges Coucoupoulos, B.A.,
in partial fulfilment of the requirements for the degree of
Master of Computer Science

Thesis Supervisor

Director, School of Computer Science
Carleton University

Date
Abstract

In this thesis, we consider the problem of maintaining full availability in distributed file systems in presence of failures.

The system is $K$-tolerant if the entire file is available after the simultaneous failure of up to $K$ nodes. $K$-tolerance is achieved by data replication.

The problem is that a $K$-tolerant system, after failures, is no longer $K$-tolerant; that is, subsequent failures may compromise the availability of the file.

We are concerned with the design of systems which continuously guarantee data availability. A continuously $K$-tolerant system is one which, starting from a $K$-tolerant configuration, after the failure of up to $K$ nodes, reconfigures itself so to remain $K$-tolerant.

We first propose replication strategies that achieve $K$-tolerance with minimum amount of replication.

We then present several Mechanisms that keep the system continuously $K$-tolerant; each mechanism consists of a Replication Strategy, a Failure Protocol and a Recovery Protocol. The difference between these mechanisms depends on the requirements (e.g., data balancing, ordered data) imposed on the system. All proposed mechanisms are fully distributed. We analyze the amount of communication required by the mechanisms and prove that they are highly efficient.
Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Nicola Santoro, for his guidance and valuable insight through this research. His support and patience were invaluable in the preparation of this thesis.

I would also like to thank my wife Zoë for her understanding, support and encouragement.

Last but not least, I would like to thank the staff of the Computer Science School for their assistance.
# Table of Contents

List of Figures ........................................................................................................... viii  
List of Symbols .......................................................................................................... ix  

Chapter 1  Introduction .............................................................................................. 1  
1.1   The Framework ................................................................................................. 1  
1.2   The Problem and the Results ........................................................................... 3  
    1.2.1  The Problem ............................................................................................ 3  
    1.2.2  Assumptions .......................................................................................... 5  
    1.2.3  The Results ............................................................................................ 5  
1.3   Thesis Organization ......................................................................................... 7  

Chapter 2  Current Approaches to Managing K-tolerant Systems ......................... 9  
2.1   Introduction .................................................................................................... 9  
2.2   K-tolerant Systems ....................................................................................... 10  
2.3   Consistency in K-tolerant Systems ................................................................ 13  
    2.3.1  Voting ................................................................................................... 14  
    2.3.2  Available Copy Scheme ....................................................................... 16  
    2.3.3  Primary Copy Scheme ......................................................................... 17  
    2.3.4  Missing Writes ..................................................................................... 18  
    2.3.5  Regeneration ....................................................................................... 19  
    2.3.6  Comparison of the Two Approaches .................................................. 21  

Chapter 3  K-Tolerant Distributions and Data Replication: Definitions and Basic Properties ........................................................................................................... 25  
3.1   K-tolerant Distributions, Basic Properties and Definitions. ............................ 25  
3.2   K-Tolerant Distributions and Data Replication ............................................ 28
# TABLE OF CONTENTS

3.2.1 Neighbouring Data Replication ........................................ 29
3.2.2 Scattered Data Replication ............................................ 32

Chapter 4 Continuously K-tolerant Systems ................................. 37

4.1 Definitions and Basic Structures ....................................... 37
4.2 Basic Continuously K-tolerant System ................................ 44
  4.2.1 Neighbouring Data (ND) Replication Strategy ................. 44
  4.2.2 Failure Strategy .................................................. 44
  4.2.3 Recovery Strategy ............................................... 51
  4.2.4 Summary and Concluding Remarks ............................... 61

4.3 Balanced Case: Continuously K-tolerant, Balanced Load Requirement ........................................ 63
  4.3.1 Balanced Neighbouring Data (BND) Replication Strategy .......... 64
  4.3.2 Failure Strategy .................................................. 64
  4.3.3 Recovery Strategy ............................................... 74
  4.3.4 Summary and Concluding Remarks ............................... 74

4.4 Unbalanced Distribution, with Ordered Data Set F .................... 77
  4.4.1 Unbalanced, Ordered, Neighbouring Data (UOND) Replication Strategy ........................................ 78
  4.4.2 Failure Strategy .................................................. 78
  4.4.3 Recovery Strategy ............................................... 87
  4.4.4 Summary and Concluding Remarks ............................... 87

4.5 Balanced Distribution, with Ordered Data Set F ..................... 89
  4.5.1 Balanced, Ordered, Neighbouring Data (BOND) Replication Strategy ........................................ 90
PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1070a ANSI/ISO #2 EQUIVALENT

1.0  1.1  1.25  1.4  1.6
  2.0  2.2  2.5
  3.0  3.2

PRECISIONSM RESOLUTION TARGETS
List of Figures

Figure 3.1  Failure of a Node $S_i$ in a Data Configuration  26
Figure 3.2  Neighbour Data Replication Strategy  30
Figure 3.3  Scattered Data Replication Strategy  33
Figure 4.1  State Transition  38
Figure 4.2  Failure Mechanism  40
Figure 4.3  Recovery Mechanism  40
List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>p, q, u, v, a</td>
<td>Cardinals</td>
</tr>
<tr>
<td>i, j, k, r, z</td>
<td>Indices/Working indices</td>
</tr>
<tr>
<td>l, h</td>
<td>Range variables</td>
</tr>
<tr>
<td>F</td>
<td>A set of data (file)</td>
</tr>
<tr>
<td>$F_h$</td>
<td>A block of data with block index h</td>
</tr>
<tr>
<td>S</td>
<td>The set of nodes in the network</td>
</tr>
<tr>
<td>$S_i$</td>
<td>A node with working index i</td>
</tr>
<tr>
<td>n</td>
<td>The size of the network</td>
</tr>
<tr>
<td>X</td>
<td>Data distribution of F</td>
</tr>
<tr>
<td>$X_i$</td>
<td>The set of blocks stored at node $S_i$ (in a distribution)</td>
</tr>
<tr>
<td>Y</td>
<td>Data configuration of F</td>
</tr>
<tr>
<td>$Y_i$</td>
<td>The set of blocks stored at node $S_i$ (in a configuration)</td>
</tr>
<tr>
<td>$\alpha(F, Y)$</td>
<td>The degree of availability of F under Y</td>
</tr>
<tr>
<td>$\delta(X), K$</td>
<td>The degree of redundancy of a distribution</td>
</tr>
<tr>
<td>$\mu(x, X)$</td>
<td>The multiplicity of an object $x \in F$ in X</td>
</tr>
<tr>
<td>ML</td>
<td>Node membership list</td>
</tr>
<tr>
<td>FI</td>
<td>Array of failed indices</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>FWI</td>
<td>Set of Failed Working Indices</td>
</tr>
<tr>
<td>STM</td>
<td>Send to manager</td>
</tr>
<tr>
<td>STN</td>
<td>Send to node</td>
</tr>
<tr>
<td>RFM</td>
<td>Receive from manager</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 The Framework

The sharing of data in distributed systems is already common and will become pervasive as these systems grow in scale and importance. The need to share resources in a computer arises due to economics or the nature of some applications. In such cases, it is necessary to facilitate sharing long-term storage devices and their files. The challenge is to provide this functionality in a secure, reliable, efficient and usable manner that is independent of the size and the complexity of the distributed system.

A distributed file system allows users of physically distributed computers to share data and storage resources by using a common file system; examples of distributed file systems include the Andrew File System [HKM], Coda File System [SKK], Eden System [PNP] and the Sprite File System [HEL]. The term distributed system, as it used in this thesis, refers to a collection of nodes (workstations) interconnected by a communication network; unless otherwise specified, the network is a local area network (LAN). Every node in the system views the rest of the nodes and their respective resources as remote and its own resources as local [LeS].
A file is a sequence of blocks (pages, segments) of data. We consider a distributed file system in which each of the blocks constituting the file is stored at one or more of the nodes and each node contains at least a block of data.

A distributed system should continue to function, perhaps in a degraded form, after the occurrence of one or more faults in the system. Communication faults, machine failures (of type fail-stop), and storage device crashes are faults that should be tolerated to some extent. The degradation of the system can be in performance, functionality or both, but it should be proportional to the failures causing it.

An important property associated with distributed file systems is availability. A file is said to be available if it can be accessed whenever needed, despite machine and storage device crashes and communication faults. The goal is to have all files available at all times. On the other hand, node crashes cause loss of data. If a node S crashes, the data stored at S will be lost or become inaccessible. Thus the notion of availability is intrinsically related to the one of fault-tolerance.

To improve the fault tolerance of the system and thus the availability of the files, the blocks are often replicated at several nodes. A distributed file system is K-tolerant if each file is available after the simultaneous failure of up to K nodes. Replication is the method used to achieve K-tolerance.

The problem of making a file system K-tolerant has been addressed in [Bak] and [Rab]. Also in [Bak], K-Tolerant Distributions were defined and ways were considered to minimize the amount of replicated data required to achieve availability. In this thesis we will use some of these results in defining our solutions.
Chapter 1

Introduction

Given a K-tolerant system, most of the research has focused on the problem of maintaining the file consistency in such a system. In fact, independent updates at different copies of the file in conjunction with node failures and recoveries can compromise the integrity of the file. Several mechanisms have been proposed to solve this problem. Noticeable examples are: Voting and its variations ([DB], [El], [EIF], [Gar], [GB], [Gif], [RT], [Tho]), the Available Copy scheme ([BeG], [LP1]), the Primary Copy scheme [AD], the Missing Writes scheme ([COK], [EaS]), the Regeneration technique and its variations ([PNP], [AdT], [HL], LoCS]).

In this thesis, we are not concerned with the issue of consistency, an issue that has been dealt with extensively in the literature, but rather with a severe problem arising in K-redundant systems which will be addressed in the next section.

1.2 The Problem and the Results

1.2.1 The Problem

The major problem with a K-redundant system is that, after the occurrence of up to K simultaneous failures, the files are fully available but the system is no longer K-tolerant; that is, subsequent failures may compromise the availability of the files.

In this thesis, we are concerned with the design of systems which continuously guarantee data availability. Of the existing mechanisms for consistency, only the regeneration technique addresses this problem; however, it does not solve it.
A continuously K-tolerant system is a system which, starting from a K-tolerant configuration, after the simultaneous failure of up to K nodes, reconfigures itself so that it remains K-tolerant.

The goal of this thesis is to design Mechanisms that keep the system continuously K-tolerant. Each mechanism must consist of a Replication Strategy, a Failure Protocol and a Recovery Protocol:

• The Replication Strategy is used to achieve the initial K-tolerance in the system.

• The Failure Protocol is executed locally at every live node upon the detection of node failures in the network. The protocol reconfigures the data so that the new instance of the system is K-tolerant.

• The Recovery Protocol is executed locally at every live node when failed nodes recover. The protocol reconfigures the data so that new instance of the system is K-tolerant and every node contains at least one block. The Recovery Protocol is needed because the data of a failed node could be obsolete (and, thus, cannot be trusted) or destroyed; for these reasons when it recovers, a node is considered "empty".

In some cases, these Mechanisms must satisfy additional requirements which can be imposed on the system, such as local load balancing and data ordering.

Load balancing refers to the fact that the data is stored so to satisfy some balancing criteria imposed on the system. In some applications, where storage is expensive and the computation time is a factor of the amount of data stored at a given node, load balancing may be a required/attractive feature.

Data ordering refers to the fact that some ordering is applied to the data. Ordered data speeds up data searches and retrieval; thus, this property may be required on some systems that perform mostly searches and retrieval of data.
1.2.2 Assumptions

In this thesis a distributed system refers to a collection of nodes interconnected by a LAN. We view faults as node crashes, and we assume that node failures are detectable. A file is viewed as a sequence of blocks (pages) of data. Blocks constituting a file are stored at some nodes. Blocks stored at a node are a consecutive subset of the file. Also in order maintain full distribution of the data, we assume that live nodes hold at least one block of data.

1.2.3 The Results

The main contribution of this thesis is the development of efficient solutions to the problem of continuously guaranteeing the availability of the files under four different system requirements. Specifically we study the following cases:

1) Basic case: no restrictions imposed on the system.
2) Balanced case: balanced load requirement.
3) Ordered case: ordered data requirement.
4) Balanced Ordered case: balanced load and ordered data requirements.

For each of these cases, we develop a mechanism that keeps a K-tolerant system continuously K-tolerant after the occurrence of up to K faults in the system and the recovery of an arbitrary number of nodes. Each mechanism consists of a Replication Strategy, a Failure Protocol, and a Recovery Protocol.
Given a distributed file system, we study the minimum degree of redundancy required to make the system K-tolerant and present optimal Replication Strategies to achieve K-tolerance with the minimum amount of redundancy. These replication strategies provide an alternative intermediate solution between the fully redundant (where all the data is replicated at every node) and fully partitioned (with no replicated data) models.

The Failure Protocols we present depend on the properties and the restrictions imposed on the system. Thus, for each of the four cases, we will develop a Failure Protocol that preserves the properties of the initial system and keeps the system continuously K-tolerant.

With careful definition and selection of the data structures associated with the blocks of data stored at the nodes, we develop a general Recovery Protocol independent of the properties imposed on the system; thus, this protocol can be used for all the four cases. Furthermore it can be applied to any other mechanism that uses our proposed data structures.

The Recovery Protocol performs a perfect undo of the last execution of the Failure Protocol used by the system.

We study the complexity of the proposed solutions and prove their correctness. All the protocols we develop are fully distributed and highly efficient.

In this thesis we present a theoretical study of the problem described above; however, the solutions we present can be easily incorporated to existing methods resulting in an increased availability of the data.
We discuss how our protocols can be used in the Eden system as an effective alternative to the Regeneration technique.

1.3 Thesis Organization

This thesis is organized as follows.

In Chapter 2, we provide an overview of the current research on K-tolerant systems.

In Chapter 3, we introduce the terminology and definitions for K-tolerant distributions, and discuss their basic properties. We also give two examples of Data Replication Protocols which, once applied to a distributed system, will transform it to a K-tolerant system with minimum redundancy.

In Chapter 4, we introduce the notion of a continuously K-tolerant system and study the problems of making a system continuously K-tolerant. We define four Mechanisms that keep a K-tolerant system continuously K-tolerant after the occurrence of up to K simultaneous faults. Each Mechanism consists of a Replication Protocol, Failure Protocol and Recovery Protocol. The difference between these Mechanisms depends on the requirements (e.g., data balancing, data ordering) imposed on the system. For each Failure and Recovery protocol, we study its complexity and prove its correctness.
In Chapter 5, we look at an existing distributed system, and discuss how to improve the availability of its data by incorporating one of the Failure Protocols and the Recovery Protocol developed in this thesis. We also summarize the results of this thesis, give some concluding remarks, and discuss the areas of future work.
Chapter 2

Current Approaches to Managing K-tolerant Systems

2.1 Introduction

Data replication allows information to be located close to its point of use, either by statically locating copies in high use areas, or by dynamically creating temporary copies as dictated by demand. Replication of data also increases the availability of data and thus the fault tolerance of the system, by allowing many nodes to service requests for the same information in parallel and by masking partial system failures. In some cases, the cost of maintaining copies is offset by the performance, communication cost, and reliability benefits that replicated data affords.

A distributed file system is *K-tolerant* if each file is available after the simultaneous failure of up to K nodes.

In this chapter we will present some of the most noticeable work and results that are available in the area of K-tolerant systems. First we will look at approaches that have been proposed to make distributed systems K-tolerant, then we will look on how
the issue of data consistency and data availability is dealt with in K-tolerant system and compare the different methods that are available.

2.2 K-tolerant Systems

In designing fault-tolerant distributed file systems, a frequent goal is to make the system highly available despite component failures. To make a distributed file system K-Tolerant, it suffices to keep K+1 copies of the data at different nodes. When q ≤ K failures occurs, the data is still available (in the worst case, at K+1-q locations).

The problem of making a distributed file system K-tolerant has been addressed in [Rab] and [Bak].

In [Rab], the Information Dispersal Algorithm (IDA) was presented. Originally designed for fault-tolerant transmission of data, it provides a mechanism for constructing K-tolerant systems.

Given a distributed network of size n, the Information Dispersal Algorithm breaks a file F of length L = |F| into n blocks F₁, ..., Fₙ, each of size |Fᵢ| = L/m, 1 ≤ i ≤ n; m is chosen in such a way that the file can be reconstructed from any m blocks but can not be reconstructed in its entirety from any set of m-1 blocks.

The file is encoded and then dispersed over the n nodes. Thus, each block Fᵢ stored at node Sᵢ contains information of the file as a whole, but contains no individual data items.
Since m can be chosen such that n ≥ m and n/m ∼ 1, the algorithm is highly efficient in terms of storage requirement. Furthermore, the algorithm covers two distinct aspects of security: it guarantees confidentiality and can recover from loss of information. Also, the standard replication is a special case of this technique when m = 1.

The dispersal and reconstruction of data are computationally efficient. If n = m+k, then the IDA algorithm can tolerate up to k failures and "theoretically" provide full data availability since only m nodes are required to reconstruct the data.

The IDA has numerous potential applications to secure and fault tolerant storage and transmission of information. A beneficial side effect of using IDA is improved load balancing in storage and transmission. All of these claimed benefits of IDA are quantitatively analyzed and demonstrated in the case of packet switching on the n-cube [Rab]. Other applications of this technique are described in [Lyu].

It is interesting to note that the IDA considers only the case when a write operation updates n blocks and a read operation requires m blocks. This protocol is fault-tolerant for read operations but is not fault-tolerant for write operations. That is, all n blocks must be available for a write operation to complete successfully. A solution to this problem was presented in [AgE], where a quorum based data management protocol was combined with the dispersal algorithm to form a generalized algorithm to attain security and reliability in a distributed environment.

The major problem with Rabin's algorithm is that all the data is encoded before being stored; thus, the real data is not stored anywhere. Each of the n blocks contains
information of the file as a whole, but contains no individual data items. Thus, the blocks cannot be used for retrieval operations on the file. As a result, data access is very expensive since, for example, for a simple read operation, m nodes need to be accessed and the data has to be reconstructed before being made available.

These problems have motivated the research of [Bak] which considers a communication network in which nodes hold data in files but are subject to possible file-loss due to crashes. Assume that a node \( v \) has \( d \) neighbours and that \( t \leq d \) is a reasonable estimate on the number of neighbours that almost certainly remain active whenever \( v \) crashes. Suppose node \( v \) keeps a collection \( B \) of blocks. If \( v \) crashes, it is possible that the blocks in \( B \) will be lost. Thus, it is important that some \( t \) or more neighbouring nodes that are still active are able to supply the data necessary to reconstruct \( B \).

In this case, the solution of [Rab] cannot be employed since, as mentioned before, it uses blocks which contain information of the file as a whole, but no individual data items. Of interest is a solution to the problem such that the information held by the neighbours can be used for some retrieval operations on \( F \). A simple solution would be to store copies of \( F \) at \( (d - t + 1) \) neighbours of \( v \). A disadvantage of this solution is that it does not spread the burden of keeping the file \( F \) equally among the neighbours of \( v \), i.e., there is no load-sharing. An alternative solution that addresses this problem is also studied.

Furthermore, [Bak] defined and discussed \( K \)-tolerant distributions, their properties and how to build a \( K \)-tolerant distribution with minimum amount of redundancy. The paper also discussed two relevant problems: segmentation and covering. In this thesis
we will adopt the definition of K-tolerant distributions introduced in [Bak] and build on his results.

2.3 Consistency in K-tolerant Systems

Most of the research on K-tolerant systems has focused on the problem of maintaining the file consistency in such systems. Two distinctive approaches are easily detected in the literature.

The first approach consists in insuring the consistency of the replicated data by honouring read and write requests when appropriate quorums hold. Examples of schemes based on this approach include: Voting ([DB], [El], [Elf], [Gar], [GB], [Gif], [RT], [Tho]), the Available Copy scheme ([BeG], [LP1]), the Primary Copy scheme [AD], and the Missing Writes scheme ([COK] [EaS]). None of these schemes deal with the issue of file availability.

The second approach uses data "regeneration" to assure consistency and availability. Schemes that use this approach include ([LoCS], [PNP], [AdT], [HL]).

The first approach will be discussed in Sections 2.3.1-2.3.4, and the second approach in Section 2.3.5. A comparison of the two approaches is found in Section 2.3.6.
2.3.1 Voting

Voting is the most popular technique for managing data consistency in a distributed replicated file system; its popularity rests on its simplicity and robustness. Voting schemes do not depend on any sophisticated message passing and are unaffected by network partitions. They insure the consistency of replicated files by honouring read and write requests when appropriate quorums hold.

In order to access a replicated resource, voting algorithms define a read quorum $N_r$ and write quorum $N_w$ such that $N_r + N_w > N$ (the total number of copies). In a read operation the availability of $N_r$ copies is required; in a write operation, the availability of $N_w$ copies is required. The number of inaccessible copies tolerated in a read operation is $N - N_r$, and in a write operation is $N - N_w$. In case of network partitions, voting allows access only from the majority partition if one exists.

Voting assumes that the correct state of a replicated file is the state of the majority of its copies. Ascertaining the state of a replicated file requires collecting a quorum of the copies. Should this be prevented by one or more node failures, the file is considered to be unavailable. Consistency is guaranteed as long as:
- The write quorum is high enough to disallow simultaneous writes on 2 disjoint subsets of the copies.
- The read quorum is high enough to disallow simultaneous reads and writes on two disjoint subsets of the copies.

These conditions are simple to verify, and account for much of the conceptual simplicity and the robustness of voting schemes.
A problem with voting is that it disallows all file accesses when a majority of the copies are not accessible; thus, it may declare "unavailable" a file which is fully available but at a small number of nodes. This fact considerably reduces file availability in the presence of node failures.

Several variations of the voting algorithm have been developed to overcome these limitations.

*Dynamic Voting* [DB] is another algorithm capable of functioning in the presence of network partitions and node failures. Dynamic Voting adjusts the necessary quorums of copies required for update or read operations (i.e., the values of $N_r$ and $N_w$) according to the changing state of the network. This scheme improves file availability at the cost of increased network traffic and computation.

A more general voting scheme is *Weighted Voting* [Gif]. In this algorithm for maintaining replicated data, every copy of a replicated file is assigned a number of votes. Every transaction collects a read quorum of "$r" votes in order to read a file, and a write quorum of "$w" votes in order to write to a file, where $r + w$ is greater than the total number of votes assigned to the file. This property ensures that there is a non-null intersection between every read quorum and write quorum. The reliability and performance characteristics of a replicated file can be controlled by appropriately choosing $r$, $w$, and the file's voting configuration. The algorithm guarantees serial consistency and admits temporary copies by introducing copies with no votes.

The Weighted Voting algorithm has the following desirable properties:

- It continues to operate correctly with inaccessible copies.
• It requires a small amount of overhead that runs on top of a transactional file system.
• It provides serial consistency.
• By manipulating r,w, and the voting structure of a replicated file, the file's performance and reliability characteristics can be altered.

Several other variations of the voting technique have been studied and developed; for example Voting with Witness [Par1], Voting with a Variable Number of Copies [Par2], and Voting with Ghosts [RT].

2.3.2 Available Copy Scheme

When network partitions are known to be impossible, the Available Copy schemes provide a simple means for maintaining file consistency ([GSC], [BeG], [LP1]). Available Copy schemes are based on the observations that, as long as one node has been continuously available, it is known to hold the most recent version of the data. The update and recovery rules are defined as follows:
• The update rule is to write to all available copies. Since all available copies receive each update, they are all kept in a consistent state. Therefore, the data can be read from any available copy.
• When a node recovers following a failure, the recovering node takes a copy of the file from a node holding the most recent copy.
A complication occurs when all nodes fail. In fact, in this case, it is not known to the recovering nodes which of them hold the most recent copy of the file; this problem can be resolved by determining the last node to fail.

In the original proposal for the available copy scheme, detecting the last node to fail was achieved by maintaining several sets of failure information, such as: all nodes participating in the replication file, those nodes which have been specifically included, and those which have been specifically excluded. An included node is one which is known to hold a copy of the most recent version of the data, an excluded node is one which has failed and that failure has been detected by an operational node. This method of detecting the last node to fail is unreliable, and heavy in communication. For these reasons, the assumption is made that failures are easily detected and that notification of their occurrence can be reliably broadcasted to all surviving nodes. An improved method was presented in [LP1], which requires only that the availability information be brought up to date when the file is modified or when repair operation occurs.

2.3.3 Primary Copy Scheme

The Primary Copy Scheme for data replication designates one copy as the primary and the others as back-ups [AD]. All writes are processed first by the node holding the primary copy of data, which then propagates the updates to the nodes holding the secondary copies. Reads can be serviced by any node. Inaccessible nodes with secondary copies are removed from the chain of communications and reinitialized when they join the network.
With a fixed primary node, a resource replicated with the primary copy method only increases read availability through secondary copies. Its write availability is the same as the availability of the primary node.

The main difficulty with the primary copy method is determining what to do when the primary copy is inaccessible. If the node holding the primary copy is down, a reassignment is in order. However, if the network has partitioned, a reassignment would compromise consistency.

2.3.4 Missing Writes

First described in [EaS] and later analyzed in [COK], this scheme attempts to achieve high availability by switching between different quorum sizes; an extra mechanism is required to accomplish this switch without loss of one copy-serializability. Transactions run in either of two modes: the normal mode or the partitioned mode.

In the normal (no failures) mode, transactions read any copy and write to all copies. The algorithm keeps track of which updates were not made to nodes due to failure; this constitutes the Missing-Write information. Whenever a normal mode transaction has read a data item that has missing write information associated with it, it must abort. To proceed, it restarts in partitioned mode using a voting mechanism.

Since normal mode and partitioned mode transactions can coexist, a normal mode transaction may be serialized before and after some partitioned mode transaction,
leading to a non-serializable state. To prevent this from happening, partitioned mode
transactions are always serialized after normal mode transactions.

This algorithm achieves long term availability at the expense of computational
speed.

2.3.5 Regeneration

A different approach to maintaining replicated data has been adopted by the
proponents of a technique called Regeneration. The reliability and availability of
replicated data can often be increased by generating new replicas when some become
inaccessible due to failures. This technique has been used in the *Regeneration
Algorithm*, a replica control protocol based on file regeneration.

The notion of regenerating replicas to replace those lost due to node failures was
first proposed by as a technique for increasing the availability of replicated data
objects in the Eden System ([Pu] and [PNP]). The protocol, called the Regeneration
Algorithm, provides mutual and serial consistency of replicated data objects in a
partition free distributed system. When it detects that one or more of the replicas have
become inaccessible due to system failure, it creates new replicas of the data object to
replace those lost.

A replica control protocol based on regeneration begins with a set of replicas
placed on nodes around the computer network. The protocol requires only one copy
for reading, but all copies for writing. When an update occurs, if fewer than the
required copies are available, additional copies are regenerated on other operating nodes. In doing so, the system must check to be sure that space for the new copy is available and no other copy of the particular resource already exists on the node. Using this method, one is assured of regenerating up to the required maximum number of copies whenever an update occurs, as long as adequate nodes exist on which to make copies. A write failure occurs if the maximum number of copies cannot be provided. A read failure occurs when no copy can be found.

This method also requires some work to be done when a node recovers and rejoins the set. For each replicated resource in the returning node, the system must check to see if the maximum number of copies exists. If so, the returning copy is deleted. If not, the system must check another copy to see if the resource was updated during the absence of the node. If not, then the copy may be used; otherwise it should be deleted: its replacement exists, but has temporarily vanished because of the crash of some other node.

This approach differs from previous work in several ways:
1. Regeneration does not have to trade off read availability against write availability. This is like the Available Copies method, but unlike Voting.
2. Resources replicated using Regeneration are not only consistent and highly available, but also self adaptive to system configuration changes.
3. In an unstable environment, Regeneration requires fewer copies for a given probability of survival, compared to the Primary Copy, Voting, and Available Copies methods.
4. The Regeneration method was built and tested. Most replication methods, with the exception of Voting, have not been used in implemented systems.
5. Consistency is maintained since competing read and write requests are serialized by transaction techniques.

6. It provides maximum availability for both read and write requests.

7. Resources and copies are not fixed to specific nodes, providing for system reconfigurability.

8. Transparency is offered since clients can access the directory without knowing about the replicated structure and components.

The Regeneration algorithm is simple and efficient, but some weaknesses can be identified.

It allows reads to continue as long as one current replica of the object remains accessible; however, writes are disabled if fewer than the initial number of replicas are accessible and there are insufficient spares to replace the missing replicas.

Recovering from a total system failure is unspecified, requiring manual intervention in the event of a total system failure. It is also unable to guarantee mutual consistency in the presence of network partitions. Regeneration when as few as one replica is accessible means that partitioned networks could regenerate independently, so one must depend on other methods to restore consistency after a network partitions (e.g. merging, or offer update availability only in the majority partition). Also it is essential that the regeneration protocol be atomic to ensure a consistent generation of replicas.
2.3.6 Comparison of the Two Approaches

In contrast to the Voting scheme described previously, the Regeneration method requires only one copy for reading, and all copies for writing, and tolerates all but one inaccessible copies for both read and write. For network partitions, Voting allows access only from the majority partition if one exists. Voting can be combined with Regeneration to increase resource availability. In gathering a write quorum, the quorum collector can copy the current version onto obsolete representatives.

The main difference between Missing Writes and Regeneration is the length of time they operate in the normal mode. Missing scheme falls back to voting whenever one copy is inaccessible. The Regeneration method replaces the out of date copy and keeps operating normally. However, if no additional nodes are accessible for placing new copies, Missing Writes still works in voting mode, while Regeneration cannot proceed unless it is modified to allow dynamic alteration of the required number of copies.

The Available Copies and Regeneration algorithms use the same basic principle, and configuration data can be changed to provide single copy read and write availability of replicated resources despite detectable crashes. The Available Copies algorithm requires only one copy for reading, and all accessible copies for writing.

There are three important differences between the Available Copies and Regeneration algorithms:

1. The first difference is the amount of work done at crash detection and recovery. The Available Copies algorithm is pessimistic, excluding all potentially out of date
copies, while Regeneration is optimistic, replacing only actually inconsistent copies when necessary.

2. The second difference resides in the scope of adaptation. The Available Copies algorithm can exclude out of data copies, but it does not replace them until their nodes recover.

3. The third difference is in space requirements. The minimum storage requirements for regeneration are slightly in excess of the amount required for the initial redundancy since there must be room somewhere in the system to make the extra copies produced by regeneration. So, in comparison to Available Copies, Regeneration trades some extra storage for greater availability.

Using regeneration to maintain a viable set of current replicas is a technique that can be adapted to many existing replica control protocols [Lo]. By combining regeneration with protocols with desirable characteristics, the weakness of the Regeneration Algorithm can be successfully addressed.

Regeneration can be used with the Available Copy protocols ([BeG], [BHG], [CLP], [LP1]) to improve fault tolerance in non partitionable computer networks. This combination would considerably improve on write availability by allowing writes as long as one replica of the data remains accessible. Recoveries from total failures would also be accelerated as the protocol would keep track of replica states.

When the communication network is susceptible to partitioning, consensus protocols are required. The Available Copy protocol with regeneration performs better than the Regeneration Algorithm and all Available Copy protocols not including regeneration [LP3].
By combining regeneration with static majority consensus, a simple protocol for maintaining mutual consistency among replicas of a data object is obtained [LP3].

Regeneration can be combined with Dynamic Voting protocols ([DB], [JM], [LP2]) to provide an increased level of fault tolerance over that of static consensus protocols [LP3].
Chapter 3

K-Tolerant Distributions and Data Replication: Definitions and Basic Properties

3.1 K-tolerant Distributions, Basic Properties and Definitions.

We consider a distributed system of n processing nodes $S = \{S_1, \ldots, S_n\}$ which communicate by transmitting messages over a local area network.

Given a finite set of data $F$, let $Y_i \subseteq F$ denote the elements of $F$ stored at node $S_i$.

The n-tuple $Y = < Y_1, \ldots, Y_n >$ is called the data configuration (or, simply, configuration) of $F$.

A data configuration $Y = < Y_1, \ldots, Y_n >$ is:

1. a distribution (or cover of $F$) if: $\bigcup_i Y_i = F$
2. a partition, if $Y$ is a distribution and: $Y_i \cap Y_j = \emptyset$ for $1 \leq i \neq j \leq n$.
3. a total replication, if: $Y_i = F$ for all $i$.

Definition 3.1. Given a configuration $Y$, the degree of availability $\alpha(F,Y)$ of $F$ under $Y$ is: $\alpha(F,Y) = |\bigcup_i Y_i|/|F|$. Note that $\alpha(F,Y) = 1$ (i.e., $F$ is fully available) if and only if $Y$ is a distribution.
In this chapter we are mainly interested in distributions.

**Definition 3.2.** Given a distribution $X$, the *degree of redundancy* $\delta(X)$ is the ratio between the amount of stored data in the system and $F$; that is $\delta(X) = \sum X_i / |F|$.

The nodes of a distributed system are subject to failures. A node failure makes the data stored at the failed node inaccessible for the duration of the failure.

We consider crash failures only and assume that failures are detectable (e.g., there is a reliable group membership service which provides information about the available/unavailable nodes). We also assume that the operation of transmitting data from one node to another is atomic, and that it successfully completes before the next set of failures. Since the communication network is a LAN, the system is not subject to partitions.

These same assumptions were made on the Regeneration Algorithm for replica replacement [PNP].

We describe the failure of node $S_i$, and indicate that the data stored at $S_i$ has become unavailable, by saying that the system has a new configuration $Y' = <Y'_1, ..., Y'_n>$, where $Y'_i = \emptyset$ and $Y'_j = Y_j$ if $j \neq i$ (see Figure 3.1).

![Figure 3.1 Failure of a Node $S_i$ in a Data Configuration](image-url)
Definition 3.3. K-tolerant Configuration: Given a configuration $Y$ of $F$, $Y$ is $K$-tolerant if $K$ is the largest integer such that the degree of availability of $F$, $\alpha(F,Y)$, is unchanged after the simultaneous occurrence of up to $K$ faults.

The setting considered here is strictly related to the one investigated in [Bak]. In the following we will discuss this relationship.

Definition 3.4. $(d,t)$-distribution [Bak]. Let $t$ and $d$ be positive integers such that $1 \leq t \leq d$. A $(d,t)$-distribution of $F$ is any collection $d$ of subsets $X_1, \ldots, X_d \subseteq F$ such that the union of every $t$ subsets is equal to $F$ but no union of $(t-1)$ subset is.

Property 3.1. A distribution $X$ of $F$ is $K$-tolerant iff it is an $(n, n-K)$-distribution of $F$.

Proof: (only if) If the distribution $X$ is $K$-tolerant, then it can tolerate the simultaneous failures of any $K$ nodes. That is, the union of every $n-K$ sets $X_i$ is equal to $F$; since $K$ is the largest integer for which this is true, no union of $n-K-1$ sets is equal to $F$. Therefore $X$ is a $(n, n-K)$ distribution of $F$.

(if) If $X$ is an $(n, n-K)$ distribution of $F$, then the union of any of $n-K$ subsets $X_i$ is equal to $F$, but no union of $(n-K-1)$ is. That is, the degree of availability after the failure of any $K$ subsets $X_i$ is still unchanged; on the other hand, the failure of $K+1$ subsets will reduce the availability of $F$. Therefore $X$ is a $K$-tolerant distribution.[1]
Property 3.2. There exist a K-tolerant distribution of F if and only if

\[ |F| \geq \binom{n}{n-k-1} \]

Proof: By Property 3.1 and the fact that a file has a (d,t)-distribution if and only if

\[ |F| \geq \binom{d}{t-1} \] [Bak.]

To achieve fault tolerance, data is often replicated on several nodes. Such replication allows the system to remain available despite of node failures. Replicated data has to be managed properly in order to ensure consistency and availability of the data. The remainder part of this chapter introduces the notion of replication and shows how replication is necessary to make a system K-tolerant.

3.2 K-Tolerant Distributions and Data Replication.

Replication in distributed systems has been primarily used as a mean of increasing the degree of availability in the system and reducing the cost associated with accessing remote data. The motivation behind our work is to develop Mechanisms that provides complete data availability in presence of K failures. In this section, we are concerned with constructing K-tolerant distributions using a minimum amount of data replication.

Definition 3.5. The multiplicity of an object \( x \in F \) in a distribution \( X, \mu(x,X) \), is the number of occurrences of \( x \) in \( X \); that is, \( \mu(x,X) = \sum_i |x \cap X_i| \).
We will first establish an upper bound on the degree of redundancy \( \delta(X) \) of a K-tolerant distribution.

**Theorem 3.1.** In a K-tolerant distribution \( X \), \( \delta(X) \geq K+1 \).

**Proof:** Let \( X \) be a distribution of \( F \) on \( S \). It suffice to prove that if \( X \) is K-tolerant then for all \( x \in F \), \( \mu(x,X) \geq K+1 \).

By contradiction, let \( X \) be K-tolerant and assume that there exists an object \( x \in F \) such that \( \mu(x,X) < K \). Without loss of generality, let \( X_1, \ldots, X_i \) (\( i < K \)) be the subsets containing \( x \). The simultaneous failure of the nodes \( S_j \) (\( 1 \leq j \leq i \)) makes \( x \) unavailable, contradicting the fact that \( X \) is K-tolerant.[]

A similar bound was established by [Bak] for (d,t)-distributions using a somewhat more complex proof.

The bound of Theorem 3.1 is achievable. In fact, there are many different ways of constructing K-tolerant distributions \( X \) with \( \delta(X) = K+1 \). Following are two strategies illustrating that the bound is achievable and, thus, it is tight.

For presentation purposes, for both strategies, we consider first the case \( K = 1 \), and then the generalization \( K \geq 1 \).
3.2.1 Neighbouring Data Replication

In this section, we present the Neighbouring Data (ND) Replication Strategy, a strategy that constructs a K-tolerant distribution with minimum redundancy. First we present the case where \( K = 1 \), then the generalization where \( K \geq 1 \).

\( K = 1 \).

Given a distributed system with \( n \) nodes \( S = \{S_1, \ldots, S_n\} \), and a data set \( F \) to be stored on the system, consider the partition of \( F \) into \( n \) subsets \( F_0, \ldots, F_{n-1} \), such that:

\[
|F_j| = [m/n] \quad \text{for} \quad 0 \leq j \leq [m]_n \quad \text{and},
\]

\[
|F_j| = [m/n] \quad \text{for the remaining indices},
\]

where \( m = |F| \) and \([x]_y\) denotes "\( x \) modulo \( y \)"

Define \( X = \langle X_1, \ldots, X_n \rangle \) as follows:

\( X_i = F_i \cup F_{i+1} \), where all indices of the \( F_i \)'s are modulo \( n \). (see Figure 3.2.)

<table>
<thead>
<tr>
<th>Site allocation:</th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>( S )</th>
<th>( S )</th>
<th>( S_5 )</th>
<th>\ldots</th>
<th>( S_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition F:</td>
<td>( F_1 )</td>
<td>( F_2 )</td>
<td>( F_3 )</td>
<td>( F_4 )</td>
<td>( F_5 )</td>
<td>\ldots</td>
<td>( F_0 )</td>
</tr>
<tr>
<td>Replication:</td>
<td>( F_0 )</td>
<td>( F_1 )</td>
<td>( F_2 )</td>
<td>( F_3 )</td>
<td>( F_4 )</td>
<td>\ldots</td>
<td>( F_{(n-1)} )</td>
</tr>
</tbody>
</table>

*Figure 3.2. Neighbour Data Replication Strategy*

**Properties:**

1. \( X = \langle X_1, \ldots, X_n \rangle \) is a distribution since \( \cup X_i = F \)

2. \( X \) is a 1-tolerant distribution. If at any given time a node \( S_i \) crashes then the data stored at \( S_i \) is inaccessible, i.e.: \( X_i = F_{i-1} \cup F_i \), is inaccessible.

However, \( X_{i+1} \) can be accessed from node \( S_{i-1} \) and \( X_i \) can be accessed from \( S_{i+1} \). Thus \( F \) is still fully available.
3. The total amount of replication is \(|F|\). Thus the degree of redundancy is 
\[ \delta(X) = K+1 = 2; \] which, by Theorem 3.1, is optimal.

Generalization: \( K \geq 1 \)

Let \( S, F \) and \( F_i \) be as described above.

Define \( X = \langle X_1, ..., X_n \rangle \), where
\[ X_i = F_i \cup F_{i-1} \cup ... \cup F_{i-(k+1)}, \] and all the operations on the indices are modulo \( n \).

**Theorem 3.2.** The distribution constructed using the Neighbouring Data Replication strategy described above is \( K \)-tolerant and minimal.

**Proof:** First observe that \( X = \langle X_1, ..., X_n \rangle \) is a distribution since \( \cup X_i = F \).

Since each \( F_i \) is stored at \( K+1 \) nodes, if any \( K \) nodes crash, each \( F_i \) is still stored in at least one node. Therefore \( X \) is a \( K \)-tolerant distribution.

The total amount of replication is \( K \times |F| \). Thus the degree of redundancy is 
\[ \delta(X) = K+1; \] which by Theorem 3.1 is optimal.[]
The construction above implies a simple protocol for transforming any partition $X = <X_1, ..., X_n>$ into a $K$-tolerant distribution $X' = <X'_1, ..., X'_n>$.

**ND Replication Strategy**

**Protocol** executed at node $S_i$:

Begin

send $X_i$ to $S_{(i+1)}$ ... $S_{(i+K)}$

$X'_i = X_i$

For $1 \leq j \leq K$

receive $Z_j$ from $S_{(i-j)}$ $1 \leq j \leq K$

$X'_i = X'_i \cup Z_j$

End

End

**3.2.2 Scattered Data Replication**

In this section, we present another replication strategy, the Scattered Data (SD) Replication, that constructs a $K$-tolerant distribution with minimum redundancy. First we present the case where $K=1$, then the generalization where $K \geq 1$.

$K = 1$

Given a distributed system with $n$ nodes $S = \{S_1 ... S_n\}$, and a data set $F$ to be stored on the system, let $F_i$ be as defined in the ND strategy. In addition, we partition $F_i$ into $n$ subsets $F_{ij}$ ($0 \leq j \leq n-1$) where $F_{ij} \neq \emptyset$ iff $i \neq j$. Let $F_{*j} = \bigcup F_{ij} (0 \leq i \leq n-1)$. 
Define $X = <X_1, ..., X_n>$ as follows:

$X_i = F_i \cup F_{*,i}$ (see Figure 3.3)

<table>
<thead>
<tr>
<th>Site allocation:</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>...</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition $F$:</td>
<td>$F_1$</td>
<td>$F_2$</td>
<td>$F_3$</td>
<td>$F_4$</td>
<td>$F_5$</td>
<td>...</td>
<td>$F_0$</td>
</tr>
<tr>
<td>Replication:</td>
<td>$F_{1,2}$</td>
<td>$F_{1,3}$</td>
<td>$F_{1,4}$</td>
<td>$F_{1,5}$</td>
<td>...</td>
<td>$F_{1,0}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$F_{2,1}$</td>
<td>$F_{2,3}$</td>
<td>$F_{2,4}$</td>
<td>$F_{2,5}$</td>
<td>...</td>
<td>$F_{2,0}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$F_{0,1}$</td>
<td>$F_{0,2}$</td>
<td>$F_{0,3}$</td>
<td>$F_{0,4}$</td>
<td>$F_{0,5}$</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 3.3. Scattered Data Replication Strategy*

**Properties:**

1. $X = <X_1, ..., X_n>$ is a distribution since $\cup X_i = F$

2. $X$ is a 1-tolerant distribution. If at any given time a node $S_i$ crashes then the data stored at $S_i$ is inaccessible, i.e.: $X_i = F_i \cup F_{*,i}$ is inaccessible. However, each sub-block $F_{ij}$ of $F_i$ can be accessed from node $S_j$ (0 ≤ $j$ ≤ n-1 and $j \neq i$). Furthermore, each $F_{pi}$ can be accessed from $S_i$ (0 ≤ $p$ ≤ n-1 and $p \neq i$). Thus, $X_i$ is still fully accessible and $F$ is fully available.

3. The total amount of replication is $|F|$. Thus the degree of redundancy is $\delta(X) = K+1 = 2$; which, by Theorem 3.1 is optimal.

**Generalization: $K > 1$**

Let $S$, $F$ and $F_i$ be as described above.

Define $X = <X_1, ..., X_n>$, where

$X_i = F_i \cup F_{*,i} \cup F_{*,i+1} \cup ... \cup F_{*,i+K}$
Theorem 3.3. The distribution constructed using the Scattered Data Replication Strategy described above is K-tolerant and minimal.

Proof: First observe that \( X = <X_1, ..., X_n> \) is a distribution since \( \cup X_i = F \).

If at any given time, \( K \) nodes crash then the data stored at the \( i \) nodes that failed becomes inaccessible. \( F_i = \cup F_{i,j}, 0 \leq j \leq n-1 \) and \( j \neq i \). Each \( F_{i,j} \) is replicated at \( K+1 \) nodes. This is equivalent to saying that each \( F_i \) is replicated in \( K+1 \) nodes. If at most \( K \) nodes fail, \( F_i \) is still accessible from at least one node. In other words every \( F_i \) is accessible after \( K \) simultaneous failures; thus \( F \) is still fully available. Therefore \( X \) is a \( K \)-tolerant distribution.

The total amount of replication is \( K \times |F_i| \). Thus the degree of redundancy is \( \delta(X) = K+1 \); which by Theorem 3.1 is optimal.[]
The construction above implies a simple protocol for transforming any partition $X = \langle X_1, ..., X_n \rangle$ into a K-tolerant distribution $X' = \langle X'_1, ..., X'_n \rangle$.

**SD Replication Strategy**

**Protocol executed at node $S_i$:**

Begin

For $1 \leq j \leq n$, and $j \neq i$

If $0 \leq i - j \leq K$ then

send $X_{ij}$ to $S_i$:

\[ j \leq 1 \leq j + K + 1 \text{ and } l \neq i \]

Else

send $X_{ij}$ to $S_i$:

\[ j \leq 1 \leq j + K \]

\[ X'_i = X_i \]

For $1 \leq j \leq n$, and $j \neq i$

For $1 \leq l \leq K$

receive $Z_{j,l}$ from $S_j$

\[ X'_i = X'_i \cup Z_{j,l} \]

End

End

**Theorem 3.4.** Both protocols transform a partition into a K-tolerant distribution with the minimum amount of communication.
Proof: The total amount of communication is $K |F|$. By Theorem 3.1, in a $K$-tolerant distribution, each data item must be stored at $K+1$ nodes. Since in a partition, each data item is stored at only one node, the theorem follows."
Chapter 4

Continuously K-tolerant Systems

4.1 Definitions and Basic Structures

In the previous chapter we defined and discussed K-tolerant configurations. In this chapter we are going to define and discuss continuously K-tolerant systems.

Definition 4.1. A system is said to be in a *K-tolerant state*, if the current distribution is K-tolerant and any live node contains a non empty subset of the data.

Obviously, if the number of live nodes is less than K+1, the system cannot be in a K-tolerant state; in the following we will consider that the number n of live nodes is at least K+1.

If p ≤ K failures occur in the system, the data is still fully available; however, the system may enter in a (K - p)-tolerant state. What we are interested in is to provide mechanisms such that, upon the occurrence of the failures, the system automatically moves to a K-tolerant state.
Depending on the actual system, failure of a node might imply permanent loss of part or all of the data stored at that node; or, perhaps, corruption of part or all of the data. Also in case data is not lost nor corrupted, the data stored in a failed node $S_i$, reflects the state of the system preceding its failure, which could be different from the current state if one or more nodes have failed or recovered while the node $S_i$ was down. For all these reasons, we consider the data of a failed node as lost (the same assumption was made by the Regeneration Algorithm [PNP]). Thus, upon recovery, a node does not have any data and waits for further processing (as defined by the failure/recovery mechanisms). A recovering node will be declared live only upon completion of the recovery mechanism; obviously, it can fail while recovering (see Fig. 4.1)

![State Transition Diagram](image)

*Figure 4.1 State Transition*

Note that, if a failed node $S_i$ recovers, the resulting distribution is $K$-tolerant; however the system is not in a $K$-tolerant state since the recovered node does not contain any data. We would like the system to automatically move to a $K$-tolerant state also in this case.
In other words, we are interested in mechanisms which restore the system to a K-tolerant state following \( p \leq K \) failures or an arbitrary number of recoveries. We shall call any such mechanism \( M \).

**Definition 4.2:** A system is continuously K-tolerant if, starting with a distribution in a K-tolerant state it enters a K-tolerant state after the simultaneous failure of \( q \leq K \) nodes and/or the recovery of an arbitrary number of failed nodes.

Such a system obviously requires both processing and communication activities by the live nodes. In particular, data must be replicated at additional nodes, possibly removed from some nodes, so that the resulting distribution is K-tolerant. Thus, any mechanism \( M \) is a protocol defining which action must be performed by each node upon detection of a change in the set of live nodes.

The specification of a continuously K-tolerant system requires the specification of two sets of operations: Failure and Recovery (see Figures 4.2 and 4.3). Each set of operations is to be performed locally by all nodes and might require communication activities and transmission of data among nodes; upon completion of each operation, the system is again in a K-tolerant state.

The failure operations are triggered by the failure of \( p \leq K \) nodes; the recovery operations are triggered by the recovery of any failed nodes. Assuming that the distribution before the event (failure or recovery) is K-tolerant, following the execution of the appropriate set of operations the resulting distribution must still be K-tolerant. Thus, these two operations are sufficient to keep a K-tolerant system continuously K-tolerant.
Obviously, the type of replication strategy employed has an impact on the definition of Failure and Recovery. In other words, the mechanism $M$ contains the specification of the Replication Strategy, Failure Protocol and Recovery Protocol. In the previous chapter, we defined and discussed two such replication strategies, the Neighbour Data (ND) and the Scattered Data (SD) strategies. The ND Replication Strategy is the strategy that will be used in this chapter when specifying instances of the mechanism $M$.

In the rest of this chapter, we consider and analyze several instances of $M$, by specifying their corresponding Failure and Recovery Protocols. We will also consider two additional requirements which can be imposed on the system, local load balancing and data ordering, discussed below:

- **Load balancing**: refers to the fact that the data satisfies a balancing criteria imposed by the system. In some applications, where storage is
expensive and the computation time is a factor of the amount of data stored at a
given node, load balancing might be a required/attractive property.

◇ Data ordering: refers to the fact that some ordering is applied to the data.
Data ordering speeds up data searching and retrieval. Thus this property may
be required on some systems that perform mostly searches and retrieval of
data.

In particular, we will consider the following cases:

1. Basic case: Continuously K-tolerant and no additional requirements.
4. Balanced and ordered case: Continuously K-tolerant, balanced load and
   ordered data requirements.

For each case, we will present an instance of $M$, analyze its complexity and prove
its correctness.

We will now introduce the terminology and the basic structures used by the
proposed solutions.

The failure and recovery operations use the notion of working indices.

Definition 4.3. Working indices: A working index is the index assigned to a given
node. Initially, the working index of a node is its own index. When a node fails, its
working index becomes available and is stored in the (chronologically ordered) set
FWI of failed working indices. When a node recovers the most recent working index
stored in FWI is assigned to that node.
In the following, we will denote by $S_i$ the node whose working index is $i$. There exists a natural cyclic ordering of the nodes using the associated working indices; i.e., $S_i$ is followed by $S_{i+1}$.

**Definition 4.4. Block of data:** A block of data $F_h$ is an atomic unit of data; i.e., $F_h$ cannot be subdivided into smaller components.

The size of a block of data varies with the application and can be user definable. It can be a single record, a page of data or multiple pages of data; once specified, it cannot be altered.

**Basic Structures:**

1. All live nodes in the network keep a list of the working indices of the failed nodes; recall that, by definition, all failures are detectable. In case of simultaneous failure of $p$ nodes, the corresponding working indices are added to the list in increasing order.
2. At any given time, each live node in the network has access to a local variable CurrentFailure which indicates the number of nodes that have failed to date.
3. With every block of data $F_h$ stored at a node is associated:
   
   a) a *node membership list* $ML = \{\text{set of the working indices of the live nodes in the network that hold a copy of the data block } F_h\}$.
   
   b) an *array of failed indices* $FI$, of size $n$ such that:

   $FI[i] = 0$ \hspace{1cm} For initial data partition and replication

   $FI[i] = p$ \hspace{1cm} If the block of data $F_h$ was replicated at node $S_i$ when $\text{CurrentFailure} = p$

   $FI[i] = \text{NULL}$ \hspace{0.5cm} otherwise
When one or more nodes fail, blocks will be sent to live nodes to restore a K-tolerance state; when a node recovers, the recovery mechanism acts as an "UNDO" on the last failure. Thus, it is necessary to establish a total "temporal" order also among simultaneous failures or recoveries; such an ordering is purely a logical one and is used solely for computation purposes. In particular, the simultaneous failure of several nodes is viewed as if the nodes failed according to the order of their working index values.

Conversely, the simultaneous recovery of several nodes is viewed as if the nodes recovered in the reverse ordering of their working index values.
4.2 Basic Continuously K-tolerant System

Consider a network $S$ with $n$ nodes, a set of data $F$ and a partition $< F_1, F_2, ..., F_n >$ of $F$ over $S$, where $F_i$ denotes the block stored at $S_i$. We would like to maintain a continuously K-tolerant system.

4.2.1 Neighbouring Data (ND) Replication Strategy

Initial K-tolerance is achieved by introducing the ND replication strategy as follows:

**ND Replication Strategy:** At every node $S_i$ we replicate the block of the previous K nodes. Thus the data stored at node $S_i$ is: $X_i = F_h \cup F_{(h-1)} \ldots \cup F_{(h-k)}$.

The degree of redundancy with this strategy is $\delta(X) = K+1$, which is optimal by Theorem 3.1.

4.2.2 Failure Strategy

We introduce the failure strategy which will be used to restore the system to a K-tolerant state following the simultaneous failure of $q \leq K$ nodes. For illustration purposes, we will describe it when $K=1$, and than give the generalization to $K \geq 1$.

The failure of a node $S_k$ in the network, causes the following operation to be executed locally at every live node of the network:
Failure Strategy (Basic case). K=1

Failure Protocol (1), executed at node $S_j$:
update the current working indices failure set FWI. The new set is FWI U \{k\}

/* add k to the end of the FWI list */
For every block of data $F_h$ in $S_j$ that belonged also to $S_k$ do

Begin
select the leader node to be the live node with the smallest index $z$ ($z <$ k) from ML($F_h$)
select the receiver node to be the live node $S_r$ (following $S_k$ in the ordering) such that $S_r$ does not hold the data $F_h$
update ML($F_h$) to include the index $r$ of $S_r$ and remove the index $k$
set $\overline{FI}(F_h)[r] = \text{CurrentFailure}$

End
If ($z = j$)
send $< F_h, \text{ML}(F_h) \text{ and } \overline{FI}(F_h)[r] >$ to $S_r$

Let us illustrate the above failure strategy by an example.

Example 4.1: Consider the case where $n = 5$ and $K = 1$. Initially the data is partitioned over the network as follows:

\[
\begin{array}{ccccccc}
S_1 & S_2 & S_3 & S_4 & S_5 \\
FI & F_1 & F_2 & F_3 & F_4 & F_0 \\
\end{array}
\]
After applying the ND replication strategy, the resulting distribution is $X = <X_1, ..., X_5>$ where, $X_i = F_i \cup F_{i-1}$. Below we show the distribution as well as all the information associated with every block of data.

<table>
<thead>
<tr>
<th>Sl</th>
<th>S_1</th>
<th>S_2</th>
<th>S_3</th>
<th>S_4</th>
<th>S_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl</td>
<td>$F_1$</td>
<td>$F_2$</td>
<td>$F_3$</td>
<td>$F_4$</td>
<td>$F_0$</td>
</tr>
<tr>
<td></td>
<td>(1,2)</td>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td>(1,5)</td>
</tr>
<tr>
<td></td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
</tr>
</tbody>
</table>

**Initial Partition**

| Fl | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ |
|    | (1,5) | (1,2) | (2,3) | (3,4) | (4,5) |
|    | (0,0,0) | (0,0,0) | (0,0,0) | (0,0,0) | (0,0,0) |

**Initial Replication**

Let us consider the failure of node $S_1$:

FWI = \{1\} and CurrentFailure = 1.

<table>
<thead>
<tr>
<th>Sl</th>
<th>S_2</th>
<th>S_3</th>
<th>S_4</th>
<th>S_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl</td>
<td>$F_2$</td>
<td>$F_3$</td>
<td>$F_4$</td>
<td>$F_0$</td>
</tr>
<tr>
<td></td>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td>(1,5)</td>
</tr>
<tr>
<td></td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
</tr>
</tbody>
</table>

**Initial Partition**

| Fl | $F_1$ | $F_2$ | $F_3$ | $F_4$ |
|    | (1,2) | (2,3) | (3,4) | (4,5) |
|    | (0,0,0) | (0,0,0) | (0,0,0) | (0,0,0) |

**Initial Replication**

The execution of the proposed failure strategy follows:

1. At node $S_2$:

**Initial scan:** The set of blocks is scanned; it is found that the block of data $F_1$ belonged also to $S_1$.

**Select a leader:** $ML(F_1)$ is scanned and the leader (the live node in $ML(F_1)$ with the smallest index) is selected; in this case, the leader is $S_2$. 
Select a receiver: The receiver node $S_r$ is the live node following $S_1$ in ordering, such that $S_r$ does not hold a copy of $F_1$. In this case the receiver node is $S_3$.

Update ML and FI: $ML(F_1)$ is updated to include 3 (the index of node $S_3$) and to remove 1 (the index of node $S_1$); thus now $ML(F_1) = \{2,3\}$. $FI(F_1)[3] = 1$.

Data transmission: The leader node $S_2$ sends a copy of $F_1$, $ML(F_1)$, and $FI(F_1)$ to the receiver node $S_3$.

2. At nodes $S_3$ and $S_4$:

Initial scan: No blocks in common with $S_1$ are found; thus, no action is taken.

3. At node $S_5$:

Exactly as node $S_2$ using block $F_0$ instead of $F_1$.

After the Failure Protocol is executed locally at all live nodes of the network, the new data distribution is as follows:
It is easy to verify that the resulting distribution $X'$ has the following properties:

(a). The degree of redundancy $\delta(X') = 2$.

(b). The distribution $X'$ supports full availability of the original data in $X$,
     (i.e., any query that could be answered by $X$ can be answered by $X'$)

(c). The system is in a K-tolerant state.

If node $S_2$ fails now, then the blocks of data $F_1$ and $F_2$ will be sent to $S_4$ and the
block of data $F_0$ will be sent to $S_3$. FWI = \{1,2\} and CurrentFailure = 2.
The new distribution is clearly unbalanced with the data being clustered at the nodes neighbouring the nodes that have failed. It can be easily verified that the new resulting distribution \( X'' \) also satisfies the above properties.

In general, up to \( q \leq K \) failures can occur in the system at once. For simplicity of notation, we denote a node by its working index, and introduce the following notation:

At any given time, let

- \( D_i \) denote the set of data blocks stored at node \( i \)
- \( FS = \{ f_1, \ldots, f_q \} \) denote the set of the failed nodes sorted by their working indices.
- \( FD \) be the set of all data blocks \( F_h \) held by the set of failed nodes \( FS \)
- \( LS \) be the set of live nodes in the network
- \( \text{next}(c, L) \) be a function that takes a constant \( c \) and a set \( L \) of live nodes as input parameters, and returns a set \( R = \{ r_1 \ldots r_c \} \) of receiver nodes, such that \( r_m \) follows \( f_m \) in the ordering and does not contain \( F_h \).
- For each \( f \in FS \), let \( \text{rank}(f) \) denote its rank in \( FS \), and let \( cf(f) = \text{CurrentFailure} + \text{rank}(f) \)

As mentioned above, the simultaneous failure of \( FS = \{ f_1, \ldots, f_q \} \) will be logically seen as the sequence of failure of the \( f_i \)'s occurring according to their ranks in \( FS \).
Failure Strategy (Basic Case), $K \geq 1$

**Failure Protocol (2), executed at node $j$:**

Update the set of failed working indices, $FWI = FWI \cup FS$

For all the data blocks $F_h$ belonging to $FD \cap D_j$

1. Find the set of failed nodes $FS_h$ that had a copy of $F_h$:
   
   $FS_h = \{ f'_1, \ldots, f'_{q_h} \} \subseteq FS$

   where $q_h$ is the number of failed nodes holding a copy of $F_h$

2. Find the set of live nodes that have a copy of $F_h$:
   
   $M_h = ML(F_h) - FS_h$ (membership list of $F_h$ consisting of live nodes only)

3. Find the set of receiver nodes $R_h$, by applying the primitive "next" defined above: $R_h = next (q_h, LS - M_h) = \{ r_1, \ldots, r_{q_h} \} \subseteq LS$

4. Update the new membership list $ML(F_h)$ to remove the indices of the failed nodes and include the indices of the new receiver nodes: $ML(F_h) = M_h \cup R_h$

5. Find the leader node, to be the live node with minimum index from the list $M_h$:
   
   $l_h = \min \{ l : l \in M_h \}$

6. If the leader node $l_h$ is the same as the current node $j$, then send to each receiver node $r_i$, $1 \leq i \leq q_h$, the block of data $F_h$, the new $ML(F_h)$ and the value $cf(f'_i)$:
   
   send $<F_h, ML(F_h), cf(f'_i)>$ to $r_i$

**Theorem 4.1.** Given a system in a $K$-tolerant state, upon the failure of $q \leq K$ nodes, the execution of the following local failure strategy at all remaining live node of the network will restore the system to a $K$-tolerant state.
Proof: Let $F_h$ be a block held by $q_h$ failed nodes. Since the distribution is $K$-tolerant and $q \leq K$, then there exists at least one live node holding $F_h$. The block $F_h$ is sent to $q_h$ live nodes (receivers) which were not holding $F_h$; the transmission of data is performed only by one of the live nodes holding $F_h$ (the leader node). Thus, after the execution of the failure strategy, $F_h$ will be again held by $K+1$ live nodes. Since, this process is simultaneously performed for each such block $F_h$, the resulting distribution is $K$-tolerant.[]

Theorem 4.2. The above strategy is communication optimal.

Proof: All operations except the transmissions are done locally and do not involve any communication. Each $F_h$ is sent to the $q_h$ receivers only by the leader. Hence, for each block of $F_h \in FD$, the total number of transmissions is exactly $q_h$; i.e., the number of copies lost during this simultaneous failure. In other words, the total number of blocks transmitted is exactly equal to the number of blocks held by the failed nodes, which is optimal.[]

4.2.3 Recovery Strategy

We introduce the recovery strategy which will be used to restore the system to a $K$-tolerant state following the recovery of $q \geq 1$ arbitrary nodes. For illustration purposes, we will describe it when $K=1$, and than give the generalization to $K \geq 1$.

When a node recovers, it is assigned the working index that was added to the set of failed working indices FWI most recently. If the working index $k$ was the most recent
entry to FWI, then the recovery of any node $S$ is treated as if the node $S_k$ recovered, and $\text{FWI} = \text{FWI} - \{k\}$.

The recovery of a node $S_k$ in the network, causes the following local recovery operation to be executed at every live node in the network:

**Recovery Strategy (Basic case), K=1**

**Recovery Protocol (1), executed at node $j$:**

CurrentFailure = CurrentFailure - 1

$k =$ working index most recently added to FWI

FWI = FWI - $\{k\}$

the node $S$ that recovered is assigned the working index $k$

For all data blocks $F_h$ in $j$ do

Begin

If $\text{FI}(F_h)[j] \leftrightarrow \text{CurrentFailure}$

/* data block $F_h$ belonged to $j$ before to the last failure */

find $l$ (if it exists) such that $\text{FI}(F_h)[l] = \text{CurrentFailure}$ and reset $\text{FI}(F_h)[l] \text{ to NULL}$

update the node membership list by replacing $l$ with $k$

Else

reset $\text{FI}(F_h)[j]$ to NULL

update $\text{ML}(F_h)$ by replacing $j$ with $k$.

send $< F_h, \text{ML}(F_h) \text{ and } \text{FI}(F_h) \text{ > to } S_k$.

End
If \( q \geq 1 \) nodes recover simultaneously, then the above strategy is executed \( q \) times, once per recovered node. We will process these recovered nodes in an arbitrary sequential order.

**Example 4.2.** Now let us illustrate the above recovery strategy by an example. Consider the situation depicted Example 4.1 and assume that a node \( S \) recovers. Since \( S_2 \) was the last node that failed, the recovered node is assigned the working index 2. Let us stress that working index and initial name of a node (initial index) might not coincide; (e.g., the recovered node could be \( S_1 \))

\[ \text{FWI} = \{1, 2\} \text{ and } \text{CurrentFailure} = 2. \]

1. At node \( S_3 \):
   
   The FI of each block is scanned for entry values = CurrentFailure. A value 2 is found at FI(\( F_2 \))[4]. But since the value 2, did not occur at FI(\( F_2 \))[3], this means that the block of data \( F_2 \) belonged to this node before the last node failure. Thus, \( S_3 \) will not send \( F_2 \) to \( S_2 \). FI(\( F_2 \))[4] is set to Null, and ML(\( F_2 \)) is updated by replacing the index of node \( S_4 \) (4) with the index of node \( S_2 \) (2).

   Similarly FI(\( F_1 \))[4] is set to NULL, ML(\( F_1 \)) is updated by replacing index of node \( S_4 \) (4) with the index of node \( S_2 \) (2).

   A value 2 is found at FI(\( F_0 \))[3]. This means that node \( S_3 \) received the block \( F_0 \) after the failure of node \( S_2 \). Thus \( S_3 \) sends the block \( F_0 \) to \( S_2 \). FI(\( F_0 \))[3] is set to Null, and ML(\( F_0 \)) is updated by replacing the index of node \( S_3 \) (3) with the index of node \( S_2 \) (2).
2. At node $S_4$: Performing exactly the same operation as in node $S_3$: $S_4$ sends $F_2 \to$ node $S_2$, $Fl(F_2)[4]$ is set to NULL, and $ML(F_2)$ is updated by replacing the index of node $S_4$ (4) with the index of node $S_2$ (2). For Block $F_1$, $Fl(F_1)[4]$ is set to NULL, $ML(F_1)$ is updated by replacing the index of node $S_4$ (4) with the index of node $S_2$ (2).

3. At node $S_5$: $Fl(F_0)[3]$ is set to Null, and $ML(F_0)$ is updated by replacing the index of node $S_3$ (3) with the index of node $S_2$ (2).

After the recovery of $S_2$ the distribution looks as follows:

<table>
<thead>
<tr>
<th>Sl</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>$F_2$</td>
<td>$F_3$</td>
<td>$F_4$</td>
<td>$F_0$</td>
</tr>
<tr>
<td>$F_1$</td>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td>(2,5)</td>
</tr>
<tr>
<td>$F_2$</td>
<td>(.0,.0)</td>
<td>(.0,.0)</td>
<td>(.0,.0)</td>
<td>(.0,.0)</td>
</tr>
</tbody>
</table>

Initial Partition
ML
$Fl$

Initial Replication
ML
$Fl$

Rep. after a node failure
ML
$Fl$

It is easy to verify that the resulting distribution $X'$ has the following properties:

(a). The degree of redundancy $\delta(X') = 2$.

(b). The distribution $X'$ supports full availability of the original data in $X$,
(i.e., any query that could be answered by $X$ can be answered by $X'$)

(c). The system is in a K-tolerant state.

■
Theorem 4.3. Given a system in a $K$-tolerant state with $\text{CurrentFailure} \geq 1$, upon the recovery of a node $k$, the execution of the local recovery strategy at all live nodes of the network, will restore the system to a $K$-tolerant state. Furthermore, the recovery operation is a perfect undo of the latest failure operation.

Proof: When a node $S$ recovers, it is assigned the working index from FWI of the last node that failed, say $k$.

When $S_k$ failed, its blocks had been sent to a set of receiver nodes; these blocks are the only ones with failure indices equal to $\text{CurrentFailure}$.

By executing the recovery strategy, all live nodes $j$ in the system send to $S$ any block of data $F_h$ that they received during the last execution of the failure strategy. This is accomplished by locally scanning the array of failed indices FI and locating all data blocks $F_h$ with the $\text{CurrentFailure}$ level and sending them to $S$. Thus, after the execution of the recovery strategy, the system is in a $K$-tolerant state.

Observe that, $S$ receives all the blocks held by $k$ before its failure, and all nodes which had received blocks because of $k$'s failure remove this data. In other words, the recovery is a perfect undo of $k$'s failure.[1]

As mentioned earlier, when $q \geq 1$ nodes recover simultaneously, the recovery strategy is performed $q$ times, once for each recovered node. The execution of the strategy $q$ times corresponds to the undo of the $q$ latest failures. It might be possible that, in this process, a data block $F_h$ is transmitted to one or more intermediate nodes before reaching its final destination. This is functionally correct but not communication optimal.
As in the case of node failures, it is possible to process the recovery of the $q \geq 1$ nodes simultaneously in one local recovery operation. This will result in a more complex but transmission optimal strategy.

As mentioned above, the simultaneous recovery of nodes $r_1, ..., r_q$ will be logically seen as the sequence of recoveries of the $r_a$'s with each one being assigned the most recent entry in FWI.

Let:

* $q$ be the number of nodes that recovered

* $RS = \{r_1 ..., r_q \}$ be the set of recovered nodes

* $\text{recovery\_range} = [\text{CurrentFailure} - |q| + 1, \text{CurrentFailure}]$
Recovery Strategy (Basic case) \( K \geq 1 \)

**Recovery Protocol(2), executed at node \( j \):**

1. remove from FWI the most recent \( q \) entries and assign them to the recovered nodes
2. For all \( F_h \) in \( j \) do:
   
   If there is in \( F_i(F_h) \) at least a value in the recovery_range do:
   
   2.1 let \( u \) be the smallest value in \( F_i(F_h) \) which is in the recovery_range
   
   2.2 let \( v \) be the largest value in \( F_i(F_h) \) which is in the recovery_range
   
   2.3 set receiver to be \( r (CurrentFailure - u + 1) \) (see note)
   
   2.4 set sender to be the index of \( F_i \), such that \( F_i(F_h)[sender] = v \)
   
   2.5 add receiver to ML\((F_h)\)
   
   2.6 For every \( p \) such that \( F_i(F_h)[p] \) is in the recovery range do
   
   2.6.1 remove \( p \) from ML\((F_h)\)
   
   2.6.2 If \( F_i(F_h)[p] \neq 0 \) then set \( F_i(F_h)[p] = Null \)
   
   2.7 set \( F_i(F_h)[receiver] = CurrentFailure - u \)
   
   2.8 If \( j = sender \)
   
   2.8.1 send \(<F_i, ML(F_h), F_i>\) to the receiver node

3. \( CurrentFailure = CurrentFailure - q \)

Note: If \( CurrentFailure = q \), there are multiple receiver nodes. In this case, in Step 2.3, receiver is a set, and in Step 2.7, the sender sends the data to all the receiver nodes.
Example 4.3: Let us illustrate the above strategy by an example. In this case, we consider \( n = 6 \) and \( K = 2 \).

The original distribution looks as follows:

<table>
<thead>
<tr>
<th>SL</th>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>( S_3 )</th>
<th>( S_4 )</th>
<th>( S_5 )</th>
<th>( S_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FL</td>
<td>( F_1 )</td>
<td>( F_2 )</td>
<td>( F_3 )</td>
<td>( F_4 )</td>
<td>( F_5 )</td>
<td>( F_0 )</td>
</tr>
<tr>
<td></td>
<td>(1,2,3)</td>
<td>(2,3,4)</td>
<td>(3,4,5)</td>
<td>(4,5,6)</td>
<td>(5,6,1)</td>
<td>(6,1,2)</td>
</tr>
<tr>
<td></td>
<td>(0,0,0,0)</td>
<td>(0,0,0,0)</td>
<td>(0,0,0,0)</td>
<td>(0,0,0,0)</td>
<td>(0,0,0,0)</td>
<td>(0,0,0,0)</td>
</tr>
</tbody>
</table>

| FL | \( F_0 \) | \( F_1 \) | \( F_2 \) | \( F_3 \) | \( F_4 \) | \( F_5 \) |
|    | (6,1,2)  | (1,2,3)  | (2,3,4)  | (3,4,5)  | (4,5,6)  | (5,6,1)  |
|    | (0,0,0,0) | (0,0,0,0) | (0,0,0,0) | (0,0,0,0) | (0,0,0,0) | (0,0,0,0) |

| FL | \( F_0 \) | \( F_1 \) | \( F_2 \) | \( F_3 \) | \( F_4 \) |
|    | (5,6,1)  | (6,1,2)  | (1,2,3)  | (2,3,4)  | (3,4,5)  |
|    | (0,0,0,0) | (0,0,0,0) | (0,0,0,0) | (0,0,0,0) | (0,0,0,0) |

Initial Partition
ML
FI

Initial Replication
ML
FI

After the failure of nodes \( S_1 \) and \( S_2 \), the distribution is represented by:
Now let us consider the simultaneous recovery of nodes $S_1$ and $S_2$.

The set of recovered nodes is \{ $r_1 = S_2$, $r_2 = S_1$ \}

The recovery strategy is executed locally at every live node of the network. For simplicity, we will focus on the execution of the protocol at a given block of data at a given node. Consider the execution of the protocol for block $F_1$ at node $S_5$.

**Define the recovery range:** The recovery range is $(r_r) = [1 \ldots 2]$.

**Find in FL the values that are in the recovery range:**

$FL(F_1)[4] = 1$ and $FL(F_1)[5] = 2$. Both values $\in r_r$.

**Define u:** $u = 1$

**Define v:** $v = 2$.

**Find receiver node:** The receiver node is $r_{(2\cdot1+1)} = r_2$.

**Find sender node:** The sender node is $S_5$. 
Since the number of recovered nodes, \( q = 2 \), equals the value of Current\( \text{Failure} \), there is full network recovery. In this case \( F_1 \), will be sent to all the node with \( \text{FI}(F_1) \) value of 0, i.e., nodes 1,2,3.

**Find new ML:**

\[
\text{ML}(F_1) = (3,4,5) \cup (1,2,3) - (4,5) = (1,2,3)
\]

Since \( j = 5 \), \( S_5 \) sends \( F_1 \), \( \text{ML}(F_1) \) and \( \text{FI}(F_1) \) to \( S_1 \) and \( S_2 \) and \( S_3 \).

Similarly, \( F_2 \) is sent to \( S_2 \), \( S_3 \) and \( S_4 \) from node \( S_5 \); \( F_3 \) is sent to nodes \( S_1, S_5, S_6 \) from node \( S_4 \); and \( F_6 \) is sent to nodes \( S_1, S_2, S_6 \) from node \( S_4 \).

**Theorem 4.4:** Given a system in a K-tolerant state, upon the simultaneous recovery of \( q \geq 1 \) nodes, the execution of the local recovery strategy at all live node of the network, will restore the system to a K-tolerant state.

**Proof:** Let \( \{ r_1, \ldots, r_q \} \) recover simultaneously. If \( q = 1 \), the theorem trivially holds. Let \( q > 1 \). The only difference in the two strategies occurs if, in Recovery Strategy 1 (RS1), a block sent to a recovered node \( r_j \) is subsequently transmitted by \( r_j \) to another recovered node. Let \( F_h \) be such a block; and let \( r_{a1}, r_{a2}, \ldots, r_{ai} \) be the sequence of all recovered nodes such that \( r_{a1} \) sent \( F_h \) to \( r_{a(i+1)} \) in RS1. Obviously, \( F_h \) will be stored at \( r_{ai} \) with failure index equal to \( \text{(Current\text{Failure} - ai)} \).

Let \( i \) be the node which in RS1 sent \( F_h \) to \( r_{a1} \); at \( i \), \( \text{FI}(F_h) \) must have included the values:

\[
\text{Current\text{Failure} - a_1 + 1, Current\text{Failure} - a_2 + 1, Current\text{Failure} - a_3 + 1, \ldots, Current\text{Failure} - a_i + 1} \text{ (all these values are in the recovery range).}
\]
In the Recovery Strategy 2 (RS2) node \( i \) would determine \( u = \text{CurrentFailure} - a_i + 1 \) and send \( F_h \) directly to the node \( r(\text{CurrentFailure} - u + 1) = r a_i \).\]

**Theorem 4.5:** The above Recovery Strategy (2) is communication optimal

**Proof:** All operations except the transmissions are done locally and do not involve any communication. Each Block which has to be transmitted is sent to only one receiver node when \( \text{CurrentFailure} > q \). Thus the total number of transmissions is exactly equal to the number of blocks that need to be transmitted to undo the last \( q \) failures, where \( q \) is the number of recovered nodes.

In the case that \( \text{CurrentFailure} = q \), i.e., case of total recovery, a transmitted data block \( F_h \) is also sent to all nodes with failure index 0. \]

**4.2.4 Summary and Concluding Remarks**

This chapter dealt with basic continuously K-tolerant distributions with no restrictions imposed on the data. We presented a mechanism \( M \) consisting of the ND Replication strategy (presented in Chapter 3), a Failure Protocol and a Recovery Protocol that keep the distribution in a continuously K-tolerant state after the simultaneous failure of \( q \leq K \) or the recovery of \( p \geq 1 \) nodes in the system, respectively.

**Theorem 4.6.** The mechanism \( M \) (ND, Failure, Recovery) consisting of the ND Replication Strategy, the Failure and Recovery Protocols discussed in Section 4.2 is continuously K-tolerant and communication optimal.
**Proof:** From Theorems 4.1, 4.2, 4.4 and 4.5.]

With every block of data $F_h$ stored at a node $S_i$ is associated a node membership list of size $K+1$ and an array $FI$, array of failed indices, of size $n$. If $n$ is very large and $K$ relatively small, then $FI$ can be implemented as a list, since at most 2 $(K+1)$ entries are stored in the list. $K+1$ entries are required for the original 0 level failure, and $K+1$ entries for the subsequent $K$ failures.

These lists are scanned in $O(K)$ time during the execution of the protocols. Thus the storage and computation overhead associated with the structure chosen is a reasonable price to pay in order to achieve the results presented above.

This strategy makes no attempt to keep a specific ordering on the data: data are sent to the first available node. Thus, this strategy can be used in the case where applications do not depend on the size of the data stored in a node or on the ordering of the data.

If nodes with consecutive working indices fail in the network, then the data is clustered in the first few neighbouring nodes. The data will be unevenly distributed over the network, and the distribution will be completely unbalanced.

If it is required to keep the data distribution balanced, then starting with a balanced distribution $X$, the Failure and Recovery protocols must transmit data in such a way that the resulting distribution is balanced, which is the topic of the next section.

Consider a network $S$ with $n$ nodes, a set of data $F$ and a partition $<F_1, F_2, ..., F_n>$ of $F$ over $S$, where $F_i$ denotes the block stored at $S_i$. We would like to maintain a continuously K-tolerant system with the following properties:

a. There is no ordering requirement on the data set $F$ stored in the network.

b. There is requirement to keep the distribution $X$ balanced.

**Definition 4.5:** A block transformation $t$ is the process of transforming a distribution $X$ into a distribution $X'$, by moving a block of data, say $F_i$, from a node $S_i$ to a node $S_j$, which does not hold a copy of that block.

Let $T(X)$ be the set of all the distributions obtainable by applying block transformations to $X$.

Let $x_i$ denote the number of blocks held by $S_i$ in $X$.

**Definition 4.6:** Given a distribution $X$, the unbalance of $X$, $UB(X) = \max\{|x_i - x_j|\}$ for all $i,j$.

**Definition 4.7:** Given a distribution $X$, we say that $X$ is balanced if and only if:

$$\forall \ X' \in T(X) \quad UB(X') \geq UB(X)$$
In this section we define a mechanism $M$ consisting of the BND replication strategy, which introduces replication to a balanced distribution as defined above, resulting in another balanced distribution. We also define the failure and recovery protocols of this mechanism that maintain a balanced and continuously $K$-tolerant distribution after their local execution at every live node.

### 4.3.1 Balanced Neighbouring Data (BND) Replication Strategy

Initial $K$-tolerance is achieved by introducing the BND replication strategy as follows:

**BND Replication Strategy**: At every node $S_i$ we replicate the block of the previous $K$ nodes. Thus the data stored at node $S_i$ is: $X_i = F_h \cup F_{(h-1)} \ldots \cup F_{(h-k)}$.

Assuming an original balanced partition, the resulting data (original data and replicated data) is balanced throughout the network.

The degree of redundancy with this strategy is $\delta(X) = K+1$, which is optimal by Theorem 3.1.

### 4.3.2 Failure Strategy

To maintain a $K$-tolerant distribution, following the failure of a node, instances of blocks of data (that belonged to the node that failed) are transmitted to live nodes not holding a copy of the block being sent to them. In addition, in order to maintain the distribution balanced, the blocks of data are transmitted to live nodes with minimum size (i.e., with minimum numbers of data blocks). The entire process is coordinated
by a specific node, called the *manager*, which is chosen according to a predefined scheme e.g., the node with smallest index ($S_n$).

The failure strategy is executed locally at every live node $S_j$. For every block of data $F_h$ common to $S_j$ and the node that failed, we select a leader node from the node membership list of $F_h$, say the node with the smallest index different than the node that failed.

If $S_j$ is selected as the leader node for $F_h$, then $S_j$ sends to the manager the index $h$ of the data block $F_h$, the node membership list associated with $F_h$ and the number of blocks it holds. Otherwise, $S_j$ sends to the manager the number of blocks it holds.

Once the manager receives all the information associated with a data block (from a leader) and the number of blocks held by every live node, it finds the receiver node (the live node with smallest number of blocks not holding a copy of the data block $F_h$). The manager sends to the leader the index of block $h$, the index of the receiver node $r$ (where a copy of $F_h$ should be sent) and the failure_level value which is used in order to update the array of failed indices of $F_h$.

Once the leader $S_j$ receives the message from the manager, the node membership list of $F_h$ is updated to include the index of the receiver node $r$ and to remove the index of the failed node, and the array of failed indices is updated so that $FI(F_h)[r] = \text{failure_level}$.

The updated $ML(F_h)$ and $FI(F_h)$ are sent to every node belonging in the node membership list of $F_h$. In addition, an instance of the data block $F_h$ is sent to the receiver node $S_r$.

We now define the following types of messages:
1. <STM, h, ML(h), number_of_blocks>. Every live node sends this message to the manager. This message consists of the identifier STM (Send To Manager), the index of a data block h, the node membership list, ML, associated with the data block and the number of blocks in the node. The manager receives the above message from every live node.

2. <RFM, h, r, failure_level>. Every leader node (of a block F_h), receives from the manager (RFM) the index h of the data block F_h, the index r of the receiver node, and the failure_level value that is used to update FI(F_h). The manager sends the above message to every leader node.

3. <STN, Block, ML(F_h), FI(F_h)>. Once the information of F_h is updated, the leader sends ML(F_h) and FI(F_h) to all the nodes that belong to ML(F_h). In addition the data block F_h is sent to the receiver node S_r. Live nodes receive from leaders the above message.

The failure of a node S_k in the network, causes the following operation to be executed locally at every live node of the network:
Failure Strategy (balanced case), K=1

Failure Protocol executed at node S_j:

update the current working indices failure set FWI: FWI = FWI ∪ {k}

/**< Add k at the end of the set */

let number_of_blocks = the number of data blocks in S_j

For every Block of data F_h in S_j that belonged also to S_k do

Begin

select the leader node to be the live node with the smallest index z (< k) from ML(F_h)

If (z = j)

Begin

send < STM, h, ML(F_h), number_of_blocks >

receive < RFM, h, r, failure_level >

update ML(F_h), by replacing the index k by the index r

update F_l, by setting F_l(F_h)[r] = failure_level

send < STN, Data, ML(F_h), F_l(F_h) >

End

Else

Begin

send < STM, 0, NULL, number_of_blocks >

receive < STN, Data, ML(F_h), F_l(F_h) >

End

End
The broadcast is done in this case, because we cannot locally determine the receiver node with minimum number of data blocks. For the same reason, it is also necessary to send the updated node membership list and array of failed indices to all the nodes that hold a copy of the data block $F_h$. This is performed by the message send $<STM,...>$. 

If $n$ is the number of nodes in the network and $f$ is the number of failed nodes, then $(n-f)$ messages are sent to the manager.

If $B$ is the number of blocks in the failed nodes, then $B$ messages are sent by the manager to the leader nodes.

Every leader node sends $K$ messages to other live nodes.

Thus, the execution of the failure strategy involves $[(n-f) + B + K]$ messages.

Example 4.4: Let us illustrate the above failure strategy by an example. As before, we consider the case where $n = 5$ and $K = 1$.

After applying the ND replication strategy, the resulting distribution is $X = <X_1, ..., X_5>$ where, $X_i = F_i \cup F_{(i-1)}$. Below we show the distribution as well as all the information associated with every block of data.
Now let us consider the failure of node $S_1$ in the network:

<table>
<thead>
<tr>
<th>Sl</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl</td>
<td>$F_1$</td>
<td>$F_2$</td>
<td>$F_3$</td>
<td>$F_4$</td>
<td>$F_0$</td>
</tr>
<tr>
<td></td>
<td>(1,2)</td>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td>(1,5)</td>
</tr>
<tr>
<td></td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
</tr>
</tbody>
</table>

Initial Partition
ML
Fl

<table>
<thead>
<tr>
<th></th>
<th>$F_0$</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>$F_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1.5)</td>
<td>(1.2)</td>
<td>(2.3)</td>
<td>(3.4)</td>
<td>(4,5)</td>
</tr>
<tr>
<td></td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
</tr>
</tbody>
</table>

Initial Replication
ML
Fl

Failure strategy:

Select a manager: Being the live node with the smallest index, $S_2$ is chosen as the manager to coordinate the failure operation.

All live nodes send the message $<\text{STM, ... }>\,$ to the manager.

1. At node $S_2$:

Send to manager: $S_2$ scans its data, and finds the block $F_1$ in common with $S_1$. $\text{ML}(F_1)$ is scanned and the leader (the live node in $\text{ML}(F_1)$ with the minimum index) is selected; in this case, the leader is $S_2$.

Node $S_2$ being the manager (does not send itself any information but) is aware of the following:

data block index: 1

$\text{ML}(F_1)$: (1,2)
size: 2

2. At node S₃:
   **Send to manager:** in S₃ no blocks in common with S₁ are found, thus S₃ sends the manager its size.
   size: 2

3. At node S₄ :
   **Send to manager:** in S₄ no blocks in common with S₁ are found, thus S₃ sends the manager its size.
   size: 2

4. At node S₅ :
   **Send to manager:** S₅ (being a leader node for F₀) sends the manager the following information:
   data block index: 0
   ML(F₀): (1,5)
   size: 2

The manager (S₂) has now all the information required to manage the redistribution of the data in order to keep the distribution balanced. Data block F₁ from node S₂ can be sent to any of the nodes S₃, S₄ or S₅, since none of these nodes have a copy of F₁ and their size |S₃|=|S₄|=|S₅|=2. Say that F₁ is to be sent to S₃.
Data block $F_0$ from node $S_5$ can be sent to any of the nodes $S_2$, $S_3$, $S_4$, since none of these nodes have a copy of $F_0$. But since $|S_3| = 3$ and $|S_2| = |S_4| = 2$, $F_0$ is sent to either $S_2$ or $S_4$. Say that $F_0$ is sent to $S_2$.

Now that the data transmission issue is resolved, the manager sends the message $<RFM, ..., >$ to all the leader nodes. This operation has the following effect: the manager notifies node $S_2$ to send the block of data $F_1$ to node $S_3$, and $S_5$ to send the block of $F_0$ to node $S_2$.

5. At node $S_2$:

**Data update/propagation:** $ML(F_1)$ in $S_2$ is updated; the new $ML(F_1) = (2,3)$. $FI(F_1)$ is updated to reflect the current failure level, $FI(F_1)[3]$ = failure_level = 1.

$S_2$ sends to $S_3$ (the receiver node) the data block $F_1$ along with $ML(I_1)$ and $FI(F_1)$. $ML(F_1)$ and $FI(F_1)$ are also sent to all the nodes (if any) that are members of $ML(F_1)$.

6. At node $S_5$:

**Data update/propagation:** The above operation is also performed for the block $F_0$ at $S_5$ resulting in the following distribution:
<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>Initial Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F2</td>
<td>F3</td>
<td>F4</td>
<td>F0</td>
<td>ML</td>
</tr>
<tr>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td>(2,5)</td>
<td></td>
<td>FI</td>
</tr>
<tr>
<td>(0,0,0,)</td>
<td>(0,0,0,)</td>
<td>(0,0,0,)</td>
<td>(0,1,0,0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F0</th>
<th>Initial Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F0</td>
<td>F2</td>
<td>F3</td>
<td>F4</td>
<td>ML</td>
</tr>
<tr>
<td>(2,3)</td>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td></td>
<td>FI</td>
</tr>
<tr>
<td>(0,0,1,0,)</td>
<td>(0,0,0,)</td>
<td>(0,0,0,)</td>
<td>(0,0,0,)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F0</th>
<th>F2</th>
<th>F1</th>
<th>Rep. after a node failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F0</td>
<td>F2</td>
<td>F1</td>
<td>(2,3)</td>
<td>ML</td>
</tr>
<tr>
<td>(2,5)</td>
<td>(2,3)</td>
<td></td>
<td>(0,1,0,0)</td>
<td>(0,1,1,0,)</td>
<td>FI</td>
</tr>
</tbody>
</table>

It is easy to verify that the new distribution $X'$ has the following properties:

(a). The degree of redundancy $\delta(X') = 2$.
(b). The distribution $X'$ supports full availability of the original data in $X$, (i.e., any query that could be answered by $X$ can be answered by $X'$)
(c). The system is in a K-tolerant state.
(d). $X'$ satisfies the balancing criteria defined above.

If the node $S_2$ fails, then the block of data $F_2$ will be added to the node $S_4$, the block of data $F_1$ will be added to the node $S_5$, and the block of data $F_5$ will be added to either node $S_3$ or $S_4$. The new distribution is clearly balanced. The resulting distribution $X''$ will also satisfy the properties defined above. Thus the system is again in a K-tolerant state.

The generalization to the case when $q \leq K$ nodes simultaneously fail in the network is straightforward. When $q$ nodes fail, the manager is chosen to be the live node with the smallest index. The manager treats the $q$ failed nodes as successive
failures using a mechanism to assign failure indices similar to the one described in Section 4.2.

**Theorem 4.7.** Given a system in a K-tolerant state, upon the failure of \( q \leq K \) nodes, the execution the local failure strategy at all remaining live nodes of the network will restore the system to a K-tolerant state; furthermore the distribution is balanced.

**Proof:** The proof that the resulting distribution is K-tolerant follows the same lines as for the unbalanced case. To show that the final distribution is balanced, first consider the case where \( q = 1 \). Since each block on the failed node is sent to the live node with minimum size, who does not have that block, the resulting distribution is balanced. For \( q > 1 \), the simultaneous failures are treated by the manager as successive ones, hence the resulting distribution is balanced.[]

**Theorem 4.8.** The above strategy is block-communication optimal.

**Proof:** The failure operation as described above is block communication optimal. i.e., the minimum number of blocks are transmitted between the nodes in order to make the distribution balanced and K-tolerant. All communications between the manager and the leader nodes do not involve the transmission of data blocks. Every leader transmits exactly one data block to the new receiver node. And since the number of leaders equal the number of blocks in the failed nodes (i.e., the minimum number of nodes required to replicate in order to make the distribution K-tolerant) the operation is block communication optimal.[]
4.3.3 Recovery Strategy

In Section 4.2 we described a general recovery protocol that performs a perfect undo of the last failure operation. The recovery operation is dependent on the data structure used by the model but independent from the specifics of the failure strategy.

In this section, we use the same data structures as the previous model. In addition, in this model, the failure of a node leads to a balanced distribution (due to the failure strategy that keeps distributions balanced). \( T^* \)'s, the recovery operation (which is a perfect undo operation of the last failure operation) will result in a balanced distribution.

4.3.4 Summary and Concluding Remarks

This section dealt with continuously K-tolerant distributions with balanced load restriction imposed on the data. We presented a mechanism \( M \) consisting of the BND Replication Strategy (based on the replication technique presented in Chapter 3), a Failure Protocol and a Recovery Protocol that keep the distribution in a continuously K-tolerant balanced state after the failure of \( q \leq K \) or the recovery of \( n \geq 1 \) nodes in the system, respectively.
Theorem 4.9. The mechanism $M$ (BND, Failure, Recovery) consisting of the BND Replication Strategy, the Failure and Recovery Protocols discussed in Section 4.3 is continuously K-tolerant and block communication optimal. Further more applying the mechanism $M$ to a balanced distribution $X$, preserves the balancing of $X$.

Proof: From Theorems 4.4, 4.5, 4.7 and 4.8.[]

In this model, where data balancing is important, we could refine the size of the unit element $F_h$. For example, if $F_h$ consists of $w$ pages, we could consider the unit element to be a page rather than the whole block $F_h$. When a node $S_j$ fails, instead of shipping the whole block of data to a node $S_i$, we just ship one page. The process is repeated for the remaining pages.

In a similar fashion, we can refine the unit element further and consider it to be a record rather than a page. Every page consists now of a collection of records. When a node $S_j$ fails, we can just ship one record. We then continue the process until all records, all pages or all blocks are transmitted.

The finer the unit element is, the more balanced the resulting distribution is. However, this is achieved at the expenses of computation time and storage. In fact, it is required to keep a node membership list and an array of failed indices for every unit element; furthermore, these lists need to be updated every time a unit element is shipped to another node in the network. There is a clear tradeoff between the time complexity of the failure operation and the size of the unit element that is maintained in order to maximize the balancing of the distribution.
If it is required to keep the distribution ordered rather than balanced, then starting with an ordered distribution $X$, the failure and recovery protocols must transmit the data in such a way that the resulting distribution is ordered, which is the topic of the next section.
4.4 Unbalanced Distribution, with Ordered Data Set F

Consider a network $S$ with $n$ nodes, a set of data $F$ and a partition $< F_1, F_2, ..., F_n >$ of $F$ over $S$, where $F_i$ denotes the block stored at $S_i$. We would like to maintain a continuously K-tolerant system with the following properties:

a. There is an ordering requirement on the data set $F$ stored in the network.

b. There is no requirement to keep the distribution $X$ balanced.

**Node ordering:** As mentioned before, there exists a natural ordering "->" on the nodes based on their indices; i.e., $S_i$ is followed by $S_{i+1}$. We assume the ordering of the nodes to be circular; i.e., $S_n$ is followed by $S_1$. This ordering extends in a natural way to any subset $S' \subseteq S$ of nodes. In the following, we will refer to the subset $S'$ of live nodes and use the term "consecutive" to describe the node $x \in S'$ which follows a given node $y \in S'$ according to their ordering.

If the indices are consecutive then we will denote the ordering by "<->".

**Block ordering:** There exists a cyclic ordering "->" among the blocks; without loss of generality let $F_1 -> F_{i+1}$ and $F_n -> F_1$.

**Definition 4.8:** A K-tolerant distribution is **ordered** if it satisfies the following properties:

a. Each block is stored at $K+1$ consecutive live nodes.

b. Each node contains consecutive blocks of data.
In this section we define a mechanism $M$ consisting of the UOND replication strategy; note that if the initial partition is ordered (i.e., blocks are assigned in increasing order to consecutive nodes), this strategy does create an initial $K$-tolerant ordered distribution. We also define the failure and recovery protocols of this mechanism that maintain an ordered and continuously $K$-tolerant distribution after their local execution at every live node.

4.4.1 Unbalanced, Ordered, Neighbouring Data (UOND) Replication Strategy

Initial $K$-tolerance is achieved by introducing the UOND replication strategy as follows:

**UOND Replication Strategy:** At every node $S_i$ we replicate the block of the previous $K$ nodes. Thus the data stored at node $S_i$ is: $X_i = F_h \cup F_{(h-1)} \ldots \cup F_{(h-K)}$.

Assuming an original ordered partition, the resulting distribution is also ordered. The degree of redundancy with this strategy is $\delta(X) = K+1$, which is optimal by Theorem 3.1.

4.4.2 Failure Strategy

To maintain a $K$-tolerant distribution following the failure of a node, say $S_k$, instances of blocks that belonged to $S_k$ are transmitted to live nodes not holding a copy of the
block being sent to them. To maintain the distribution ordered, each data block instance that belonged to \( S_k \) is transmitted to the first live node that directly follows \( S_k \) (in its ordering).

The failure strategy is executed locally at every live node \( S_j \). For every block of data \( F_h \) common to \( S_j \) and \( S_k \), we select a leader node from the node membership list of \( F_h \), say the node other than \( S_k \) with the smallest index. The receiver node \( S_r \), is selected to be the first node following \( S_k \) and not holding a copy of the block of data \( F_h \). The node membership list of \( F_h \) is updated to include the index \( r \) of \( S_r \) and to remove the index \( k \) of \( S_k \). The array of failed indices is updated as follows: 

\[
FI(F_h)[r] = \text{CurrentFailure.}
\]

If \( S_j \) is selected as the leader node for \( F_h \), then \( S_j \) sends to \( S_r \), \( F_h \), \( ML(F_h) \), and \( FI(F_h) \).

The failure of a node \( S_k \) in the network causes the following operation to be executed locally at every live node of the network:
Failure Strategy (Ordered case) K=1

Failure Protocol, executed at node $S_j$:
update the current working indices failure set FWI. The new set is FWI $\cup \{k\}$

/\* add $k$ to the end of the FWI list /\*

For every block of data $F_h$ in $S_j$ that belonged also to $S_k$ do

Begin

select the leader node to be the live node with smallest index $z$ ($z \leftrightarrow k$) in ML($F_h$)
select the receiver node to be the first live node $S_r$ (following $S_k$ in the ordering) such that $S_r$ does not hold a copy of the data $F_h$

update ML($F_h$) to include the index $r$ of $S_r$ and remove the index $k$

Set FL($F_h$)$[r] = \text{CurrentFailure}$

end

If ($z = j$)

Send $< F_h, ML(F_h) \text{ and } FL(F_h) >$ to $S_r$

Example 4.5: As before, we consider $n = 5$ and $k = 1$. The initial distribution looks as follows:
<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl</td>
<td>FI</td>
<td>F_1</td>
<td>F_2</td>
<td>F_3</td>
</tr>
<tr>
<td>(1,2)</td>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td>(1,3)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F_0</th>
<th>F_1</th>
<th>F_2</th>
<th>F_3</th>
<th>F_4</th>
<th>Initial Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,5)</td>
<td>(1,2)</td>
<td>(2,3)</td>
<td>(3,4)</td>
<td>(4,5)</td>
<td>ML</td>
</tr>
<tr>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>Fl</td>
</tr>
</tbody>
</table>

Now, we consider the failure of node $S_3$ in the network:

$FWI = \{3\}$ and $Current\text{Failure} = 1$.

The execution of the proposed failure strategy follows:

1. At node $S_2$:
   - **Initial scan:** The set of blocks is scanned, it is found that the block of data $F_2$ belongs also to $S_3$.
   - **Select a leader:** $ML(F_2)$ is scanned and the leader (the live node in $ML(F_2)$ with the minimum index) is selected; in this case the leader is $S_2$.
   - **Select a receiver:** The receiver node $S_r$ is the first live node following $S_3$ in ordering, not holding an instance of $F_2$. In this case the receiver node is $S_4$.
   - **Update ML and FI:** $ML(F_2)$ is updated to include 4 (the index of node $S_4$) and to remove 3 (the index of node $S_3$); thus the new $ML(F_2) = \{2,4\}$. $FI$ is updated by setting $FI(F_2)[4] = 1$.
   - **Data transmission:** The leader node $S_2$ sends a copy of $F_2$, $ML(F_2)$, and $FI(F_2)$ to the receiver node $S_4$.

2. At nodes $S_1$ and $S_5$:...
Initial scan: No blocks in common with \( S_3 \) are found; thus, no action is taken.

3. At node \( S_4 \):

Exactly as node \( S_2 \) using block \( F_3 \) (which is transmitted to \( S_5 \)).

The new distribution looks as follows:

<table>
<thead>
<tr>
<th>( S_1 )</th>
<th>( S_2 )</th>
<th>( S_4 )</th>
<th>( S_5 )</th>
<th>Initial Partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_1 )</td>
<td>( F_2 )</td>
<td>( F_4 )</td>
<td>( F_0 )</td>
<td>ML</td>
</tr>
<tr>
<td>(1,2)</td>
<td>(2,4)</td>
<td>(4,5)</td>
<td>(1,5)</td>
<td>FI</td>
</tr>
<tr>
<td>(0,0,0)</td>
<td>(0,0,1)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( F_0 )</th>
<th>( F_1 )</th>
<th>( F_3 )</th>
<th>( F_4 )</th>
<th>Initial Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,5)</td>
<td>(1,2)</td>
<td>(4,5)</td>
<td>(4,5)</td>
<td>ML</td>
</tr>
<tr>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>FI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( F_2 )</th>
<th>( F_3 )</th>
<th>Replication after node failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,4)</td>
<td>(4,5)</td>
<td>ML</td>
</tr>
<tr>
<td>(0,0,1)</td>
<td>(0,0,1)</td>
<td>FI</td>
</tr>
</tbody>
</table>

It is easy to verify that the resulting distribution \( X' \) has the following properties:

(a). The degree of redundancy \( \delta(X') = 2 \).

(b). The distribution \( X' \) supports full availability for the original data in \( X \)

(i.e., any query answered by \( X \) can be answered by \( X' \))

(c). The system is in a K-tolerant state.

(d). \( X' \) satisfies the ordering criteria defined above, i.e.:

a. Every block is stored in 2 \((K+1)\) consecutive nodes.

b. Each node contains consecutive blocks of data.
Other than the fact that the blocks of data are sent to the first live node that follows in the ordering the failed node, the failure strategy of this model is exactly the same strategy as the one of the model of Section 4.2 (unordered, unbalanced). Thus, the generalization to the case where \( q \leq K \) nodes simultaneously fail follows the same lines as for the unbalanced, unordered case.

**Theorem 4.10.** Let \( S_a \) and \( S_b \) be two consecutive live nodes. If \( F_j \in S_a \) and \( F_{j-1} \in S_b \) then:

(i). \( F_{j-1} \in S_a \)

(ii). \( F_j \in S_b \)

**Proof:** The theorem holds before the occurrence of any failure, based on the UOND replication strategy and the definition of data ordering. Let the theorem hold at a point in time and let a failure occur; we will show that it holds after the application of the failure strategy.

We will first prove (i). Consider the failure of node \( S_k \) and by contradiction, after the application of the failure strategy, let \( F_j \in S_a \), \( F_{j-1} \in S_b \), but \( F_{j-1} \notin S_a \). Since before the failure of \( S_k \) the theorem held, \( F_{j-1} \) was not in \( S_b \) before the failure. This implies that \( F_{j-1} \in S_k \) and it was sent to \( S_b \) by the failure strategy.

We now consider two cases:

1. \( S_a \leftrightarrow S_k \leftrightarrow S_b \):

   The node \( S_k \) had \( F_{j-1} \) before the application of the failure strategy. If the node \( S_a \) had the data block \( F_j \) and not the block \( F_{j-1} \) before the failure of \( S_k \), this would
contradict the fact that the theorem held true before the failure. Thus the node \( S_a \) must have the block \( F_{j-1} \).

2. \( S_k \rightarrow S_a \rightarrow S_b \):

The node \( S_k \) had \( F_{j-1} \) before the application of the failure strategy. Upon the failure of \( S_k \), the failure strategy sends \( F_{j-1} \) to the first node that does not have it, in this case to \( S_b \). This implies that the node \( S_a \) must have had the block \( F_{j-1} \) before the failure of \( S_k \), otherwise this would contradict the data block placement of the failure strategy. This concludes the proof of part 1 of the theorem that \( F_{j-1} \in S_a \).

Now we will prove part (ii). Consider the failure of node \( S_k \) and by contradiction, after the application of the failure strategy, let \( F_j \in S_a \), \( F_{j-1} \in S_b \) but \( F_j \notin S_b \). This means that \( F_j \notin S_b \) before the failure of \( S_k \). But since the theorem held before the failure of \( S_k \), this also means that \( F_{j-1} \) was not in \( S_b \) before the failure of \( S_k \). This implies that \( F_{j-1} \in S_k \) and it was sent to \( S_b \) by the failure strategy.

We now consider two cases:

1. \( S_a \leftrightarrow S_k \leftrightarrow S_b \):

The node \( S_k \) had \( F_{j-1} \) before the application of the failure strategy. We also know that \( F_j \in S_a \). But since the theorem held true before the failure of \( S_k \), this implies that \( F_j \in S_k \) as well. Since \( S_b \) is the first live node following \( S_k \) in the ordering which does not hold a copy of \( F_j \), it should, according to the failure strategy, receive the block \( F_j \) after the failure of \( S_k \).
List of Figures

Figure 3.1  Failure of a Node $S_i$ in a Data Configuration  26
Figure 3.2  Neighbour Data Replication Strategy  30
Figure 3.3  Scattered Data Replication Strategy  33
Figure 4.1  State Transition  38
Figure 4.2  Failure Mechanism  40
Figure 4.3  Recovery Mechanism  40
PM-1 3½" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

1.0  1.1  1.25

1.25 1.4  1.6

PRECISIONSM RESOLUTION TARGETS
2. \( S_k \rightarrow S_a \rightarrow S_b \):

In this case we will consider two subcases, depending on whether \( S_a \) had \( F_{j-1} \) before the failure of \( S_k \) or not.

2.1 \( F_{j-1} \in S_a \) before the failure of \( S_k \)

If \( S_a \) had \( F_{j-1} \) before the failure of the node \( S_k \), then \( S_a \) must also have had \( F_j \) before the failure of \( S_k \). Suppose \( S_a \) did not have \( F_j \) before the failure. Since we know it has it after, it must have received it from \( S_k \).

In order for \( S_a \) to receive \( F_j \) from \( S_k \), all the nodes between \( S_k \) and \( S_a \) must have had a copy of \( F_j \) (since the failure strategy sends the block \( F_j \) to the last node following \( S_k \) that does not have it). In particular, let \( S_a' \) be the live node immediately preceding \( S_a \) before the failure, obviously \( S_a' \) has a copy of the block \( F_j \). If before the failure of \( S_k \), \( S_a' \) had a copy of \( F_j \) and \( S_a \) had a copy of \( F_{j-1} \), for the theorem to hold true, \( S_a \) must have a copy of \( F_j \). Thus \( S_a \) has a copy of \( F_j \).

We know that \( S_b \) does not have a copy of \( F_j \). \( S_b \) could not have had a copy of \( F_{j-1} \) either, otherwise this would violate the theorem before the failure. Thus, \( F_{j-1} \) must have been given to \( S_b \) by \( S_k \).

We know that, before the failure of \( S_k \), \( F_{j-1} \) is in \( K \) consecutive locations ending at \( S_a \) and that \( S_k \) has a copy of \( F_{j-1} \). We also know that \( F_j \) is in \( K \) consecutive locations ending at \( S_a \). Therefore \( S_k \) must also have a copy of \( F_j \). We also know that \( F_{j-1} \) is sent to \( S_b \) (the first live node following \( S_k \) in ordering not holding a copy of \( F_{j-1} \)). For the same reason, we can conclude that also \( F_j \) is sent to \( S_b \).
2.1 $F_{j-1} \not\in S_a$ before the failure of $S_k$

This means that $F_{j-1}$ was sent to $S_a$ after the failure of $S_k$. But according to the
failure strategy, if $S_a$ gets $F_{j-1}$, then $S_b$ cannot get it, which contradicts our
assumption. Thus, this case cannot occur. This concludes the proof of part (ii) of the
theorem that $F_j \not\in S_b$.[]

Theorem 4.11. Given a system in a $K$-tolerant state, upon the failure of $q \leq K$ nodes,
the execution of the local failure strategy at all remaining live nodes of the network
will restore the system to a $K$-tolerant state; furthermore the distribution is ordered.

Proof: Consider the case $q = 1$. Let $S_k$ fail and let $F_h$ be a block of data stored at $S_k$.
The proof that the resulting distribution is $K$-tolerant is the same as the proof of
theorem 4.1. We will now show that the distribution remains ordered.

By definition of $K$-tolerant ordered distribution, $F_h$ is stored at $K+1$ consecutive
nodes. The failure of $S_k$ causes the leader node of $F_h$ to send $F_h$ to the first live node
following $S_k$ in the ordering and not containing an instance of $F_h$. This results in $F_h$
again being stored at $K+1$ consecutive live nodes, which satisfies part (a) of the
definition of data ordering.

Let $S'_k$ be the first live node that follows $S_k$ holding a copy of $F_h$. By
contradiction, assume that after the failure of $S_k$, the blocks in $S'_k$ are not consecutive.
The blocks were consecutive before the failure (by definition of data ordering). This
means that the failure of $S_k$ created a gap in $S'_k$. A gap means that $S'_k$ holds a copy of
$F_j$ but not $F_{j-1}$ nor $F_{j+1}$.

For $S'_k$ to receive $F_j$, this means that $S'_k$ is the first node following $S_k$ in ordering
not having a copy of $F_j$. It also means that all the nodes between $S_k$ and $S'_k$ have a
copy of $F_j$. By Theorem 4.10 all these nodes must also have a copy of $F_{j+1}$. Thus, the
failure strategy will send a copy of $F_{j+1}$ to $S'_k$. Thus, a gap cannot occur in $S'_k$. Therefore the blocks in $S'_k$ must be consecutive. The generalization to $q > 1$ follows in the same lines.[]

**Theorem 4.12.** The above failure strategy is communication optimal.

The proof is the same as the proof of Theorem 4.2.[]

### 4.4.3 Recovery Strategy

Since in this model we use the same data structure as in the previous ones, we can rely on the recovery protocol presented in Section 4.2 of this chapter to perform a perfect undo of the last failure operation.

In addition, in this model, the failure of a node leads to an ordered distribution (due to the failure strategy that keeps the distribution ordered). Thus, the recovery operation (which is a perfect undo of the last failure operation) will result in a ordered distribution.

### 4.4.4 Summary and Concluding Remarks

This section dealt with continuously K-tolerant ordered distributions. We presented a mechanism $M$ consisting of the UOND Replication Strategy (based on the replication technique presented in Chapter 3), a Failure Protocol and a Recovery Protocol that
keep the distribution in a continuously K-tolerant ordered state after the failure of \( q \leq K \) or the recovery of \( p \geq 1 \) nodes in the system, respectively.

**Theorem 4.13.** The mechanism \( M \) (UOND, Failure, Recovery) consisting of the UOND Replication Strategy, the Failure and Recovery Protocols discussed in Section 4.4 is continuously \( K \)-tolerant and communication optimal. Furthermore, the application of the mechanism \( M \) to an ordered distribution \( X \) preserves the ordering of \( X \).

**Proof:** From Theorems 4.4, 4.5, 4.11 and 4.12.[1]

If it is required to keep the distribution balanced and ordered, then starting with an ordered and balanced distribution \( X \), the failure and recovery protocols must transmit the data in such a way that the resulting distribution is ordered and balanced, which is the topic of the next section.
4.5 Balanced Distribution, with Ordered Data Set F

Consider a network S with n nodes, a set of data F and a partition \(< F_1, F_2, \ldots, F_n >\) of F over S, where \(F_i\) denotes the block stored at \(S_i\). We would like to maintain a continuously K-tolerant system with the following properties:

a. There is an ordering requirement on the data set F stored in the network.

b. There is a requirement to keep the distribution \(X\) balanced.

In this section we define a mechanism \(M\) consisting of the BOND replication strategy, which introduces replication to a balanced and ordered distribution as defined above, resulting in another balanced and ordered distribution. We also define the failure and recovery protocols of this mechanism that maintain a balanced, ordered and continuously K-tolerant distribution after their local execution at every live node.

In this section, we assume that the distribution \(X\) satisfies the data balancing definition defined in Section 4.3 and the data ordering definition defined in Section 4.4.
4.5.1 Balanced, Ordered, Neighbouring Data (BOND) Replication Strategy

Initial K-tolerance is achieved by introducing the BOND replication strategy as follows:

**BOND Replication Strategy:** At every node $S_i$ we replicate the block of the previous K nodes. Thus the data stored at node $S_i$ is: $X_i = F_h \cup F_{(h-1)} \cup \cdots \cup F_{(h-k)}$.

Assuming an original balanced and ordered partition, the resulting distribution is also balanced and ordered.

The degree of redundancy with this strategy is $\delta(X) = K+1$, which is optimal by Theorem 3.1.

4.5.2 Failure Strategy

To maintain a K-tolerant distribution, following the failure of a node, instances of blocks of data (that belonged to the node that failed) are transmitted to live nodes not holding a copy of the block being sent to them. In addition, in order to maintain the distribution ordered and balanced, blocks of data are transmitted to live nodes in such a way that data ordering and balancing are preserved. The entire process is coordinated by a specific node, called the manager, which is chosen according to a predefined scheme e.g., the node with smallest index ($S_n$).

For this case where data ordering and balancing are important, we will develop a failure strategy which is centralized rather than decentralized as in the previous
sections. As more properties and restrictions are imposed on the system, the computation complexity and the amount of communication required in order to preserve these properties in the resulting distribution increase substantially. After we present the centralized failure strategy, we will discuss the difficulties that arise in defining a decentralized solution to this problem.

Upon the failure of a node $S_k$ in the network, every live node $S_j$ in the network sends to the manager the list of indices of all the data blocks it hold. For every data block $F_h$ a leader node is selected (to be the node with smallest index $k$ in $ML(F_h)$). If $S_j$ is the leader node of $F_h$ then $S_j$ sends to the manager also $ML(F_h)$ and $Fl(F_h)$.

With all the information provided, by executing a sequential ordering and balancing algorithm, the manager determines the data blocks to be allocated at every live node in order to keep the distribution balanced and ordered. Among all possible solutions, the manager chooses the one which minimizes the number of block transmissions. The node membership list and the array of failed indices of every block are also updated.

After this local computation, the manager sends to the block leader node of $F_h$ a message containing the data block index $F_h$, the new $ML(F_h)$, and the new $Fl(F_h)$. Non leader nodes receive a message containing the block indices of the blocks they should discard.

Every leader node sends the data block $F_h$ to its new members, and sends $ML(F_h)$ and $Fl(F_h)$ to all its members.
Theorem 4.14. In the worst case, the total amount of communication required with the above centralized solution is:

- $2(n-q) + (K-1)$ b messages, and
- $(K-1)$ b block transmissions,

where $n$ is the size of the network, $q$ is the number of failed nodes and $b$ is the number of distinct blocks held by the failed nodes.

Proof: Every live node $S_j$ sends to the manager the list of indices of the data blocks it holds and the ML and FI lists of the blocks for which $S_j$ is a leader. After determining the new data block redistribution, the manager sends to every live node the list of indices of its new data blocks, and the ML and FI lists of the blocks for which $S_j$ is a leader. Non leader nodes receive a message from the manager informing them which blocks to discard. This amounts to $2(n-q)$ messages of the form $<\text{indices}(F_h),\text{ML}(F_h),\text{FI}(F_h));$ where indices $(F_h)$ is the list of the data block indices held by a given live node.

The leader node of block $F_h$ sends to all its new members the data block $F_h$ and to all its members $\text{ML}(F_h)$ and $\text{FI}(F_h)$. In the worst case the block $F_h$ is transmitted to $(K-1)$ nodes while the $\text{ML}(F_h)$ and $\text{FI}(F_h)$ lists are transmitted to $(K-1)$ nodes. Thus, the total amount of transmissions is $(K-1)$ b blocks and messages of the form $<\text{ML}(F_h),\text{FI}(F_h)>$. The number of block transmissions is the optimal one determined by the centralized algorithm executed by the manager. []

A decentralized solution to this problem is offered by the following approach consisting of two steps:

1. use the failure strategy for the unbalanced ordered case of Section 4.4.
2. Balance the distribution while preserving the ordering of the data.
Step 1 is performed exactly as described in Section 4.4 of this chapter. After the execution of step 1, the distribution is ordered but not balanced. In step 2, a manager is selected to coordinate the balancing operation. Every live node sends to the manager the list of the indices of the data blocks it holds. After collecting this list from all the live nodes, the manager executes the function IsBalanced() to find out whether the distribution is balanced. If the distribution is balanced, then no further processing is required. Otherwise, the manager sends a message to the node with the largest size, $S_i$, informing it to transmit a candidate block, $F_h$, to the first node following $S_i$ in ordering not holding a copy of $F_h$. A candidate block $F_h$ is a block that belongs to the live node $S_i$ (with maximum size) and its occurrence in $S_i$ is the first or last occurrence within the K+1 consecutive nodes it belongs to.

We now discuss some points of this approach in more detail. The function IsBalanced(), returns TRUE if the current distribution is balanced, FALSE otherwise. From the data block index lists (of all the live nodes), the function computes $UB(X)$, the unbalance value of the current distribution. Then the function tries to compute a new value $UB'$ such that $UB' < UB$ (if such a value exist).

If the live node $S_i$ has no candidate block $F_h$, then $S_i$ can not transmit any of its blocks to another node without violating property one of the data ordering definition. A possible way to solve this problem is to interchange the index of node $S_i$ with the index of another node $S_{i'}$ in such a way that $F_h$ becomes a candidate block in the new node $S_{i'}$. This intermediate step can be repeated as long as $S_{i'}$ is the node with maximum size and has no candidate block. If $F_h$ becomes a candidate block in $S_{i'}$ then it can be shipped to another node (as determined by the balancing process). If $S_{i'}$
ceases to be the node with maximum size, then a new node is selected and the algorithm continues its execution.

Note that, if index node switching is not permitted, this problem cannot be solved with a decentralized solution. If index switching is allowed, then the algorithm can proceed but there are no guarantees that after the execution of the balancing step the balance of the new distribution $X'$ is better than the balance $X$. It is possible that in some cases the balancing algorithm never provides an improvement. This can happen when the function IsBalanced returns false, and after executing the balancing process, the balance of the distribution remains unchanged, e.g., a node with size $m$ sends a candidate node to a node with size $m-1$.

### 4.5.3 Recovery Strategy

Although in this section we used a centralized approach to define the failure strategy, the data structures used are similar to the ones used in the previous sections and their contents at the end of the execution of the failure strategy reflects the state of the current distribution. Defining the failure strategy in such a way, allows us to use the distributed recovery strategy defined in Section 4.2 that performs a perfect undo of the last failure operation also with this model.

In addition, the failure of a node leads to an ordered and balanced distribution (due to the failure strategy that keeps the distribution ordered and balanced). Thus, the recovery operation (which is a perfect undo of the last failure operation) will result in a ordered and balanced distribution. Since the failure strategy defined is block
communication optimal, the recovery strategy which is a perfect undo of the failure strategy is also communication optimal.

An alternative to this approach is to appoint a manager node to always execute the failure strategy after the failure of any live node and keep a history of all the block movement. In such a situation, upon the recovery of a node, the manager determines the new allocation of the data blocks. This approach is also communication optimal. The drawback of this solution is that it relies on the manager to always remain alive and handle the failure and recovery operations. Should the manager fail, the history status of the network is lost. Since in our model definition there is no guarantee on the survivability of any given node, this approach is not feasible.

4.5.4 Summary and Concluding Remarks

This section dealt with continuously K-tolerant ordered and balanced distributions. We presented a mechanism $M$ consisting of the BOND Replication Strategy (based on the replication technique presented in Chapter 3), a Failure Protocol and a Recovery Protocol that keep the distribution in a continuously K-tolerant balanced and ordered state after the failure of $q \leq K$ or the recovery of $p \geq 1$ nodes in the system, respectively.

Theorem 4.15. The mechanism $M$ (BOND, Failure, Recovery) consisting of the BOND Replication Strategy, the Failure and Recovery Protocols discussed in Section 4.5 is continuously K-tolerant and block communication optimal. Furthermore, the
application of the mechanism $M$ to a balanced and ordered distribution $X$ preserves the balancing and the ordering of $X$.

**Proof:** From the failure strategy presented in this section and Theorems 4.4, 4.5, and 4.14.[1]

As we already mentioned in this section, as more properties and restrictions are imposed on the initial distribution, the computation complexity and the amount of communication required to preserve these properties in the resulting distributions increase.
Chapter 5

Applications and Conclusions

In this chapter, we look at an existing distributed system, and discuss how to improve the availability of its data by incorporating one of the Failure Protocols and the Recovery Protocol developed in this thesis. We also summarize the results of the thesis, give some concluding remarks, and discuss the areas of future work.

5.1 Applications

In this section, we look at an existing distributed file system, Eden, and describe how we can integrate one of our mechanisms to this system. The Eden system, discussed in [PNP], has requirements and assumptions which are similar to the ones we used. In particular, the Eden system consists of a number of individual nodes connected through a LAN network; thus, network partitioning cannot occur, and node crashes are detectable.

A replicated directory structure in the Eden system has been designed and implemented based on the method of regeneration [PNP]. The directory system allows for selection of arbitrary objects to be replicated, choice of the number of replicas for each object, and placement of the copies on machines with independent failure mode.
Copies may become inaccessible due to node crashes, but as long as a one copy survives, the replication level is restored by automatically replacing lost copies on other active machines.

Eden is an experimental local area network used as a test bed for research on distributed system implementation and distributed applications. It provides a unique combination of features that are extensions of some found in other systems. It is a distributed operating system that supports objects that are extensible and defined by the user. Its view of objects is similar to the one in the Hydra project [WCC] but extended to the distributed case. Objects can migrate from one node to another, and the binding of a client to a server can take place at every invocation rather than during an initial establishment of connection. The objects may contain data, procedures, and active processes. They are general objects, not just files, and the operations to which they respond when invoked depend on the design of the object. Each object is identified by a capability, a system wide unique identifier plus a set of access rights that define the operations a client can invoke on the object. Invocation is location independent and the Eden objects are mobile. Default modes for placement of objects may be overridden, for example, in load sharing or in replication to place copies on separate machines.

The underlying Kernel at each node protects capabilities, performs garbage collection of unreferenced objects, handles invocation messages between objects, and undertakes the task of locating the invoked object.

For this system, a replicated directory structure based on regeneration has been developed. The main feature of the structure is the regeneration of replicas that are
missing due to crashes in the distributed system, given that at least one copy survives. This restores objects to the specified level of replication and adapts to planned or unplanned configuration changes.

After initial replication, the system (described above) is in a K-tolerant state. Each object is replicated at K+1 different live nodes. The system can tolerate the failure of up to K nodes while keeping full availability of the data.

During an update operation, if the number of replicas of a given object is less than K+1, then the object is regenerated at different live nodes in the network. However, because regeneration of objects occurs only during an update (write) operation, the complete loss of the availability of an object is possible.

Consider an object O which is stored at K+1 locations. Assume that the nodes holding O fail one after the other and no updates occur in the system during all this time. In this situation, even if there was enough time between failures to replace the failed replica, no regeneration would take place and object O will be lost forever.

The mechanisms proposed in this thesis offer a viable alternative to the regeneration strategy. In fact, they can be immediately and directly integrated within the Eden system (since all assumptions on which they rely are met there). Furthermore, they would totally avoid the drawbacks of the Regeneration.

For instance; instead of regenerating objects during an update operation as suggested by the Regeneration technique, objects can be regenerated upon the detection of a node failure as suggested in our Failure Protocols. This would avoid the complete loss of the availability of an object.
5.2 Summary and Conclusions

5.2.1 Summary of Main Contributions

The presence of faults in a distributed K-tolerant file system, raises the issues of data file consistency and data file availability. As mentioned earlier on, the issue of data file consistency is an issue that has been studied extensively and is not within the scope of this paper. Our interest in this thesis lied in the issue of data file availability; more precisely we were interested in the issue of maintaining continuously K-tolerant file systems in the presence of up to K faults in the system.

A continuously K-tolerant system is a system which, starting from a K-tolerant configuration, after the simultaneous failure of up to K nodes or the recovery of an arbitrary number of nodes, reconfigures itself so that it remains K-tolerant.

The main contribution of this thesis is in the design and development of efficient solutions to the problem of continuously guaranteeing the availability of files under different system requirements. Specifically we studied the following cases:

1) Basic case: no restrictions imposed on the system.
2) Balanced case: balanced load requirement.
3) Ordered case: ordered data requirement.
4) Balanced Ordered case: balanced load and ordered data requirements.
For each of the above cases, we developed a mechanism that keeps a K-tolerant system continuously K-tolerant after the occurrence of up to K faults in the system and the recovery of an arbitrary number of nodes. Each mechanism consists of a Replication Strategy, a Failure Protocol and a Recovery Protocol.

We studied the minimum degree of redundancy required to make a system K-tolerant and presented optimal Replication Strategies to achieve K-tolerance with the minimum amount of redundancy.

The Failure Protocols we presented depend on the properties and the restrictions imposed on the system. Thus for each of the four cases, we developed a Failure Protocol that preserved the properties of the initial system and kept the system continuously K-tolerant.

With careful definition and selection of the data structures associated with the blocks of data stored at the nodes, we developed a general Recovery Protocol independent of the properties imposed on the system; thus, this protocol was used for all the four cases discussed. Furthermore, it can be applied to any other mechanism that uses our proposed data structures.

The Recovery Protocol performs a perfect undo of the last execution of the Failure Protocol used by the system.

We studied the complexity of the proposed solutions and proved their correctness. All the protocols we developed are fully distributed and highly efficient.
We also discussed how our protocols can be used in the Eden system as an effective alternative to the Regeneration technique.

5.2.2 Open Problems and Areas of Future Work

The solution we proposed for the case where there are balanced and ordered data requirement, is distributed but centralized. Developing a decentralized and (block) communication optimal solution remains an open problem.

In this thesis, we assumed only read access to our data. In the case of read and write operations, where the issue of data consistency is involved, our solutions can be integrated with the consistency methods.

In our study, we limited ourselves to the LAN model. A future direction for research is to extend and generalize the results to a graph topology, where network partitions can occur.

We also assumed an underlying reliable mechanism for node failure detection. Removing this assumption makes the problem more difficult, but one that should be examined in future work.
Bibliography


END
31-05-95
FIN