

Contents

| | | |
|----------|--|----------|
| 1 | Introduction | 1 |
| | <i>Kenneth P. Birman and Robbert van Renesse</i> | |
| 1.1 | Categorizing computing systems | 2 |
| 1.2 | A philosophical evolution | 5 |
| 1.3 | Goals of this book | 6 |
| 1.4 | A brief history of distributed computing | 7 |
| 1.4.1 | Network computing systems | 7 |
| 1.4.2 | Remote procedure calls | 8 |
| 1.4.3 | Distributed computing systems | 9 |
| 1.4.4 | Controversy | 11 |
| 1.5 | Isis background and development | 13 |
| 1.6 | Isis Applications | 15 |
| 1.7 | Underlying Theory | 16 |
| 1.8 | The Horus Architecture | 20 |
| 1.9 | A Security Architecture For Horus | 21 |
| 1.10 | Future Directions | 22 |
| 1.11 | Outline of this Book | 22 |
| 1.12 | Acknowledgements | 22 |

| | | |
|----------|---|-----------|
| I | Fundamentals | 24 |
| 2 | The Process Group Approach to Reliable Distributed Computing | 27 |
| | <i>Kenneth P. Birman</i> | |
| 2.1 | Introduction | 27 |
| 2.2 | Process groups | 31 |
| 2.3 | Building conventional distributed services | 32 |
| 2.3.1 | Conventional message passing technologies | 32 |
| 2.3.2 | Failure model | 34 |
| 2.3.3 | Building groups over conventional technologies | 35 |
| 2.4 | Virtual synchrony | 41 |
| 2.4.1 | Summary of benefits due to virtual synchrony | 46 |
| 2.5 | The Isis Toolkit | 47 |
| 2.5.1 | Styles of groups | 47 |
| 2.5.2 | The toolkit interface | 48 |
| 2.6 | Who uses Isis, and how? | 50 |
| 2.6.1 | Brokerage | 50 |
| 2.6.2 | Database replication and database triggers | 53 |
| 2.6.3 | Major Isis-based utilities | 54 |
| 2.6.4 | Other Isis applications | 55 |
| 2.7 | Isis and other distributed computing technologies | 56 |
| 2.8 | Conclusions | 56 |
| 3 | Causal Controversy at Le Mont St.-Michel | 58 |
| | <i>Robbert van Renesse</i> | |
| 3.1 | Introduction | 58 |
| 3.2 | What is causality? | 59 |
| 3.3 | Misconceptions | 60 |
| 3.4 | Examples | 63 |
| 3.4.1 | Files and other objects | 63 |
| 3.4.2 | Global Predicate Evaluation | 63 |
| 3.4.3 | Replicated objects | 65 |
| 3.4.4 | Lazy Release Consistency | 65 |
| 3.4.5 | Consistency in the ORCA Programming Language | 66 |
| 3.4.6 | Teleconferencing, trading, and USENET | 66 |
| 3.5 | Conclusion | 67 |
| 4 | RPC Considered Inadequate | 68 |
| | <i>Kenneth P. Birman and Robbert van Renesse</i> | |
| 4.1 | Introduction | 68 |
| 4.2 | Modern Distributed Systems | 69 |
| 4.3 | RPC and Performance | 70 |
| 4.4 | RPC and Robustness | 72 |

| | | |
|----------|--|------------|
| 4.5 | Group styles of computing | 74 |
| 4.6 | Location transparency: Too much of a good thing? | 74 |
| 4.7 | What is really needed is a new architecture | 76 |
| 4.7.1 | What should we adopt from the RPC model? | 76 |
| 4.7.2 | The new architecture | 76 |
| 4.8 | Conclusions | 78 |
| 5 | Exploiting Virtual Synchrony in Distributed Systems | 79 |
| | <i>Kenneth P. Birman and Thomas Joseph</i> | |
| 5.1 | A toolkit for distributed systems | 79 |
| 5.2 | Virtual synchrony | 81 |
| 5.2.1 | Assumptions | 81 |
| 5.2.2 | Subproblems we wish to solve | 81 |
| 5.2.3 | Existing methodologies | 82 |
| 5.2.4 | Virtually synchronous environments | 83 |
| 5.2.5 | Other virtually synchronous tools | 84 |
| 5.3 | Virtually synchronous tools | 85 |
| 5.3.1 | Atomic multicast primitives | 85 |
| 5.3.2 | Process groups and group RPC | 86 |
| 5.3.3 | Cooperating to execute requests | 87 |
| 5.3.4 | Concurrency | 88 |
| 5.3.5 | Semaphores | 88 |
| 5.3.6 | Replicated data | 88 |
| 5.3.7 | Detecting and reacting to failures | 89 |
| 5.3.8 | Recovery and reconfiguration | 89 |
| 5.3.9 | News service | 90 |
| 5.3.10 | Protection | 90 |
| 5.3.11 | Additional tools | 90 |
| 5.4 | Miscellaneous system-level facilities | 90 |
| 5.4.1 | Run time facilities | 91 |
| 5.4.2 | Machine Independence and scaling | 92 |
| 5.5 | A toolkit application | 92 |
| 5.6 | Inside the coordinator-cohort tool | 96 |
| 5.7 | Performance | 97 |
| 5.8 | Status | 100 |
| 6 | Virtual Synchrony Model | 101 |
| | <i>Kenneth P. Birman</i> | |
| 6.1 | Basic elements of the model | 101 |
| 6.2 | Basic execution axioms | 102 |
| 6.3 | Virtually synchronous execution | 103 |
| 6.4 | Failure model | 104 |

| | | |
|-----------|--|------------|
| II | Redesign | 107 |
| 7 | Design Alternatives for Process Group Membership and Multicast | 109 |
| | <i>Kenneth P. Birman, Robert Cooper, and Barry Gleeson</i> | |
| 7.1 | Introduction | 109 |
| 7.2 | Process groups | 112 |
| 7.2.1 | Group membership | 112 |
| 7.2.2 | Why provide support for process groups? | 112 |
| 7.2.3 | Which processes should be allowed to send to a group? | 113 |
| 7.2.4 | Should group multicast provide “strong guarantees”? | 114 |
| 7.2.5 | Why not layer group communication over RPC? | 115 |
| 7.2.6 | Does multicast belong inside the operating system? | 116 |
| 7.3 | Detailed design choices for a single group | 117 |
| 7.3.1 | Group structure: Members and clients | 117 |
| 7.3.2 | Atomicity | 118 |
| 7.3.3 | Causal and total multicast orderings | 120 |
| 7.4 | Ordering properties that span group boundaries | 123 |
| 7.4.1 | Who uses overlapping groups? | 123 |
| 7.4.2 | Should causality be preserved between groups? | 123 |
| 7.4.3 | Should causality <i>always</i> be preserved between groups? | 125 |
| 7.4.4 | How visible should causality information be? | 126 |
| 7.4.5 | Should abcast be ordered between groups? | 127 |
| 7.5 | Extended example: Causal process pairs | 128 |
| 7.6 | An implementation | 131 |
| 7.7 | Conclusions | 131 |
| 8 | The Horus System | 133 |
| | <i>Robbert van Renesse, Kenneth P. Birman, Robert Cooper, Brad Glade, and Patrick Stephenson</i> | |
| 8.1 | Introduction | 133 |
| 8.2 | Design | 134 |
| 8.3 | The Multicast Transport Service | 137 |
| 8.4 | Multi-threading and Ordering | 139 |
| 8.5 | The Vsync Subsystem | 140 |
| 8.6 | Security layer and Real-time Layer | 141 |
| 8.7 | Customizing Horus to a Port Interface | 143 |
| 8.7.1 | Ports | 143 |
| 8.7.2 | Integrating Horus into a Port Interface | 144 |
| 8.8 | The Isis Toolkit | 145 |
| 8.9 | Initial Performance Results and Conclusion | 145 |
| 9 | Integrating Security in a Group-Oriented Distributed System | 148 |
| | <i>Michael Reiter, Kenneth P. Birman, and Li Gong</i> | |

| | | |
|------------|--|------------|
| 9.1 | Introduction | 148 |
| 9.2 | The Isis abstractions | 150 |
| 9.3 | The system model | 151 |
| 9.4 | Protecting the Isis abstractions | 152 |
| 9.4.1 | Multicast authentication | 154 |
| 9.4.2 | Joining groups | 156 |
| 9.4.3 | Causal multicast | 158 |
| 9.5 | Delegation and access control | 162 |
| 9.6 | Conclusion and future work | 165 |
| 10 | High Availability in a Real-Time System | 167 |
| | <i>Carlos Almeida, Brad Glade, Keith Marzullo, and Robbert van Renesse</i> | |
| 10.1 | Introduction | 167 |
| 10.2 | System structure | 168 |
| 10.2.1 | System interfaces | 169 |
| 10.2.2 | Example application | 170 |
| 10.3 | Proposed architecture and implementation | 171 |
| 10.4 | Current Status | 172 |
| III | Protocols | 173 |
| 11 | Reliable Communication in the Presence of Failures | 176 |
| | <i>Kenneth P. Birman and Thomas Joseph</i> | |
| 11.1 | Introduction | 176 |
| 11.2 | System characteristics | 178 |
| 11.3 | Fault tolerant process groups | 178 |
| 11.3.1 | Using the group broadcast primitive to maintain process group views | 180 |
| 11.3.2 | The atomic broadcast primitive | 180 |
| 11.3.3 | The causal broadcast primitive | 181 |
| 11.3.4 | Additional broadcast primitives | 183 |
| 11.3.5 | Synchronous and asynchronous uses of the primitives | 183 |
| 11.3.6 | Group addressing | 184 |
| 11.3.7 | Checkpointing the group state and transferring state | 185 |
| 11.3.8 | Example | 186 |
| 11.4 | Implementation | 186 |
| 11.4.1 | The inter-site layer | 187 |
| 11.4.2 | Site view management | 188 |
| 11.4.3 | The protocols | 190 |
| 11.4.4 | An associative store and distributed garbage collection facility | 197 |
| 11.4.5 | CBCAST flush implementation | 198 |
| 11.4.6 | Wide area networks | 199 |
| 11.4.7 | Liveness | 199 |

| | | |
|-----------|---|------------|
| 11.5 | Limitations | 199 |
| 11.6 | Conclusions | 200 |
| 12 | Lightweight Causal and Atomic Group Multicast | 201 |
| | <i>Kenneth P. Birman, André Schiper, and Pat Stephenson</i> | |
| 12.1 | Introduction | 201 |
| 12.1.1 | The Isis Toolkit | 201 |
| 12.2 | Execution model | 202 |
| 12.2.1 | Basic system model | 203 |
| 12.2.2 | Virtual synchrony requirements for multicast protocols | 204 |
| 12.2.3 | Vector time | 205 |
| 12.3 | The CBCAST and ABCAST protocol | 206 |
| 12.3.1 | CBCAST protocol | 206 |
| 12.3.2 | Causal ABCAST protocol | 209 |
| 12.3.3 | Multicast stability | 210 |
| 12.3.4 | VT compression | 211 |
| 12.3.5 | Delivery atomicity and group membership changes | 211 |
| 12.4 | Extensions to the basic protocol | 215 |
| 12.4.1 | Extension of CBCAST to multiple groups | 216 |
| 12.4.2 | Multiple-group ABCAST | 217 |
| 12.4.3 | Extended VT compression | 217 |
| 12.4.4 | Garbage collection of vector timestamps | 219 |
| 12.4.5 | Atomicity and group membership changes | 219 |
| 12.4.6 | Use of communication structure | 222 |
| 12.4.7 | Extensions to dynamic communication structures | 225 |
| 12.5 | Applying the protocols to Isis | 228 |
| 12.5.1 | Optimizations for Client/Server Groups | 228 |
| 12.5.2 | Point-to-point messages | 229 |
| 12.5.3 | System interface issues | 230 |
| 12.5.4 | Group view management | 230 |
| 12.6 | Performance | 231 |
| 12.7 | Conclusions | 235 |
| 13 | Consistent Process Membership in Asynchronous Environments | 237 |
| | <i>Aletta Ricciardi and Kenneth P. Birman</i> | |
| 13.1 | The Need for a Membership Service | 237 |
| 13.2 | System Model | 240 |
| 13.2.1 | The Formal Language | 240 |
| 13.2.2 | Failure Suspicions | 241 |
| 13.2.3 | Process Behavior and Local Observations | 241 |
| 13.3 | Strong GMP Specification | 243 |
| 13.4 | Solving Strong GMP | 244 |
| 13.4.1 | The Basic Algorithm | 245 |
| 13.4.2 | The Full Algorithm | 246 |

| | | |
|-----------|---|------------|
| 13.4.3 | The s-GMP Algorithm | 250 |
| 13.5 | Comparing Strong GMP with Weaker Variants | 254 |
| 13.5.1 | Characteristic Membership Properties | 254 |
| 13.5.2 | Three Group Membership Problems | 256 |
| 13.5.3 | Message Complexity Comparison | 258 |
| 13.5.4 | Using Group Membership Properties | 259 |
| 13.6 | Conclusion | 262 |
| 14 | Efficient Broadcast Primitives in Asynchronous Distributed Systems | 263 |
| | <i>Frank Schmuck</i> | |
| 14.1 | Specifications | 263 |
| 14.1.1 | Formal Events and Histories | 264 |
| 14.1.2 | Example: Token Passing | 265 |
| 14.2 | Executions | 267 |
| 14.2.1 | ABCAST Implementation | 267 |
| 14.2.2 | Execution Histories | 268 |
| 14.2.3 | Implementation Correctness | 270 |
| 14.3 | Asynchronous Implementations | 271 |
| 14.3.1 | CBCAST Implementation | 272 |
| 14.3.2 | Correctness and Existence | 275 |
| 14.3.3 | Weaker implementations | 277 |
| 14.4 | Commutative Specifications | 278 |
| 14.4.1 | Undecidability | 278 |
| 14.4.2 | Commutativity and Ordering Constraints | 279 |
| 14.5 | Conclusions | 282 |
| 15 | Light-Weight Process Groups in the Isis System | 284 |
| | <i>Bradford B. Glade, Kenneth P. Birman, Robert C. B. Cooper, and Robbert van Renesse</i> | |
| 15.1 | Introduction | 284 |
| 15.2 | Trends in the use of Process Groups | 285 |
| 15.2.1 | The Deceit File System | 286 |
| 15.2.2 | The Isis Transaction Tool | 286 |
| 15.2.3 | Meta | 287 |
| 15.3 | Analysis of Performance Problems | 288 |
| 15.3.1 | Failure detection | 289 |
| 15.3.2 | Overlapping Groups | 289 |
| 15.3.3 | Named groups | 290 |
| 15.4 | Overall Design | 290 |
| 15.5 | Design Issues | 292 |
| 15.5.1 | Mapping LWGs to core groups | 292 |
| 15.5.2 | Added Functionality | 294 |
| 15.5.3 | Large numbers of process groups | 295 |
| 15.6 | Interface | 295 |

| | | |
|-----------|---|------------|
| 15.7 | Initial Performance Results | 295 |
| 15.8 | Related Work | 296 |
| 15.9 | Conclusion | 297 |
| IV | Tools and Applications | 298 |
| 16 | Using the Isis Resource Manager for Distributed, Fault-Tolerant Computing | 300 |
| | <i>Timothy A. Clark and Kenneth P. Birman</i> | |
| 16.1 | The Growing Importance of Networks | 301 |
| 16.2 | Robust Applications | 301 |
| 16.3 | Architecture of the Resource Manager | 302 |
| 16.3.1 | The Resource Manager Server | 302 |
| 16.3.2 | The Resource Manager Stub | 305 |
| 16.3.3 | The Resource Manager User-Interface Programs | 306 |
| 16.4 | Conclusions | 307 |
| 17 | The Design and Implementation of Meta | 309 |
| | <i>Mark Wood and Keith Marzullo</i> | |
| 17.1 | Introduction | 309 |
| 17.1.1 | The Reactive System Architecture | 310 |
| 17.1.2 | Why Reactive Systems? | 311 |
| 17.1.3 | Why Distributed Control? | 311 |
| 17.1.4 | Characteristics of a Good Solution | 311 |
| 17.2 | The Meta Toolkit | 312 |
| 17.2.1 | The Meta Stub Library | 312 |
| 17.2.2 | The Meta Client Interface | 316 |
| 17.2.3 | The Guarded Command Language | 318 |
| 17.3 | The Implementation of the Meta Toolkit | 319 |
| 17.3.1 | Alternatives to Isis | 320 |
| 17.3.2 | The Meta Implementation | 320 |
| 17.4 | Conclusion | 327 |
| 18 | Paralex: An Environment for Parallel Programming in Distributed Systems | 328 |
| | <i>Özalp Babaoğlu, Lorenzo Alvisi, Alessandro Amoroso, Renzo Davoli, and Luigi Alberto Giachini</i> | |
| 18.1 | Introduction | 328 |
| 18.2 | The Paralex Programming Paradigm | 329 |
| 18.2.1 | Computation Nodes | 330 |
| 18.2.2 | Filter Nodes | 331 |
| 18.2.3 | Subgraph Nodes | 331 |
| 18.2.4 | Cycle Nodes | 331 |
| 18.3 | Overview of Paralex | 331 |
| 18.3.1 | Site Definition | 332 |

| | | |
|-----------|---|------------|
| 18.3.2 | The Graphics Editor | 333 |
| 18.3.3 | Specifying Program Properties | 334 |
| 18.3.4 | Compiling Paralex Programs | 336 |
| 18.3.5 | Executing Paralex Programs | 336 |
| 18.4 | Fault Tolerance | 337 |
| 18.5 | Dynamic Load Balancing | 339 |
| 18.6 | Performance Results | 339 |
| 18.7 | Related Work | 341 |
| 18.8 | Status and Conclusions | 341 |
| 19 | IMIS: A Distributed Query and Report Formatting System | 343 |
| | <i>T. Anthony Allen, William Sheppard, and Steve Condon</i> | |
| 19.1 | System Overview | 343 |
| 19.2 | Why Not Relational | 344 |
| 19.3 | Transcending Relational Technology | 346 |
| 19.4 | The Report Formatter | 346 |
| 19.4.1 | Semantics and Data Structure | 347 |
| 19.4.2 | The Tawk Calculation Language | 348 |
| 19.4.3 | Data Updating | 349 |
| 19.5 | IMIS User Interface | 350 |
| 19.5.1 | Custom Widgets | 350 |
| 19.5.2 | Modified Widgets | 350 |
| 19.5.3 | Building a Form | 351 |
| 19.5.4 | Inspectors | 352 |
| 19.5.5 | Print Quality | 353 |
| 19.6 | IMIS and Distributed Processing | 353 |
| 19.6.1 | The Isis Toolkit | 353 |
| 19.6.2 | Multiple Client/Multiple Server Model | 354 |
| 19.6.3 | Server Groups | 355 |
| 19.6.4 | Client Groups | 355 |
| 19.6.5 | Monitoring Process Groups | 356 |
| 19.6.6 | Performance | 356 |
| 19.7 | Summary | 358 |
| 20 | Distributed Programming with Asynchronous Ordered Channels in Distributed ML | 359 |
| | <i>Robert Cooper and Clifford Krumvieda</i> | |
| 20.1 | Introduction | 359 |
| 20.2 | Distributed ML | 360 |
| 20.3 | Multicasts | 361 |
| 20.4 | Multicast Ordering | 363 |
| 20.4.1 | Causal Ordering | 363 |
| 20.4.2 | Causal Completeness | 363 |
| 20.4.3 | Ordering Declarations | 364 |

| | | |
|-----------|---|------------|
| 20.5 | Message Stability | 365 |
| 20.6 | Ordered Events | 365 |
| 20.7 | Example: Replicated processing | 366 |
| 20.8 | Conclusion | 369 |
| 21 | The Isis Project: Real experience with a fault tolerant programming system | 370 |
| | <i>Kenneth P. Birman and Robert Cooper</i> | |
| 21.1 | What has been successful in Isis? | 370 |
| 21.2 | What lessons did we learn? | 371 |
| 21.3 | What did we learn from implementing Isis? | 372 |
| 21.4 | Who uses Isis? | 373 |
| 21.5 | Conclusions | 375 |
| | Bibliography | 376 |
| | Index | 393 |