lex Vrenios

Linux® Cluster Architecture

Alex Vrenios

Linux Cluster Architecture

SAMS

201 West 103rd Street, Indianapolis, Indiana 46290

Linux Cluster Architecture

International Standard Book Number: 0-672-32368-0

Library of Congress Catalog Card Number: 752063323687

Printed in the United States of America

First Printing: June 2002

05 04 03 02 4 3 2

Trademarks

Warning and Disclaimer

Acquisitions Editor
Kathryn Purdum

Development Editor
Steve Rowe

Managing Editor
Charlotte Clapp

Project Editor
George E. Nedeff

Copy Editor
Barbara Hacha

Indexer
Sharon Shock

Proofreader
Linda Seifert

Technical Editor
Kurt Wall

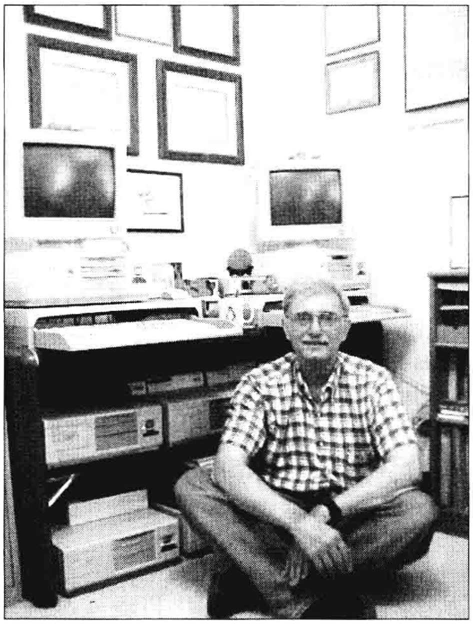
Team Coordinator
Amy Patton

Multimedia Developer
Dan Scherf

Interior Designer
Gary Adair

Cover Designer
Alan Clements

Page Layout
Brad Lenser



About the Author

Alex Vrenios is Chief Scientist at the Distributed Systems Research Lab, a technical consulting group specializing in the performance measurement and analysis of distributed computing architecture. He holds a B.S. and M.S. in Computer Science, is the author of numerous articles, and is a Professional member of the ACM and a Senior member of the IEEE. Alex started his career working with communication and data transmission software on mainframe computers. When he made the transition to the networked C/Unix environment and found out about interprocess communication, distributed algorithms, and cluster computing, he never looked back. After nearly 30 years in industry, he believes it is time to organize his experiences and passions for publication in an effort to leave a contribution for others to build on. He married his lovely wife, Diane, 13 years ago. They have two Miniature Schnauzers named Brutus and Cleo. His other interests include amateur radio, astronomy, photography, bicycling, hiking, and riding a bright red Honda PC800 motorcycle.

Dedication

for Diane

Acknowledgments

I would like to thank my lovely wife, Diane, for her patience in allowing me to undertake this long-term project when we have plenty of other demands on our time, for her diligence in reading, rereading, and gently suggesting improvements to the quality of this work, and for her confidence in my ability to present these complex technical issues.

Let me also thank the editors at Sams Publishing, without whose assistance and direction I would never have completed this work.

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book, and that because of the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email:	`opensource@samspublishing.com`
Mail:	Mark Taber Associate Publisher Sams Publishing 201 West 103rd Street Indianapolis, IN 46290 USA

Reader Services

For more information about this book or another Sams title, visit our Web site at `www.samspublishing.com`. Type the ISBN (excluding hyphens) or the title of a book into the Search field to find the page you're looking for.

Introduction

You may have opened this particular book because you heard or read something recently about cluster computing. It is definitely a hot topic! You might be a hobbyist who enjoys the hands-on experience of working with computers or an engineer or engineering manager who needs to understand the trade-offs and the pitfalls in cluster computer development. I hope that when you finish reading this book, you will either have or know how to design and build a working cluster computer. This may help you understand what kinds of problems your development project is likely to encounter. It may allow you to test and tweak distributed algorithms that are destined to run on hardware that is still under development, in parallel with your software. Or your cluster might be the first parallel Web server in your local computer club. Whatever your motivation, I want you to know how much I enjoyed learning and employing the skills necessary to build such systems; I consider it an honor and a privilege to be the one to pass them along to you.

Alex Vrenios
Phoenix, AZ

Contents at a Glance

Table of Contents

Introduction

You may have opened this particular book because you heard or read something recently about cluster computing. It is definitely a hot topic! You might be a hobbyist who enjoys the hands-on experience of working with computers or an engineer or engineering manager who needs to understand the trade-offs and the pitfalls in cluster computer development. I hope that when you finish reading this book, you will either have or know how to design and build a working cluster computer. This may help you understand what kinds of problems your development project is likely to encounter. It may allow you to test and tweak distributed algorithms that are destined to run on hardware that is still under development, in parallel with your software. Or your cluster might be the first parallel Web server in your local computer club. Whatever your motivation, I want you to know how much I enjoyed learning and employing the skills necessary to build such systems; I consider it an honor and a privilege to be the one to pass them along to you.

Alex Vrenios
Phoenix, AZ

1 Linux Cluster Computer Fundamentals

IN THIS CHAPTER

This book is about cluster computers. It explains what they are, how they can be used, what kind of hardware is needed to build one, and what kinds of software architectures bring out the best in them.

Clusters are a hot topic. What is unique about them is that the personality of their architecture—how they appear to behave to their users—is so dependent on their software. The same collection of computers interconnected via some network can become a supercomputer, a fault-tolerant server, or something completely different—something you design!

These architectural varieties are all a part of *distributed computing*, a network of individual computers working together at solving a problem or providing a service. A cluster computer is a distributed system; this concept will become more familiar in the near future as the world's problems become more geographically dispersed. Examples include international financial transactions, global weather forecasting, and rapid military deployment. The farthest reaches of the world are getting closer together through the proliferation of distributed computer and communication systems. Exposure to their underlying principles might help you get a better job some day or move up the ranks in one you already have.

Why a Cluster?

Let's begin our journey with a couple of loose definitions. First, just what is a *cluster computer*? A cluster computer is a set of individual computers that are connected through specialized hardware and software, presenting a single-system image to its users.

What is a cluster computer? A bunch of PCs connected on a network does *not* qualify as a cluster computer. To be a cluster computer, each of these PCs must be running software that takes advantage of the fact that other PCs are available. They must be made to act like a team, working together and supporting one another toward a common goal, presenting a single-system image to their users.

What is a *single-system image* then? A single-system image is one that is indistinguishable from that of a single computer. Its users are unaware that more than one computer exists.

Consider the PC on your desk connected to some distant Web site over the Internet. With each click of your mouse, you are downloading that Web page's text and image file data into your browser for display. The Web server could be dozens of PCs in a network—a cluster server. Each time your browser asks for new data, its request packet could be broken up and passed along to several of the other computers, which read the files from disk and send them back to your browser. You are unlikely to be aware that these computers were working together to service your requests. Such a cluster server presents a single-system image to your browser, to the client, and to you, the user.

Why bother? Why not just start with a single high-speed server and replace it with a faster one each time your client resource requirement grows? The answers to these questions encompass some complex issues that are the subject of much of today's computer science research: cost-effectiveness, scalability, high availability, and fault tolerance. You'll learn more about these issues later in the chapter.

Computer science brought home the concept of how a *stepwise process* leads to a desired goal. Further, it showed that any process toward a desired goal can be broken down into a set of steps, and that each step can be further broken down into smaller steps, and so on. If all the achievements to date and all the motivations that continue to drive computer science research could be condensed into a single word, that word might be *overlap.*

Overlap refers to the capability to perform two or more functions simultaneously. Early computers were required to stop all their other processing just to print a line of characters. In contrast, modern computers routinely execute instructions while redrawing a screen image, reading from or writing to the hard drive, printing, and so on. Exploiting even the smallest potential for overlap can yield a tremendous improvement in performance.

A cluster computer is a type of parallel processor called a *multicomputer* to distinguish it from a true multiprocessor, which has a single physical memory that is shared among its processors. Later chapters discuss some different kinds of parallel processors. For now, let's look toward exploiting overlap at the task execution level. You can do this in a cluster by finding tasks or portions of tasks that can be executed in parallel and by scheduling these chunks on different processors in your cluster. As each chunk finishes, its results are returned to the scheduling processor. Individual results are combined into what serial processing of these chunks might produce. This combined result is fed into the next high-level step, and so on, until all the task's processing is complete.

Examining the interdependencies among the steps of any process, you can see that some steps must be executed in strict sequence, the outcome of one feeding the requirements of another. Other steps are independent of one another and may be executed in parallel, allowing you to exploit any parallelism inherent in that process. The benefit of executing steps in parallel is a shorter process-completion time. The cost is in the extra resources required to perform this parallel execution and the time required to schedule their parallel execution. Cluster computing is one kind of parallel processing. You'll see some other forms in later chapters.

Gene Amdahl put the benefits of parallel processing in perspective back in the 1960s. He pointed out that of all the time spent by a parallel processor executing your application, it is only in that part of it where parallelism occurs, that any time can be saved, and any time savings will be in proportion to the amount of inherent parallelism, minus the overhead in managing parallel processes. This concept later came to be known as Amdahl's Law.

If the inherent parallelism in your application is so small that most of the steps have to execute sequentially, you probably do not want to consider using a parallel architecture. There are enough common processes with inherent parallelism, however, to generate a lot of interest in parallel processors. The cluster architecture, in particular, gives you the benefits of a scalable parallel processor at a very low cost and a potential for high availability through software fault tolerance.

Architectural Design Features

Cost-effectiveness, in terms of performance and reliability, is the single most important design issue. The goal is to assemble several individual computers and to sacrifice some resources in managing them as a cluster system. This should result in a higher level of performance and reliability than what you can get from a single computer in the same price range.

Scalability, high availability, and fault tolerance are the three features necessary to achieve high performance and reliability. If the support software overhead for any of these is too high, the cluster may not expand beyond a trivial number of nodes, or it may fail to sustain even a modest query load. The following sections look at each of these features.