Chapter 1

Introduction

The evolution of modern electronic computers began only 50 years ago and has been driven to a large extent by rapid advances in electronic technology during that period. The first computers ran one application at a time. Without the benefit of operating systems or compilers, application programmers were responsible for managing all aspects of the hardware. The introduction of compilers allowed programmers to express algorithms in abstract terms without being concerned with the bit level details of their implementation. Time sharing operating systems allowed hardware resources to be efficiently utilized by several applications simultaneously.

As electronics evolved from vacuum tubes to transistors to integrated circuits computers became faster and more powerful. The complexity of hardware and systems software increased to support larger numbers of users and complex applications running on faster and faster systems.

More recently the trend towards complex monolithic computer systems has changed. VLSI advances over the last 15 years have made computers cheaper and smaller. This has enabled computer manufacturers to deliver small systems that fit on or beside a desk and can satisfy the computing needs of many users more cost effectively than timesharing systems. One drawback to this approach is that it results in huge amounts of excess computing capacity: few users can generate enough work to keep their systems busy all the time.

Another trend that can be observed over the past 50 years is that scientists, mathematicians and engineers have consistently been able to envision applications that cannot be accommodated by the computers available to them. This has led to the evolution of scientific computing technology separate from commercial computing.

Microelectronics have evolved to the point where future performance improvements of several orders of magnitude cannot be expected. Electronic signals cannot propagate faster than the speed of light. This will be a limiting factor in the improvements that can be expected in processor and memory system speed. Instead, computer engineers have sought to answer the challenges of scientific computing by designing hardware systems and algorithms with high levels of parallelism. Unfortunately, Parallel systems often lack generality. An architecture which provides very good performance on one problem or algorithm may perform poorly on others. Lack of generality may even be manifested as poor scalibilty, where a particular algorithm running on a given machine runs satisfactorily for a given instance of a problem but is inefficient on much larger or smaller instances.

Consequently, two problems currently facing the field of computer engineering can be identified: (1) The trend towards workstation based computing results in large amounts of underutilized processing resources, and (2) Specialized scientific computing architectures benefit some computing tasks more than others. This dissertation will outline how both these problems have been addressed by incorporating network attached workstations, and scientific computers of various architectures, into a distributed computing environment. The environment provides tools to allow users to develop applications in terms of an interconnection of constituent tasks and a run time system which manages execution of these tasks in shared high performance computers and workstations connected across a wide area network. This distributed computing environment, which enables users to define and efficiently execute heterogeneous distributed applications, is referred to as a Virtual Computing Environment (VCE).

1.1 Research Objectives

The main focus of the VCE is to:

- Provide a software engineering environment that will enable scientists and engineers to build applications that use multiple high performance computing systems.
- Make the computing power of idle and underutilized computers available to these applications.
- Provide a run time system that employs effective scheduling and dispatching methods to maximize performance.
- Allow VCE applications to execute on shared hardware without unduly impacting non-VCE users. That is, the VCE should not require dedicated hardware.
- Use existing tools and technologies whenever possible so that the the VCE will not be too complex or too expensive to implement.
- Avoid operating system modifications so the system will be portable and can support high levels of heterogeneity.
- Be modular and flexible enough so that future software engineering tools and hardware platforms can be incorporated easily.

The use of multiple high performance computers has been shown to be an effective way to improve the performance of real world applications (Section 1.1.2). However, dedicating multiple supercomputers to a single application is often not feasible. Also, the skill required to implement and run these applications is considerable. By meeting the objectives listed above, the VCE will allow such applications to be developed by a larger group of users (not just high performance computing experts) and executed on shared hardware. Existing distributed computing environments are not robust enough to achieve this. Section 1.2 and Chapter 2 will discuss a wide variety of distributed computing environments and tools. None of them satisfies all these objectives.

The objective of the VCE design is to describe and validate a strategy for providing a set of software engineering and application execution tools which satisfies the items listed above. Central to this strategy is the differentiation between programming-in-the-small and programming-in-the-large [35]. Many tools exist to support parallel programming. The VCE, rather than focusing on development of individual parallel programs, is concerned with allowing users to build large scale applications from multiple parallel (or sequential) programs running on distributed heterogeneous computers. A VCE application designer specifies a set of programs and indicates how they are synchronized and communicate. Each program (task) in the application may have its own hardware requirements. The application designer is not concerned with how tasks are implemented. General purpose reusable task implementations are developed by parallel programming specialists. Application designers concentrate on the structure of applications in a problem area where they have expertise. The VCE run time system dynamically selects hosts for each task and manages their dispatching and communication. The VCE provides a technology independent environment for the development of reusable high performance tasks, development of applications that use these tasks and automated execution of the applications.

1.1.1 The Needs of Scientific Computing

As has been the case throughout the brief history of electronic computing, the computing needs of scientific and engineering applications continue to tax the capabilities of current hardware. The requirements of such applications have led to the development of specialized software systems to manage large numbers of dedicated workstations which work cooperatively on scientific applications [41] [88]. A variety of supercomputer architectures [58] such as SIMD, MIMD and vector machines have also been designed for use on such applications. Each architecture represents a different set of trade-offs and will give different levels of performance for any given problem. Further,

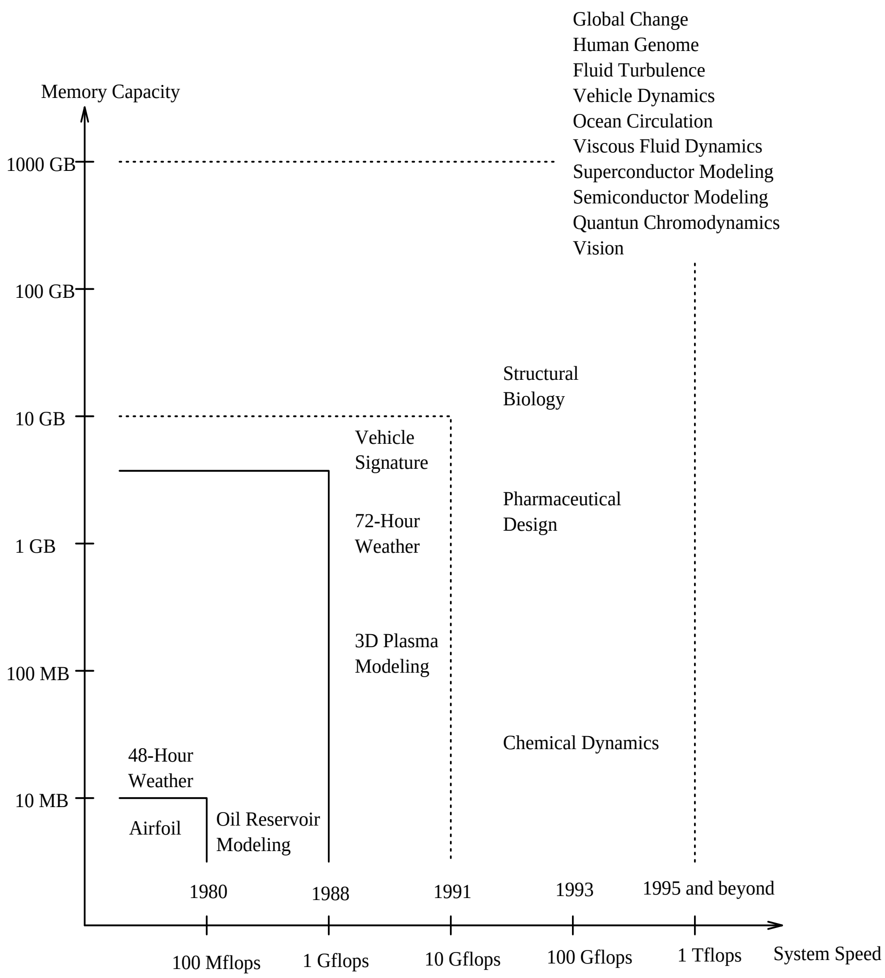

Figure 1: Computing and Storage Requirements of Grand Challenge Applications [58]

as illustrated in Figure 1 many applications that will be of interest to researchers in the near future will exceed the computing resources that could reasonably be packaged in any single processor complex.

1.1.2 Applications of Multiple High Performance Computers

The next step in developing these applications is likely to entail the use of multiple supercomputers to attack a single problem. An example of such an effort is described in [74]. Part of the project involved execution of a global climate model. The model was implemented as two separate components: a physics model which computes the effect of non-atmospheric forces on climatelogical conditions, and a dynamics model which simulates the evolution of the fluid flows using appropriate equations. The results produced by the physics component are supplied to the dynamics model as forcing terms. These two application components have different computational properties. The physics model can be efficiently parallelized by dividing the simulation domain (the atmosphere) into vertical columns. Because the physics component is primarily concerned with calculating the effects of localized phenomenon, little communication between processors is needed and high performance can be achieved on massively parallel computers. Conversely, the atmospheric dynamics model incurs much higher communication overhead which reduces its efficiency when it is executed on a parallel machine. Table 1 shows the running time in seconds per simulated day for both components of the application on seven different high performance computers. When the two components are executed on different machines they can execute in parallel and communication operations can be overlapped with computation so that the distribution overhead is minimal. If both components of the application are executed on one machine, the best possible running time is 83 seconds per simulated day on the CRAY C90. By using two supercomputers, the CRAY C90 and the Intel Paragon, performance was improved to 40 seconds per simulated day.

The performance data in Table 1 reflects benchmarks performed on otherwise idle machines. If the application were run on shared hardware its performance would be effected by transient load conditions. Using benchmark performance data to staticly allocate application components to hardware resources may not achieve the best possible results. For instance, the data shows that the physics component performs equivalently on both the IBM SP1 and the Intel Paragon, which suggests that if both kinds of machines are available at run time, choosing the one that is less loaded will improve the application's performance. This suggests the need for automated scheduling technology to make and execute these run time decisions.

In addition to the need for computer systems with adequate processing power to

Table 1: Performance of Two Components of a High Performance Application on Various Computing Platforms [74]

Platform	Dynamics Component	Physics Component	Total Running Time
CRAY C90	29	54	83
CRAY Y-MP	61	104	165
Intel Paragon	132	40	172
Intel Delta	192	48	240
IBM SP1	100	40	140
CM-5	125	45	157
DEC ALPHA Cluster	137	116	253

tackle their applications, scientists also need an environment which allows them to construct applications efficiently [7] [53]. Scientists and engineers are concerned with expressing the structure of their computations and obtaining computer systems which achieve acceptable performance for those computations. Computing environments which require them to learn new programming languages or be concerned with load balancing and similar low level hardware issues are unlikely to succeed. Conversely, environments which allow users to build large applications by reusing and customizing existing software components are more likely to be embraced [7].

1.1.3 The Needs of Workstation Users

One goal of the Virtual Computing Environment is scalability. The VCE should provide benefits in configurations as small as a few homogeneous workstations up to networks containing many supercomputers of various architectures and large numbers of heterogeneous workstations. Because almost all anticipated VCE configurations will include some workstations, the current research gives considerable attention to issues involved in optimizing the use of such machines in distributed environments.

Studies of workstation usage patterns were conducted by Mutka [81] and Mutka and Livny [80]. Such a study was also included as background material by Krueger and Chawla [67]. All these studies found that networks of workstations were largely underutilized even at peak times. The characterization of Krueger and Chawla [67] that "...unused capacity rarely falls below 87%" was typical of all three sources.

Mutka and Livny [80] found that most workstation users fell into one of three categories:

- **Casual users** who perform text editing and mail functions which never fully utilize their workstations.
- **Sporadic users** who occasionally run short non-interactive tasks such as compilations.
- **Frustrated users** who run large jobs on a regular basis and are constrained by the capabilities of their workstations.

The goal of the VCE is to provide relief to frustrated users by giving them friendly and intuitive tools for exploiting the excess resources controlled by other users. The capabilities of the VCE can also be used to provide benefits for sporadic users by allowing them to run large tasks on faster machines and by providing tools like parallel *make* [60, 109]. These benefits are realized without imposing significant performance penalties on casual or sporadic users. Such penalties could cause users to resist participating in the VCE. Workstation users expect to receive consistent service levels at all times (unlike users of timesharing systems who must accept performance degradations during the busiest times of the day).

1.2 Limitations of Current Approaches

Chapter 2 will discuss a wide variety of distributed system paradigms which have been proposed to address some of the issues which have motivated the VCE design. None of those technologies provide an adequate solution to the problems associated with designing, implementing and executing very large applications in a large shared resource network environment. This sections will outline the shortcomings of current distributed system technologies so that the contribution of the VCE approach can be demonstrated.

1.2.1 Distributed Operating Systems

Not only will the VCE provide a platform for the development and execution of large applications and exploit idle resources in computer networks, it has been engineered so as to be cost effective. No software system can be considered successful if it costs so much to develop and maintain that it cannot be made available to the user community it was designed for.

The VCE can coexist with other software. Users who prefer to use other tools to do their work will not be excessively impacted by their machine's participation in the VCE. The VCE will give users a new option for running their applications. Any application that is already running in a more conventional environment and delivering satisfactory performance will not experience significant performance degradation due to a host machine's participation in the VCE.

The importance of these two criteria is demonstrated by the failure of distributed operating systems developed over the last 15 years to gain wide usage. The approach of building a special purpose kernel to run in each machine in a distributed environment has some merit. Significant performance gains can be realized by building customized communication protocols and other distributed computing support into the kernel. However, developing an operating system from scratch is an ambitious undertaking. The LOCUS project [105] for example, resulted in a kernel one third larger than the Unix kernel and required 50 man years of effort to design and develop. The costs of such a project, and the costs of maintaining such a large system, would be prohibitive.

Some distributed operating systems (such as Amoeba [78]) present user and programming interfaces which are not compatible with the familiar Unix environment. Processors in Amoeba are assigned to tasks dynamically and are largely unable to function autonomously. Such a departure from the dominant computing model (the Unix workstation model) is unlikely to be successful. Recognition of this fact is a trend in the literature: from the distributed operating systems of the 1980's and early 1990's [11, 30, 78, 85, 105] to remote execution systems in the early 1990's that used small kernel modifications to achieve global scheduling and location transparency [67, 33] to very recent load sharing systems implemented completely above the kernel [20, 24, 60, 87, 109].

1.2.2 Remote Procedure Call, Message Passing and Distributed Shared Memory Environments

Many tools are available to ease the process of developing a single application with components spread over a number of networked machines (see Sections 2.2, 2.3 and 2.4). The focus of these tools is to provide programmers with abstract communication models which will be more usable than the communication primitives (like *sockets* [96]) provided by conventional operating systems.

Remote procedure call (RPC) systems provide mechanisms for binding a calling program with a procedure running on a different machine. RPC functions are generally used to allow users to access specialized resources such as file systems, data bases or high performance CPUs resident on remote machines. RPC functions are not usually suitable for high performance applications because the procedure call interface is inherently synchronous and awkward for use in parallel processing.

Message passing systems, compared to RPC paradigms, provide a similar but more flexible communication model. The similarity is that RPC allows a sending program to send a set of arguments to a remote process which receives them as input on its invocation. With message passing the arguments are passed to a *send* function and retrieved by a remote process using a *receive* function. The flexibility of message passing is achieved by providing blocking and non-blocking send and receive operations which facilitate parallel operations. Most message passing systems also provide *broadcast* operations which allow a master process to send parameters to a collection of slave processes which perform some computation in parallel. Such an operation is awkward to replicate with RPC primitives because the procedure call model supports a one-to-one communication pattern.

The distributed shared memory model is an even more flexible paradigm. Instead of *send* and *receive* functions, *write* and *read* semantics are used to pass data among remote processes. A distributed shared memory application might have one master component that places sets of parameters into a shared buffer where they are read by slave processes which perform computations and place the results in the buffer for retrieval by the master process. Distributed shared memory allows the application programmer to create synchronization as needed using semaphore techniques. Atomic "read and remove" operations are usually provided which provide sufficient serialization for many applications. These facilities allow the user to achieve as much or as little synchronization as is necessary. They can be more friendly than message passing schemes which may require explicit data conversion by the application and require message senders to maintain and supply addresses or address lists of recipients.

None of these tools represent a wholly satisfactory environment for developing high performance distributed applications or for harnessing idle resources. They are all low level programming tools that must be incorporated into application programs written in conventional languages such as C, C++ or Fortran. The goal of computing environments which support grand challenge applications must be to provide scientists and engineers with the ability to fashion applications from existing components rather than requiring them to become or supervise programming experts.

Many RPC, message passing, and distributed shared memory tools require the programmer to declare specific machines for each process to run on. This practice precludes dynamic run time load balancing and limits the potential of such systems to reclaim idle resources on a large scale.

1.2.3 Load Sharing Environments

Load sharing environments, discussed in detail in Section 2.6, provide the capability to schedule work on idle or lightly loaded networked computers. This is accomplished in a variety of ways. One approach is to compile large CPU bound jobs with a special library which allows their I/O requests to be intercepted by a run time system. Such jobs are scheduled on idle machines and their I/O operations are forwarded to their originating machines for processing. These remote execution applications can be migrated among homogeneous hosts as load conditions change. Another way to achieve location transparency and process migration among homogeneous hosts is to furnish operating system modifications which embody the functions of the remote execution library.

These systems are useful for putting otherwise idle processor resources to productive use, but their scope is limited. Systems which employ operating system modifications are costly and not portable. Any load sharing system which migrates an execution image will not be able to support heterogeneous environments. The alternative to moving an execution image is to develop a check-point restart protocol which also incurs high overhead. When non-preemptive scheduling is used, a placement decision is made for each process when it initiates. This policy reduces the autonomy of privately owned workstations. When a machine is hosting foreign tasks the owner must execute his or her own jobs remotely to obtain reasonable performance and avoid worsening the load imbalance.

These systems do not provide robust parallel processing facilitates or support the participation of high performance parallel platforms. There goal is to either to optimize placement of all work within a cluster or allow remote execution of batch job. Using idle resources for parallel processing is not supported.

1.2.4 Workstation Based Parallel Programming Environments

Workstation based parallel programming environments (Section 2.6.3) incorporate two elements; a parallel programming paradigm or language which allows an application to be divided into a set of tasks that can execute independently in parallel, and a run time system which schedules and dispatches the tasks on idle workstations. These systems have goals which are similar to the VCE's. That is, they seek to use idle computing resources to run parallel programs. However the VCE design is more robust. Workstation based parallel programming environments tend to be sensitive to the granularity of the applications being run. If a task of an application is placed on a machine, and the load on the machine increases, it may have to be restarted on a different host. This means the tasks must be large enough to not be slowed by excessive communication overhead and small enough to complete before being preempted. Applications that do not meet this criteria will not perform well. These systems have the further drawback that applications must be specially designed and implemented for them, again requiring the designer to be expert in his or her problem domain and in parallel programming as well. Facilitates are not included to exploit multiple parallel computers.

1.2.5 High Performance Application Development Tools

Section 2.7 will discuss tools whose primary intent is to ease the task of developing programs to run on parallel computers. These include languages like High Performance Fortran and Mentat, which enhance existing languages by adding capabilities to start, communicate with, and manage parallel threads of execution. Another approach to parallel program development is to provide a set of highly optimized template programs which can be customized for a particular application. The goal of these technologies is to permit the application developer to realize good performance without having to be concerned with as many low level details as are required by message passing systems or other parallel programming environments. Unfortunately, these application development tools still require application developers to determine how their problem can be partitioned among a set of processors. Automatic load balancing is not provided to any significant measure. Facilitates to manage tasks on multiple parallel machines simultaneously are not included.

1.3 The VCE Approach

The scope of the Virtual Computing Environment is broader than the distributed system technologies described in the previous section. Those systems generally address one issue such as load balancing, remote execution, or parallel program development. None of them is geared towards facilitating the use of multiple high performance computers in grand challenge applications like the one described in Section 1.1.2. The VCE allows scientists and engineers to easily construct applications from existing components and automatically execute them on any local or remote computing systems that are available at run time.

1.3.1 Structure of the VCE

In order to make development and deployment of very large high performance applications more feasible, tools are needed to ease their development and automate their execution. The VCE, therefore, is divided into two major components: a *software development module* which presents a suite of simple, graphical tools for application definition and development, and an *execution module*, which is the run time system that supports execution of VCE applications. Because high performance computing is a young discipline, The VCE uses a modular design so that as future design and development tools and hardware architectures become available they can be incorporated into the computing environment efficiently.

The current VCE design focuses on the capabilities required of the software development tools rather than detailing specific tools. The design does include a detailed specification of the interface between the software development module and the execution module. This *application description framework*, a concise syntax for describing the characteristics of a distributed application, allows development tools to be selected and interchanged without impacting the run time system (Chapter 4).

The VCE design does address all key issues of the execution module in some detail. The overall structure of the VCE is shown in Figure 2. The tools of the software development module produce application description files, task description files, and executable files. Each machine that is participating in the VCE runs a VCE scheduling/dispatching daemon as a background process. Machines that are architecturally similar are organized as groups. One member of each group is designated as the group leader. When a machine has excess capacity (according to criteria discussed in Section 5.1) it notifies its group leader, which maintains a list of group members that are available for use by VCE applications. Execution of a VCE application is managed by a *VCE execution process*. The execution process is initiated by a user on his or her workstation and is passed the name of an application description file produced by the software development module. The application description file refers to task description files and executable files. The execution process uses the information in these files to determine the characteristics of the application. It also communicates with the group leader daemons that represent groups which are good candidates to host portions of the application. These group leaders return indicators of the amount of idle capacity available within their groups. Once the execution process has parsed the application and task description files, and received information about the run time load conditions from selected group leaders, it can proceed to make placement decisions about the application's components. The benefits of this approach are explored in Chapter 3.

In designing the execution process different scheduling policies have been investigated. Among these are random scheduling, selective scheduling (where only lightly hosts are considered) and redundant scheduling. As indicated by the performance data in Table 1, often an application component will have similar performance characteristics on several platforms. If several types of machines are available as candidates for a portion of an application, the execution process may dispatch several instances of the task in order to maximize the chances that superior performance will

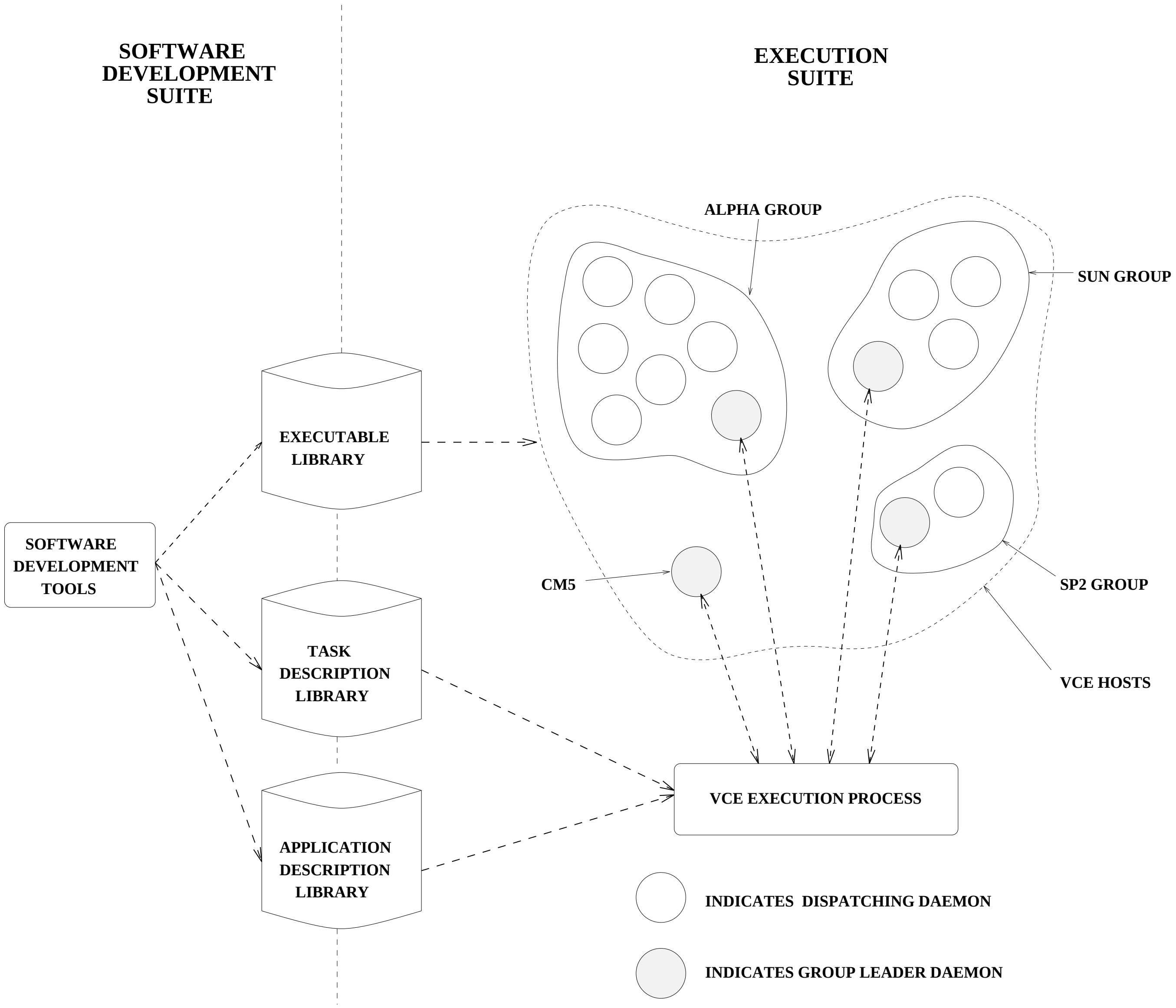

Figure 2: Major Components of the VCE

be achieved in spite of fluctuations in load conditions. Details of the mechanisms of redundant scheduling are included in Section 5.1.

Several prototype and simulated VCE systems have been constructed in order to determine the main characteristics of the VCE run time system. Experimentation has focused on compareing the VCE's redundant scheduling techniques to more conventional selective scheduling methods. Strong evidence of the advantages of redundant scheduling has been compiled. Experimental validation of the efficency of organizing host machines into groups has also been obtained. Further, a prototype implementation of the application description framework has demonstrated its utility and ease of implementation. Descriptions of the prototypes, simulations, experimental observations and analysis are included in Chapter 6

The VCE execution module also provides an infrastructure to support communication between application components (Section 5.2). This communication system makes redundant scheduling and remote execution transparent to application developers.

1.3.2 Contributions

The VCE represents a unique and valuable step forward in the design of distributed computing environments. Previous distributed computing systems have focused on load sharing (Section 2.6), easy construction of scientific applications (Section 2.7), development and execution of real time applications (Section 2.8) or support of distributed objects (Section 2.9). The VCE is a more comprehensive system. It includes a software design and development environment geared towards the creation of very large applications by scientists and engineers. The software design suite can accommodate procedural, object oriented or World Wide Web based development tools and can be extended to support new design paradigms in the future. The VCE execution environment is one of the first attempts to automatically manage distributed applications that require multiple high performance parallel computers. Advanced load measurement and scheduling techniques have been constructed for the VCE execution environment which allow applications to achieve good performance by exploiting the excess capacity of networked high performance machines.

The contributions of current VCE research can be summarized as:

- The design of an extensible software development environment that separates program design (prograaming-in-the-small) from application design (programming-in-the-large).
- The definition of an application description framework which constitutes the interface between the development and execution environments.
- The design and validation of a set of technology independent scheduling, dispatching and communication methods to manage execution of applications across multiple parallel systems.
- Incorporation of the above elements into a coherent Virtual Computing Environment.

1.4 Organization of the Dissertation

The remainder of the dissertation presents details of the VCE design, related background material and descriptions of experimental VCE prototyping and simulation efforts. The material is organized as follows;

Chapter 2 discusses a broad range of distributed computing environments. Examples of RPC and messaging passing systems, workstation based load sharing systems, workstation based parallel computing systems and parallel programming environments will be discussed and compared to the VCE. Distributed application description languages and distributed object systems, which share some of the capabilities of the VCE application description framework, are also discussed.

Chapter 3 contains a detailed summary of the VCE design. The role and functions of all the VCE components and their interfaces are outlined.

Chapter 4 presents the VCE application model. The structure of VCE applications and their construction from reusable task and object implementations is discussed. The focus of Chapter 4 is a detailed description of the VCE application framework and comparison of it to other popular application description methods.

in Chapter 5 the techniques used to manage application scheduling and communication, which underlie the VCE execution suite, are discussed. Load measurement techniques, redundant scheduling methods and techniques for managing communication among redundant task instances are described.

Chapter 6 describes two prototype VCE systems and an object oriented simulation model that were used to validate the VCE design. Experience with VCE applications, the application description framework and various scheduling techniques are outlined.

Finally, Chapter 7 contains conclusions and possible future VCE research directions.

Bibliography

[1] Adler, R. M., *Emerging Standards for Component Software*, Computer, Vol. 28, No. 3, pp. 68-76, March 1995.

[2] Advanced Visual Systems Inc., **AVS Developer's Guide (Release 4.0)**, Part No. 320-0013-02, Revision B, Advanced Visual Systems Inc., Waltham Mass., May 1992.

[3] Ahmad, I., Ghagoor, A., and Mehrotra, K., *A Decentralized Task Scheduling Algorithm and its Performance Modeling for Computer Networks*, Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, pp. 314-321, December 1991.

[4] Ahmad, I., Ghagoor, A., Mehrotra, K., Mohan, C. K., and Ranka, S., *Performance Modeling of Load Balancing Models Using Neural Networks*, Concurrency: Practice and Experience, Vol. 6, No. 5, pp. 393-409, August 1994.

[5] Allia, M., Arlat, J., Fabre, J.-C., Kanoun, K., and Powell, D., *Dependability Evaluation of the Saturation Technique*, Proceedings of the 5th Annual European Computer Conference (CompEuro '91), pp. 437-441, May 1991.

[6] Anderson, T. E., Culler, D. E., and Patterson, D. A., *A Case for NOW (Networks of Workstations)*, IEEE Micro, Vol 15, No. 1, pp. 54-64, February 1995.

[7] Armstrong, R., and Macfarlane, J., *The Use of Frameworks for Scientific Computation in a Parallel Distributed Environment*, Proceedings of the Third International Symposium on High Performance Distributed Computing, San Francisco, California, pp. 15-25, August 1994.

[8] Bach, M. J., **The Design of the Unix Operating System**, Prentice-Hall Inc., 1986.

[9] Barak, A., Guday, S.,and Wheeler, R. G., **The MOSIX Distributed Operating System, Load Balancing for UNIX**, Lecture Notes in Computer Science, Springer-Verlag, Vol. 672, 1993.

[10] Barak, A., Laden, O., and Yarom, Y., *The NOW MOSIX and its Preemptive Process Migration Scheme*, Bulletin of the Technical Committee on Operating Systems and Application Environments, IEEE Computer Society, Vol. 7, No. 2, pp. 5-11, June 1995.

[11] Barak, A., and Wheeler, R. G., *MOSIX: An Integrated Multiprocessor UNIX*, Proceedings of the Winter 1989 USENIX Conference, San Diego, CA, pp. 101-112, February 1989.

[12] Barbacci, M. R., Doubleday, D. L., Weinstock, C. B., and Wing, J. M., *The Durra Language and Runtime Environment: Tools for PMS Level Programming*, **Decentralized Systems**, Proceedings of the IFIP WG 10.3 Working Conference, pp. 25-27, North Holland, 1990.

[13] Barbacci, M. R., Weinstock, C. B., Doubleday, D. L., Gardner, M. J., and Lichota, R. W., *Durra: a structure description language for developing distributed applications,* Software Engineering Journal, pp. 83-94, March 1993.

[14] Barbacci, M. R., and Wing, J. M., *Specifying Functional and Timing Behavior for Real-Time Applications*, Proceedings of Parallel Architectures and Languages Europe (PARLE) Volume 2. Springer-Verlag, Lecture Notes in Computer Science, Vol 259, pp. 124-140, June 1987.

[15] Benmohammed-Mahieddine, K., Dew, P. M., and Kara, M., *A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems*, Proceedings of the 14th International Conference on Distributed Computing Systems, pp. 484-492, June 1994.

[16] Besana, V., De Luca, J., Giard, E., and Reger, J., **Experiences with SOMobjects: Distributed System Object Model (DSOM)**, IBM International Technical Support Centers, Document No. GG24-4165-00, November 1993.

[17] Betz, M. *Interoperable Objects*, Dr. Dobb's Journal, Vol. 19, No. 11, pp. 18-32, October 1994.

[18] Birman, K. P., Schiper, A., and Stephenson, *Lightweight Causal and Atomic Group Multicast*, ACM Transactions on Computer Systems, Vol 9, No 3. pp. 272-299, August 1991.

[19] Birman, K., Cooper, R., Joseph, T., Kane, K. and Schmuck, F., **The Isis System Manual.**

[20] Blumofe, R. D., and Park, D. S., *Scheduling Large-Scale Parallel Computations on Networks of Workstations*, Proceedings of the Third International Symposium on High Performance Distributed Computing, San Francisco CA., pp. 96-105, August 1994.

[21] Bricker, A., Litzkow, M. and Livny, M., *Condor Technical Summary*, Computer Science Department, University of Wisconsin - Madison, January 1992.

[22] Butler, R., and Lusk, E., *User's Guide to the p4 Programming System*, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4801, April 1994.

[23] Carriero, N., and Gelernter, D., *Linda in Context*, Communications of the ACM, Vol. 32, No. 4, pp. 444-459, April 1989.

[24] Carriero, N., Gelernter, D., Kaminsky, D., Westbrook, J., *Adaptive Parallelism with Piranha*, Department of Computer Science, Yale University, New Haven, Connecticut, YALE/DCS/RR-954, February 1993.

[25] Carriero, N., Gelernter, D., Mattson, T. G., and Sherman, A. H., *The Linda Alternative to Message-Passing Systems*, Parallel Computing, Vol. 20, No. 4, April 1994.

[26] Casavant, T. L., and Kuhl, J. G., *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems*, IEEE Transactions on Software Engineering, Vol. 14, No. 11, pp. 141-154, November 1988.

[27] Chang, J., and Maxemchuck, N. F., *Reliable Broadcast Protocols*, ACM Transactions on Computer Systems, Vol. 2, No. 3, pp. 251-273, August 1984.

[28] Chen, J., Xie, L., and Sun, Z.-X., *An Intelligent Network Manager for Distributed Task Scheduling*, Proceedings of the 11th International Conference on Computer Communication, pp. 493-499, October 1992.

[29] Cheng, D.Y., *A Survey of Parallel Programming Languages and Tools*, RND-93-005, NASA Ames Research Center, March 1993.

[30] Cheriton, D.R., *The V Distributed System,* Communications of the ACM, pp. 314-333, Vol. 31, No. 3, 1988.

[31] Chiu, J-F, and Chiu, G-M, *Process-Replication for Fault-Tolerance and Performance Improvement in Distributed Computing Systems*, Proceedings of the Third International Symposium on High Performance Distributed Computing, San Francisco, California, pp. 236-243, August 1994.

[32] Chu, W. W., Holloway, L. J., Lan, M., and Efe, K., *Task Allocation in Distributed Data Processing*, Computer, Vol 13, No. 11, pp. 57-69, November 1980.

[33] Clark, H. and McMillin, B., *DAWGS a distributed compute server utilizing idle workstations,* Journal of Parallel and Distributed Computing, pp. 175-186, Vol.14, No.2, 1992.

[34] Dandamudi, S., *Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems*, Proceedings of the 15th International Conference on Distributed Computing Systems, pp. 484-492, June 1995.

[35] Deremer, F., and Kron, H. H., *Programming-in-the-Large Versus Programming-in-the-Small*, IEEE Transactions on Software Engineering, Vol. 2, No. 2, June 1974.

[36] Eager, D. L., Lazowska, E. D., and Zahorjan, J., *A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing*, Performance Evaluation, Vol. 6, pp. 53-68, March 1986.

[37] Eager, D. L., Lazowska, E. D., and Zahorjan, J., *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering, Vol. 12, No. 5, pp. 662-675, May 1986.

[38] Elnozahy, E. N., and Zwaenepoel, W., *Replicated Distributed Processes in Manetho*, Proceedings of the Conference on Fault-Tolerant Computing Systems, pp 222-230, 1991.

[39] Efe, K., and Schaar, M. A., *Performance of Co-Scheduling on a Network of Workstations*, Proceedings of the 13th International Conference on Distributed Computing Systems, pp. 525-531, May 1993.

[40] Fabre, J. C., *Improving Fault Tolerance in Distributed Systems: The Saturation Approach*, Proceedings of the 5th Annual European Computer Conference (CompEuro '91), pp. 634-641, May 1991.

[41] Farrell, W. E., Fox, W. Given, J. W., MacRitchie, B., and Wang, J., *Data Processing on Networks of Workstations*, Bulletin of the Technical Committee

on Operating Systems and Application Environments, IEEE Computer Society, Vol. 7, No. 2, pp. 35-42, June 1995.

[42] Fox, G. C., et. al., **Solving Problems on Concurrent Processors, Volume 1,** Prentice Hall, 1988.

[43] Fox, G. C., **The Architecture of Problems and Portable Parallel Software Systems,** Northeast Parallel Architectures Center, Technical Report SCCS-134, Syracuse University, July 1991.

[44] Fox, G. C., and Furmanski, W., *The Use of National Information Infrastructure and High Performance Computers in Industry*, Proceedings of the 2nd International Workshop on Massively Parallel Processing Using Optical Interconnections, October 1995.

[45] Fox, G. C., Furmanski, W., Chen, M., Rebbi, C., and Cowie, J. H., *WebWorks: Integrated Programming Environment Tools for National and Grand Challenges*, Northeast Parallel Architectures Center, Technical Report sccs-0715, June 1995.

[46] Fox, G. C., Hiranandani, H., Kennedy, K., Koelbel, C., Kremer, U., Tseng, C, and Wu, M., **Fortran D Language Specifications**, Northeast Parallel Architectures Center, Syracuse University, 1990.

[47] Garcia-Molina, H., and Spauster, A., *Ordered and Reliable Multicast Communication*, ACM Transactions on Computer Systems, Vol. 9, No. 3, pp. 242-271, August 1991.

[48] Geist, A., Beguelin, A., Dongarra, J., Weicheng, J., Manchek, R., and Sunderam, V., **PVM 3 User's Guide and Reference Manual**, Oak Ridge National Laboratory, Document No. ORNL/TM-12187, September 1994.

[49] Gelernter, D., and Carriero, C., *Coordination Languages and their Significance*, Communications of the ACM, Vol. 35, No. 2, pp. 97-107, February 1992.

[50] Ghafoor, A., and Ahmad, I., *An Efficient Model of Dynamic Task Scheduling for Distributed Systems*, Proceedings of the 14th Annual International Computer Software and Applications Conference, pp. 442-447, November 1990.

[51] Graham, I. S., **The HTML Sourcebook**, John Wiley & Sons Inc., 1995.

[52] Grimshaw, A. S., *The Mentat Runtime System: Support for Medium-Grain Parallel Computation*, Proceedings of the Fifth Distributed Memory Computing Conference, Los Alamitos, CA., Vol II, pp. 1064-1073, 1990.

[53] Grimshaw, A. S., *Easy-to-Use Object-Oriented Parallel Processing with Mentat*, Computer, Vol. 26, No. 5, May 1993.

[54] Hariri, S., *High Performance Distributed Systems: Concepts and Designs*, Artech House, in preparation.

[55] Hariri, S., Park, J., Yu, F., Parashar, M., and Fox, G., *A Message Passing Interface for Parallel and Distributed Computing*, Proceedings of the Second International Symposium on High Performance Distributed Computing, pp. 84-91, July 1993.

[56] Hariri, S., Yadav, R., Thiagarajan, B., Park, S. Y., Subramanyan, M., and Reddy, R., "A Concurrent Multi Target Tracker: Benchmarking and Portability", International Conference on Parallel Processing (ICPP), August 1994.

[57] Heidelberger, P., and Lavenberg, S. S., *Computer Performance Evaluation Methodology*, IEEE Transactions on Computers, Vol. 33, No. 12, pp. 1195-1220, December 1984.

[58] Hwang, K., and Briggs, F.A., **Computer Architecture and Parallel Processing**, McGraw-Hill, Second Edition.

[59] Jacob, B., **SOMobjects: A Pratical Introduction to SOM and DSOM**, IBM International Technical Support Centers, Document No. GG24-4357-00, July 1994.

[60] Ju, J., Xu, G., and Tao, J., *Parallel Computing Using Idle Workstations,* Operating Systems Review, Vol. 27, No. 3, 1993.

[61] Kaashock, M. F., and Tanenbaum, A. S., *Group Communication in the Amoeba Distributed Operating System*, Proceedings of the 11th International Conference on Distributed Computing Systems, pp. 222-230, May 1991.

[62] Kernighan, B. W., and Ritchie, M. R., **The C Programming Language - Second Edition**, Prentice Hall, 1988.

[63] Kramer, J., *Configuration Programming - A Framework for the Development of Distributable Systems,* The Proceedings of the IEEE International Conference on Computer Systems and Software Engineering (CompEuro 90), pp. 374-384, 1990.

[64] Kramer, J., Magee, J., and Finkelstein, A., *A Constructive Approach to the Design of Distributed Systems*, Proceedings of the 10th International Conference on Distributed Computing Systems, pp. 580-587, June 1990.

[65] Kramer, J., Magee, J., and Sloman, M., *Configuration Support for System Description, Construction and Evolution*, **Managing Complexity in Software Engineering**, pp. 247-260, edited by R. J. Mitchell, Peter Pereginus Ltd., 1990.

[66] Kremien, O., and Kramer, J., *Methodical Analysis of Adaptive Load Sharing Algorithms*, IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 6, pp. 747-760, November 1992.

[67] Krueger, P., and Cawla, R., *The Stealth Distributed Scheduler,* Proceedings of the 11th IEEE International Conference on Distributed Computing Systems, May 1991.

[68] Krueger, P., and Livny, M., *A Comparison of Preemptive and Non-Preemptive Load Distributing*, Proceedings of the 8th International Conference on Distributed Computing Systems, pp. 123-130, June 1988.

[69] Kunz, T., *The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme*, IEEE Transactions on Software Engineering, Vol. 17, No. 7, pp. 725-730, July 1991.

[70] Leffler, S. J., McKusick, M. K., Karels, M. J., and Quaterman, J. S., **The Design and Implementation of the 4.3BSD Unix Operating System**, Addison-Wesley, 1989.

[71] Litzkow, M. and Solomon, M., *Supporting Checkpointing and Process Migration Outside the Unix Kernel,* Proceedings of the Winter 1992 USENIX Conference, pp. 283-290, 1991.

[72] Liu, R.-C., *Analysis on the Effects of System Parameters on Load Balancing*, Information Sciences, Vol. 81, No. 1-2, pp. 37-54, November 1994.

[73] Lord, H. D., *Improving the Application Development Process with Modular Visualization Environments*, Computer Graphics, Vol. 29, No. 2, May 1995.

[74] Mechoso, C. R., Farrara, J. D., Spahr, J. A., *Running a Climate Model in a Heterogeneous, Distributed Computer Environment,* Proceedings of the Third International Symposium on High Performance Distributed Computing, pp. 79-84, August 1994.

[75] MacDougall, M. H., **Simulating Computer Systems - Techniques and Tools**, MIT Press, 1987.

[76] Mattson, T. G., *Programming Environments for Parallel Computing: A Comparison of CPS, Linda, P4, PVM, POSYBL, and TCGMSG*, Proceedings of the 27th Annual Hawaii International Conference on Systems Science, Vol. II, pp. 586-594, January 1994.

[77] Mirchandaney, R., Towsley, D., and Stankovic, J. A., *Analysis of the Effects of Delays on Load Sharing*, IEEE Transactions on Computers, Vol. 38, No. 11, pp. 1513-1525, November 1989.

[78] Mullender, S.J., and van Rossum, G., *Amoeba: A Distributed Operating System for the 1990s,* Computer, pp. 44-53, Vol. 23, No. 5, 1990.

[79] Munetomo, M., Takai, Y., and Sato, Y., *A Genetic Approach to Dynamic Load Balancing in a Distributed Computing System*, Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Vol. 1, pp. 418-421, June 1994.

[80] Mutka, M. W., *Estimating Capacity for Sharing in a Privately Owned Workstation Environment,* IEEE Transactions on Software Engineering, Vol. 18, No. 4, 1992.

[81] Mutka, M. W., and Livny, M., *The Available Capacity of a Privately Owned Workstation Environment,* Performance Evaluation, pp. 269-284, Vol. 12, No. 4, July 1991.

[82] The Object Management Group, **The Common Object Request Broker: Architecutre and Specification**, OMG Document Number 93.12.43, December 1993.

[83] Parasoft Corporation, *Express Reference Manual,* 1988.

[84] Park, C., and Kuhl, J. G., *A Fuzzy-Based Distributed Load Balancing Algorithm for Large Distributed Systems*, Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems, pp. 366-273, April 1995.

[85] Peterson, L., Hutchinson, N., O'Malley, S., and Rao, H., *The x-Kernel: A Platform for Accessing Internet Resources,* Computer, pp. 23-33, Vol. 23, No. 5, May 1990.

[86] Prieto-Diaz, R., and Neighbors, J.M., *Module Interconnection Languages,* The Journal of Systems and Software, No. 6, pp. 307-334, 1986.

[87] Rehmet, G. M., **Remora - Implementing Adaptive Parallelism on a Heterogeneous Cluster of Networked Workstations**, Master's Thesis, Rhodes University, South Africa, September 1994.

[88] Rinaldo, F., and Wolbers, S., *Loosely Coupled Parallel Processing at Fermilab*, Computers in Physics, Vol. 7. No. 2, pp. 184-190, March 1993.

[89] Rosenberry, W., Kenney, D., and Fisher, G., **Understanding DCE**, O'Reily & Associates, 1992.

[90] Rousselle, P., Hariri, S., and Tymann, P, *Redundant Task Scheduling in a Heterogeneous Computing Environment*, Northeast Parallel Architectures Center, Technical Report SCCS-658, October 1994.

[91] Rousselle, P., Tymann, P., Hariri, S., and Fox, G., *The Virtual Computing Environment* Proceedings of the Third International Symposium on High Performance Distributed Computing, San Francisco, California, pp. 7-14, August 1994.

[92] Rousselle, P., Yadav, R., Hariri, S., Kim, D., Boddapati, S., and Tymann, P., *Application Development and Execution in a Virtual Computing Environment*, Northeast Parallel Architectures Center, Technical Report SCCS-705, April 1995.

[93] Schaar, M., Efe, K., and Delcambre, L. M., *An Analytical Model for Predicting Performance in a Heterogeneous System*, Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Computing Systems, pp. 328-335, May 1991.

[94] Shoja, G.C., *A Distributed Facility for Load Sharing and Parallel Processing Among Workstations,* Journal of Systems Software, pp. 163-172, Vol. 14, No. 3, 1991.

[95] Simpson, R. A., and Harget, A. J., *A Simulation Study to Determine the Importance of Load Balancing Algorithms for Loosely Coupled Distributed Systems*, IFIP Transactions on Computer Science and Technology, IFIP 13th World Computer Congress, Vol. A-51, pp. 69-74, August 1994.

[96] Stevens, W. R., **UNIX Network Programming**, Prentice Hall, 1990.

[97] Stroustrup, B., **The C++ Programming Language - Second Edition**, Addison-Wesley, 1991.

[98] Stroustrup, B., **The Design and Evolution of C++**, Addison-Wesley, 1994.

[99] V. S. Sunderam, **PVM: A Framework for Parallel Distributed Computing**, Oak Ridge National Laboratory, 1991.

[100] Tannenbaum, A. S., **Modern Operating Systems**, Prentice-Hall, 1992.

[101] Theimer, M., and Lantz, K., *Finding Idle Machines in a Workstation-Based Distributed System,* IEEE Transactions on Software Engineering, pp. 1444-1458, Vol. 15, No. 11, 1989.

[102] Theimer, M., and Hayes, B., *Heterogeneous Process Migration by Recompilation*, Proceedings of the 11th IEEE International Conference on Distributed Computing Systems, May 1991.

[103] Thornborrow, C., Wilson, A. J. S., and Faigle, C., *Developing Modular Application Builders to Exploit MIMD Parallel Resources*, Proceedings of Visualization '93, pp. 134-141, San Jose CA, October 1993.

[104] Waldspurger, C.A., Hogg, T., Huberman, B.A., Kephart, J.O., and Stornetta, W.S., *Spawn: a distributed computational economy,* IEEE Transactions on Software Engineering, pp. 103-117, Vol.18, No.2, 1992.

[105] Walker, B., Popek, G., English, R., Kline, C., and Thiel, G., *The LOCUS Distributed Operating System,* Proceedings of the 1983 SIGOPS Conference, pp. 49-70.

[106] Wang, C. -J., Krueger, P. E., and Liu, M. T., *Intelligent Job Selection for Distributed Scheduling,* Proceedings of the Thirteenth International Conference on Distributed Computing Systems, pp. 517-524, May 1993.

[107] Wood, M. D., *Replicated RPC Using Amoeba Closed Group Communication*, Proceedings of the 13th International Conference on Distributed Computing Systems, pp. 222-230, 1993.

[108] Zhou, S., *A Trace-Driven Study of Dynamic Load Balancing*, IEEE Transactions on Software Engineering, Vol. 14, No. 9, pp. 1327-1341, September 1988.

[109] Zhou, S., Zheng, X., Wang, J. and Delisle, P., *Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems*, Software - Practice and Experience, pp. 1305-1336, Vol. 23, No. 12, December 1993.

[110] Zhu, W., and Goscinski, A., *Low Cost Load Balancing Algorithms*, Proceedings of the 27th Annual Simulation Symposium, pp. 226-235, April 1994.

Biographical Data

Name:	Philip Rousselle
Date and Place of Birth:	September 10, 1962 Buffalo, New York
Elementary & High School:	Buffalo Public Elementary School #84 Graduated 1976
	Kensington High School Buffalo, New York Graduated 1980
College:	SUNY College at Buffalo BS (Info Systems Management), 1983
	Union College Schnectady, New York MS (Computer Science), 1989
Employment	International Business Machines Inc Endicott NY