Preface

The growing popularity of the Internet along with the availability of powerful computers, standardised software, and high-speed networks as low-cost commodity components is helping to change the way we undertake computing, communication, and business. High Performance Computing (HPC) systems developed using standard off-the-shelf hardware and software components have been employed in several application areas including science, engineering, and commerce. The benefits of these HPC technologies are now starting to reach mass markets due to their usage in Internet and Web-based applications such as digital libraries, virtual laboratories, video on demand, e-commerce, web services, collaborative systems, and so on.

Applications in the Information Age are data driven in nature and demand highly reliable computing systems that are capable of delivering balanced computing, storage, I/O, and network communication performance. We have seen a dramatic increase in microprocessor performance in the last few years and computer system performance is doubling every 18-24 months (*Moore's Law*). This is likely to continue for several years. Unfortunately, the rate of improvements in storage access speed has not kept pace with processor performance. Although disk storage densities have increased by 60-80% per year, overall improvement in disk access times, which rely upon advances in mechanical systems, has been less than 10% per year.

Amdahl's law implies that the speedup obtained from computers is limited by the slowest system component. As a result, disk system performance is becoming a dominant factor and bottleneck in overall system behavior. It is necessary to improve I/O performance so that it balances with processor performance. Hence, storage capacity and access speed have become critical issues to be considered in the design of computer systems.

To tackle both the capacity and performance problems, disk arrays and parallel I/O issues are being widely investigated. The basic objective of integrating multiple disks together is to create highly reliable, mass storage systems. To improve I/O performance, storage units in disk arrays are accessed in parallel. Although, using I/O devices in parallel increases the capacity and performance of storage systems, it does not reduce the probability of disk failure. Hence, integrated mechanisms for improving storage reliability, capacity, and performance have been proposed.

The innovative "*Redundant Arrays of Inexpensive Disks (RAID)*" idea, proposed in 1988 by David Patterson, Garth Gibson, and Randy Katz from the University of California, Berkeley, has heralded a new era in large-scale storage systems. The RAID systems use distribution algorithms and different fault tolerant mechanisms to provide improved capacity, availability and performance. The RAID concept has helped in overcoming bottlenecks due to the disparity in processor, memory, storage, and I/O performance. In just a few years, RAID technology has evolved from concept to reality.

There are several techniques proposed and used in the design and development of high performance storage systems. These include parallel I/O, caching, prefetching, smart and serverless file systems, adaptive techniques driven by storage access patterns, and rich I/O interfaces. Caching and prefetching mechanisms greatly enhance the performance of storage subsystems due to improved cache hit-ratio and reduced disk I/O access time. File systems have been modified to take advantage of the possibility of using several devices in parallel. Furthermore, these mechanisms have been enhanced for taking advantage of resources located in different nodes in a cluster or storage network. A number of high performance storage subsystems such as Storage Area Networks (SAN) and Network-Attached Storage (NAS) provide efficient I/O interfaces with rich semantics. These topics are dealt in greater depth throughout the book.

Organization

The increasing demand for high performance, large-scale storage systems and the lack of a readily accessible single source of information on this area has motivated the creation of this book. The literature in this area is usually scattered among many journals, conferences, and workshops devoted to different areas, including operating systems, parallel and distributed systems, algorithms, multimedia, and databases. The first objective of this book is to provide state-of-the-art information on high-performance mass storage systems. The second objective is to identify the emerging technologies and future trends in high performance storage systems and parallel I/O areas, including new parallel I/O programming paradigms, standards

for future storage area network and network attached storage systems, such as InfiniBand, XML, and innovative parallel I/O applications.

This book is organised as a collection of articles authored by leading researchers and experts in the field. These chapters are drawn from publications in leading conferences and journals. Some of these articles have also been updated to reflect the recent developments in the field. The book covers emerging technologies and future trends in storage, networks, and parallel I/O. This book contains 45 chapters that are grouped into the following nine parts:

- Introduction to Redundant Disk Array Architecture
- Advanced Disk Array Architectures
- Fault Tolerance Issues in Disk Arrays
- Caching and Prefetching
- Parallel File Systems
- Parallel I/O Systems
- Parallel I/O Programming Paradigms
- Parallel I/O Applications and Environments
- Emerging Technologies and Future Trends

The chapters cover the subject with greater emphasis on design, architecture, algorithms, and experimental results. A brief discussion on contents of each part is presented below.

PART I: Introduction to Redundant Disk Array Architecture

Most of the advances in the field of parallel I/O were triggered by Redundant Arrays of Inexpensive Disks (RAID) technology proposed by David Patterson, Garth Gibson, and Randy Katz in the late eighties. A RAID consists of several disks connected to a single controller offering a single I/O space. This configuration enhances the bandwidth of the I/O system depending on the number of disks operating in parallel and at the same time hides latency. The chapters included in this part introduce the fundamentals of disk arrays and their design and development along with their performance benefits and limitations.

Chapter 1 discusses the initial ideas behind the RAID architectures. In 1998, this paper was rewarded as the most influential paper from the SIGMOD proceedings 10 years earlier. RAID architecture is based on the idea that connecting several disks in parallel can improve the bandwidth of the storage system. For the first time, this paper introduces five levels of RAIDs, giving their relative cost/performance, and compares RAIDs to an IBM 3380, and a Fujitsu Super Eagle.

Chapter 2 reviews the state-of-the-art in disk devices and I/O controllers at the beginning of nineties when parallel I/O research exploration was still in its infancy.. We also discuss techniques that are employed to achieve high-performance I/O based on RAIDs.

Chapter 3 covers the implications of parity placement approaches in RAID systems. In this chapter, eight different policies to place the parity in a RAID 5 are studied, and each of these policies delivers different levels of performance. This effect is of special interest for large size data requests. As a conclusion of this paper, the authors propose a set of properties that are generally desired for parity placement.

Chapter 4 compares the performance of RAID-5 to that of log-structured arrays (LSA) on transaction-processing workloads. LSA borrows heavily from the log-structured file system (LFS) approach, but it is implemented in an outboard disk controller. The LSA technique examined in this chapter combines LFS, RAID, compression, and Non-Volatile cache. This chapter looks at the sensitivity of LSA performance to the amount of free space on the physical disks and also to the compression ratio achieved. Authors also evaluate a RAID-5 design that supports compression in cache.

Part II: Advanced Disk Array Architectures

In the recent past, RAIDs systems have evolved considerably. The chapters included in this part present many influential techniques proposed to improve the performance of storage subsystems. They range from the solution to the small-write problem to the distribution of RAID components over a network. Hierarchies of I/O devices and new scheduling algorithms are integrated into the disk array architectures to improve their performance and functionality.

Chapter 5 presents a solution to the small-write problem of RAID level 5. In this type of disk array, writing small amounts of data implies a combination of read-and-write operation instead of a single write operation. To solve this

problem, the authors propose a novel solution called *Parity Logging* that uses journaling techniques to substantially reduce the cost of small writes.

Chapter 6 extends the RAID concept to a distributed computing system and builds *Redundant Array of Distributed Disks* (RADD). RADDs are shown to support redundant copies of data across a computer network at the same space cost as RAIDs do for local data. Furthermore, the authors evaluate several strategies for data replication in order to achieve a good balance between space, cost, performance during normal operation, and performance during failures.

Chapter 7 presents a two-level storage hierarchy implemented inside a single disk-array controller. In the upper-level of this hierarchy, two copies of active data are stored to provide full redundancy and excellent performance. In the lower level, RAID 5 parity protection is used to provide economical storage cost for inactive data, at somewhat lower performance. The movement of data between these two levels is done automatically, simplifying the administration issues.

Chapter 8 proposes a new data structure called *Scalable Distributed Log Structured Array* (LSA*) that generalizes the *Log Structured Array* (LSA) structure. LSA is intended to provide high-availability at a disk-storage server level without the small-write penalty of RAID-schemes, while LSA* is intended for storage area networks. A LSA* file can scale to size orders of magnitude larger than that of an LSA file. Finally, it may allow access even in presence of entirely unavailable storage nodes. This feature enhances the high-availability beyond that of the LSA capabilities.

Chapter 9 explores the choice of sparing methods that impact the performance of RAID level 5 disk arrays. The investigation is performed on database type workloads under the three different operating modes. They are normal mode (no disks have failed), degraded mode (a disk has failed and its data is being reconstructed), and copyback mode (which is needed for distributed sparing and parity sparing when failed disks are replaced with new disks).

Chapter 10 presents and evaluates destage algorithms for RAID 5 architectures assuming that a non-volatile write cache is available. These algorithms aim at offering higher performance, minimal overhead on foreground read operations, optimal disk utilization, and tolerance of burst in the workload without causing write-cache overflow.

Part III: Fault Tolerance Issues in Disk Arrays

In addition to performance, fault tolerance is one of the major design issues in disk arrays and parallel I/O systems. As many disks are combined to build a large-scale storage device, the probability that one or more disks or components will fail is much higher than a single disk storage system. This has prompted many researchers to focus on developing techniques to enhance the system availability even under failure of many disks, reduce the overhead of parity computation, and minimise the overhead of data reconstruction after failures. Some of the key technologies and methodologies addressing storage system availability and fault tolerance are discussed in the chapters included in this part.

Chapter 11 examines alternative data encodings for reliably storing information on disk arrays. Codes are selected to maximize mean time to data loss or minimize disks containing redundant data, but are all constrained to minimize performance penalties associated with updating information or recovering from catastrophic disk failures. In this chapter, the authors show codes that give highly reliable data storage with low redundant data overhead for arrays of 1000 information disks.

Chapter 12 presents a method for tolerating multiple disk failures in disk arrays, instead of only tolerating one failure as traditional arrays do. This method can mask any number of failures, requires a minimal amount of redundant storage space, and spreads reconstruction accesses uniformly over disks in the presence of failures, without needing large layout tables in controller memory. Furthermore, it encourages an efficient hardware implementation, which means it can easily be included in array controllers.

Chapter 13 describes and evaluates a strategy for de-clustering the parity encoding in a redundant disk array. This de-clustered parity organization balances cost against data reliability and performance during failure recovery. This is done by improving parity organization and by reducing additional load on surviving disks during the reconstruction of a failed disk’s contents. This proposal is specially targeted to arrays used in continuous-operation systems.

Chapter 14 presents EVENODD, a method for tolerating up to two disk failures in RAID architectures. This method employs the addition of only two redundant disks and consists of a simple exclusive-OR computation. This redundant storage is optimal, in the sense that two failed disks cannot be retrieved with less than two redundant disks. A major advantage of this proposal is that it utilizes the same parity algorithm present in typical RAID-5 controllers, and thus can be implemented on standard RAID-5 controllers without any hardware changes.

Part IV: Caching and Prefetching

Caching and prefetching techniques are generally placed in the second level in the I/O hierarchy. These two mechanisms greatly enhance performance of file systems or I/O subsystems. Concurrent access can be made while accessing from the cache within a single device to the cache of the whole file system. Furthermore, there have been proposals to use the available parallelism in the network to improve the performance and availability. For instance, some systems have used the available I/O devices in parallel, while others have used, in a coordinated way, the available memory to increase the effectiveness of caching and prefetching. The five chapters in this part present the most significant work done in this area.

Chapter 15 presents a way to improve the performance and readability of storage systems by having two caches working in parallel. This new cache architecture, which is called RAPID-Cache, consists of two redundant write buffers on top of a disk system. One of the buffers is primary cache made of RAM or NVRAM and the other is a backup cache containing a two-level hierarchy: a small NVRAM buffer on top of a log disk. The RAPID-Cache presents an asymmetric architecture with a fast-write-fast-read RAM being a primary cache and a fast-write-slow-read NVRAM-disk hierarchy being a backup cache. This asymmetric cache architecture allows cost-effective designs for very large write caches.

Chapter 16 presents aggressive, proactive mechanisms that tailor file-system resource management to the needs of I/O-intensive applications. In particular, the authors present how to use application-disclosed access patterns (hints) to exploit I/O parallelism, and to dynamically allocate file buffers among three competing demands: prefetching hinted blocks, caching hinted blocks for reuse, and caching recently used data for un-hinted accesses.

Chapter 17 is one of the first papers on prefetching for parallel applications. It presents practical prefetching policies that base decisions only on on-line reference history. It also studies the behavior of the proposed policies depending on several architectural parameters.

Chapter 18 presents a way to improve caching effectiveness by using all the available memory in the network. Cooperative caches, proposed in the NOW project, were the first step towards the usage of remote caching, but they had coherence problems that had to be overcome using complex and time consuming algorithms. This chapter proposes a simple, yet powerful and effective approach to solve coherence problems.

Chapter 19 presents a set of algorithms known as Collective Buffering algorithms. These algorithms provide a general solution for collective parallel I/O, which seeks to improve I/O performance on distributed memory machines by utilizing global knowledge of the I/O operations, including data distribution, file layout, and architectural characteristics.

Part V: Parallel File Systems

File systems hide low-level details associated with accessing storage devices and logically couples storage devices and caches. This part describes some of the popular parallel file systems designed recently. Each of these file systems makes special efforts to improve one of the possible issues of the design such as user interface, data placement, portability, extensibility, fault tolerance, etc.

Chapter 20 presents Vesta, a parallel file system designed to provide parallel file access to applications running on clusters with parallel I/O subsystems. Vesta uses a new abstraction of files where files are not a sequence of bytes, but rather they are a set of partitions (or disjoint sequences of bytes) that are accessed in parallel. The partitioning, which can also be changed dynamically, reduces the need for synchronization and coordination during the access. Some control over the layout of data is also provided, so the layout can be matched with the anticipated access patterns.

Chapter 21 presents Zebra, a network file system that increases throughput by striping file data across multiple servers. Rather than striping each file separately, Zebra forms all the new data from each client into a single stream, which it then stripes using an approach similar to a log-structured file system. This provides high performance for writes of small files as well as for reads and writes of large files. Zebra also writes parity information in each stripe in the style of RAID disk arrays; this increases storage costs slightly but allows the system to continue operation even while a single storage server is unavailable.

Chapter 22 discusses the Portable Parallel File System (PPFS), a user-level library that supports rapid experimentation and exploration of data placement and data management policies. The PPFS includes a rich application interface, allowing the application to advertise access patterns, control caching and prefetching, and even control data placement. PPFS is both extensible and portable, making possible a wide range of experiments on a broad variety of platforms and configurations.

Chapter 23 presents the Global File System (GFS), a distributed file system based on shared network storage. GFS clients directly connect to storage devices through switched channel networks called Storage Area Networks (SANs). Clients view storage as locally attached, although no single computer owns or controls these network devices. No direct

communication exists between computers; GFS clients remain independent from failures and bottlenecks of other clients. In order to achieve atomic modification, GFS has a storage-device-managed locking mechanism.

Chapter 24 proposes a way to design serverless network file systems. While traditional network file systems rely on a central server machine, a serverless system utilizes workstations cooperating as peers to provide all file system services. Any machine in the system can store, cache, or control any block of data. This approach uses this location independence, in combination with fast local area networks, to provide better performance and scalability than traditional file systems. Furthermore, because any machine in the system can assume the responsibilities of a failed component, a serverless design also provides high availability via redundant data storage.

Part VI: Parallel I/O Systems

Based on previous theory in designing high performance storage systems with parallel I/O support, many high performance mass storage systems were built by various ways. Among them, RAID-II is an early prototype that illustrates the feasibility and advantage of the RAID architecture. Petal, RAID-x and Tertiary Disks are the three major works on distributed software RAID. Network-Attached Secure Disk (NASD) is another new way to design network attached storage systems. Storage area networks enable the storage to be externalised from the server, and allows storage to be shared among multiple servers. These systems are discussed in detail in this part.

Chapter 25 surveys a few popular parallel I/O subsystems developed recently and presents design issues including system configuration, reliability, and file systems.

Chapter 26 proposes a solution to the problem that the I/O bandwidth is limited by the memory bandwidth of the server. RAID-II was designed to deliver better disk array bandwidth to file server clients. A custom-built crossbar memory system called the XBUS board connects the disks directly to the high-speed network, allowing data for large requests to bypass the server workstation.

Chapter 27 discusses the design, implementation, and performance of Petal, which consists of a collection of network-connected servers that cooperatively manage a pool of physical disks. To a Petal client, this collection appears as a highly available block-level storage system that provides large abstract containers called virtual disks.

Chapter 28 describes the Network-Attached Secure Disk (NASD) storage architecture. NASD provides scalable storage bandwidth without the cost of servers used primarily for transferring data from peripheral networks (e.g. SCSI) to client networks (e.g. Ethernet). NASD is based on four main principles: direct transfer to clients, secure interfaces via cryptographic support, asynchronous non-critical-path oversight, and variable sized data objects.

Chapter 29 presents RAID-x (*redundant array of inexpensive disks at level x*), a new architecture for distributed I/O processing on a serverless cluster of computers. The RAID-x architecture is based on a new concept of *orthogonal striping and mirroring* (OSM) across all distributed disks in the cluster. The advantage of this approach lies in a significant improvement in parallel I/O bandwidth, hiding disk-mirroring overhead in the background, and greatly enhanced scalability, and reliability for cluster computing applications.

Chapter 30 shows the suitability of a “self-maintaining” approach to Tertiary Disk, a large-scale disk array system built from commodity components. Instead of incurring the cost of custom hardware, the authors attempt to solve various problems by design and software.

Chapter 31 describes the modeling issues taken into account when designing a SAN such as several interconnection topologies, both real-world I/O traces and synthetic I/O traffic, creation of message packets, failures in links, different routing algorithms and switch architectures, etc. Additionally, the authors analyze the effect of several parameters (such as switch architecture and a variable number of faulty links in the network) on SAN performance.

Part VII: Parallel I/O Programming Paradigms

One of the key concerns with parallel I/O systems is the lack of availability of robust tools or interfaces for expressing I/O access patterns. A number of studies have shown that existing interfaces do not meet the requirements of parallel I/O due to their restricted expressiveness capability. Due to this limitation, parallel I/O systems are unable to capture the knowledge of the access patterns of the application as a whole. This part attempts to provide guidelines on how programmers can express application level knowledge that guides the underlying system to improve the performance of the I/O operations.

Chapter 32 presents the MPI-IO interface that is proposed as an extension to the MPI standard. This interface is designed to allow the programmers to develop truly portable programs. MPI-IO supports high-level interface to describe the partitioning of file data among processes, a collective interface describing complete transfers of global data structures

between process memories and files, asynchronous I/O operations, allowing computation to be overlapped with I/O, and optimisation of physical file layout on storage devices.

Chapter 33 addresses the problem of out-of-core computations, where disk storage is treated as another level in the memory hierarchy, below cache, local memory, and remote memories. Current tools used to manage this storage are typically quite different from those used to manage access to local and remote memory. This disparity complicates implementation of out-of-core algorithms and hinders portability. This chapter describes a programming model that addresses this problem. This model allows parallel programs to use essentially the same mechanisms to manage the movement of data between any two adjacent levels in a hierarchical memory system.

Chapter 34 evaluates Active Disk architectures, which integrate significant processing power and memory into a disk drive and allow application-specific code to be downloaded and executed on the data that is being read from (written to) disk. The key idea is to offload the bulk of the processing to the disk-resident processors and to use the host processor primarily for coordination, scheduling and combination of results from individual disks. To program Active Disks, the authors propose a stream-based programming model that allows disklets to be executed efficiently and safely.

Chapter 35 proposes a new technique, disk-directed I/O, to allow disk servers to determine the flow of data for maximum performance. This technique provides consistent high performance that is largely independent of data distribution, and obtains a bandwidth close to the peak bandwidth the disk can achieve.

Part VIII: Parallel I/O Applications and Environments

The data intensive applications such as scientific programs, multimedia servers, specific databases, and digital libraries require efficient parallel I/O systems and easy-to-use environment for improved response time and throughput. The runtime environments must be able to capture application characteristics dynamically and adapt themselves to provide the best performance. The chapters included in this section identify application requirements and discuss mechanisms, techniques, policies used in developing adaptive runtime environments.

Chapter 36 investigates the needs of some massively parallel applications running on distributed-memory parallel computers at the Argonne National Laboratory and identifies some common parallel I/O operations. For these operations, a library has been developed to hide the details of the actual implementation (such as the number of parallel disks) from the application while providing good performance.

Chapter 37 presents a parallel image-server architecture that relies on arrays of intelligent disk nodes, each disk node being composed of one processor and one or more disks. This chapter analyzes the real-time behavior of two multi-processor multi-disk architectures: the GigaView and the Unix workstation cluster. The GigaView incorporates point-to-point communication between processing units, and the workstation cluster supports communication through a shared bus-and-memory architecture.

Chapter 38 studies mirroring and software RAID schemes for clustered multimedia servers. It evaluates different placement strategies that guarantee high availability in the event of disk and node failures, while satisfying the real-time requirements of the streams. It examines various declustering techniques for spreading the redundant information across disks and nodes and shows that random declustering has good real-time performance.

Chapter 39 presents the requirements for a high-performance, scalable digital library of multimedia, together with a layered architecture for a system that addresses the requirements. To scale as the amount of data increases, the object management component is layered over a storage management component, which supports hierarchical storage, third - party data transfer, and parallel I/O.

Chapter 40 analyses the I/O behavior of several versions of two scientific applications on the Intel Paragon XP/S. The versions involve incremental application code enhancements across multiple releases of the operating system. Studying the evolution of I/O access patterns underscores the interplay between application access patterns and file system features. The focus is placed on the distribution of I/O request sizes and volume, temporal access patterns, and the distribution of total I/O time across processors.

Chapter 41 presents Mitra, a scalable storage manager that supports the display of continuous media data types like audio and video clips. To reduce the cost of storage, it supports a hierarchical organization of storage devices and stages the frequently accessed objects on the magnetic disks. For the number of displays to scale as a function of additional disks, Mitra employs staggered striping. It implements three strategies to maximize the number of simultaneous displays supported by each disk, which are evaluated in this chapter.

Part IX: Emerging Technologies and Future Trends

The emerging technologies such as XML and InfiniBand are expected to make greater impact on storage technologies. The usage of XML technologies encourages interoperability and portability of systems and applications in heterogeneous environment. The InfiniBand enables federation of devices providing storage and high-performance computational services in networked environment. The invited chapters included in this section present authors' vision on these emerging trends and technologies in greater depth.

Chapter 42 presents the InfiniBand Architecture (IBA), which is a new industry-standard architecture for server I/O and inter-server communication. It was developed by the InfiniBand Trade Association (IBTA) to provide the levels of reliability, availability, performance, and scalability necessary for present and future server systems significantly better than can be achieved with bus-oriented I/O structures. This chapter discusses the InfiniBand Architecture and its components and concludes with industry implications.

Chapter 43 proposes the use of a self-describing format to store the data in files. Instead of storing the data in the most compact form possible, store it in an extended form that includes an explanation about the format used, or rather, about the meaning of the data. This has been recognized for a long time in business applications, and has lead to a progression of so-called *markup languages*' that has culminated with XML, the extensible markup language. It is high time that the same principles be applied to scientific data as well.

Chapter 44 presents a study of the current I/O programming (optimization) techniques at different software levels and discusses their deficiencies. Then, it also presents a new I/O optimization approach that demands contributions from application writers, optimizing compilers, and run-time systems.

Chapter 45 is essentially an annotated bibliography of papers and other sources of information about scientific applications using parallel I/O. The contents of this chapter will be of great use for I/O-system designers wishing to learn the needs of real applications.

Readership

The book provides state-of-the art information on high performance mass-storage systems, parallel I/O, applications, and emerging technologies in storage networking, information exchange, and policy-based device federation. The book should appeal to architects, designers, developers, and programmers of RAID and parallel I/O systems both in academia, the government, and industry. The mechanics of I/O systems will also be of great interest to application developers as this may assist them in solving performance degradation problems related to I/O in their code.

This book is particularly useful for graduate level courses and research students. It can be used as a text for research-oriented, or seminar-based, senior graduate courses. Graduate students will find the material covered by this book to be stimulating and inspiring. Using this book, they will be able to identify interesting and important topics for their Master's and Ph.D. work. The book can also serve as a supplementary material for regular courses, taught in Computer Science, Computer Engineering, Electrical Engineering, and Computational Science and Information schools.

Resources on the Web – "High Performance Mass Storage and I/O" Info Centre

The various software systems discussed in this book are freely available for download via the Web. The book's Web site is,

```
http://www.buyya.com/superstorage/
```

The site will include pointers to educational material, software, projects, and standards. The book's Web site will be regularly updated with pointers to future advances in storage and I/O technologies as and when they become available. We encourage readers to let us know of pointers to any useful resources they encounter so that we can provide a hot link to them from our Web site.

Acknowledgments

At the outset, we express our gratitude to all the contributing authors for their time, effort, and understanding during the preparation of the book. Our special thanks go to all those authors who have updated their chapters to reflect recent developments in their area.

We thank David Patterson for writing the foreword to our book and setting the vision.

We would like to thank Dror Feitelson, David Kotz, Alok Choudhary, Mahmut Kandemir, and Greg Pfister for contributing invited chapters on emerging technologies and future trends in large-scale storage and high performance I/O.

We thank the anonymous reviewers for their valuable comments and suggestions on improving the quality of the book. We appreciate the support of Dror Feitelson, Garth Gibson, and Gerhard Joubert for dealing with reprint permission issues involved with the publishers holding copyright on papers included in the book. Some of the papers were not in an electronic form and they had to be retyped to maintain a uniform look and feel of the whole book. We thank Penny Stout, Qin Xin, Shirley Connelly, and Ligang He for their assistance in retyping them. We thank Dan Hyde, Rob Gray, and Mark Baker, and Rob Ross for their comments on improving the book.

We thank our publisher and our editorial colleagues at the IEEE Press. In particular, we acknowledge Marilyn Catis, John Griffin, Robert Bedford, and Anthony VenGraitis for handling our book proposal and assisting in the development of the manuscript.

We thank the Association of Computing Machinery (ACM) and Kluwer Academic Publishers for granting permission to reprint the articles in the book.

We acknowledge the support of the University of Hong Kong (Area of Excellence Information Technology development fund), the Spanish Ministry of Education (under the TIC-95-0429 contract), Monash University and the Australian Government (international research scholarships).

Last, but not least, we thank our family members: Hai's wife Hua Gao and daughter Xiaoxiao; Toni's wife Glòria; and Rajkumar's wife Smrithi and daughter Soumya for their love, support, and understanding during this editorial endeavor.

We hope that readers will find the contents of this book useful. We welcome comments and suggestions for further improvements on the book.

Hai Jin
University of Southern California, Los Angeles, USA and
University of Hong Kong, Hong Kong
http://ceng.usc.edu/~hjin/

Toni Cortes
Universitat Politècnica de Catalunya, Barcelona, Spain
http://www.ac.upc.es/homes/toni/

Rajkumar Buyya
Monash University, Melbourne, Australia
http://www.buyya.com/

June 2001