Distributed System – A Cluster Computing Environment

P-719

PROJECT REPORT

SUBMITTED BY

A. Mahesh (0037Q0036)

GUIDED BY

Mr. S. GANESH BABU, M.C.A
Lecturer.

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF
MASTER OF SCIENCE IN
APPLIED SCIENCE-COMPUTER TECHNOLOGY
OF THE BHARATHIAR UNIVERSITY,
COIMBATORE

2001-2002

Department of Computer Science and Engineering
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641006

Kumaraguru College of Technology

Coimbatore-641006

Department of computer Science & Engineering

Certificate

This to certify that the project report entitled

DISTRIBUTED SYSTEM – A CLUSTERCOMPUTING ENVIRONMENT

Has been submitted by
A. MAHESH,

In partial fulfillment of the requirements for the Award of degree of Master Of Science (Applied Sciences Computer Technology) of the Bharathiar University, Coimbatore-46 during the year 2001-2002.

[illegible]
(Guide)
(S. GANESH BABU)

[illegible]
(Head Of Department)

Certified that the candidate was examined by us in the project viva-voce examination held on [illegible] & the University Register number is. 0037Q0036

[illegible]
(Internal Examiner)
(R. Dinesh, SL/CSE, KCT)

[illegible]
(External Examiner)

DECLARATION

I hereby declare that this project entitled

DISTRIBUTED SYSTEM – A CLUSTER COMPUTING ENVIRONMENT

is submitted in partial fulfillment of the requirement for the award of the degree of M.Sc [Applied Science Computer Technology] is the report of the original work done by me during the period of study (2001-2002) in

KUMARAGURU COLLEGE OF TECHNOLOGY
CHINNAVEDAMPATTI
COIMBATORE

Under the supervision of

Mr. S. Ganesh Babu M.C.A,
Lecturer
(Computer Science And Engineering)

Name	Register Number	Signature
A. MAHESH	**0037Q0036**	

Place : **Coimbatore**
Date :

ent Technologies India [illegible]
s Complex,
Annamalai [illegible]
[illegible]
[illegible]
[illegible]
[illegible]

April 4, 2002

<u>To whomsoever it may concern:</u>

This is to certify that the dissertation entitled "**DISTRIBUTED SYSTEM**", submitted by Mr. A. **MAHESH** student of Department of Computer Science and Engineering, Kumarguru College of Technology, Coimbatore, as part of his M.Sc. – CT final semester project work, in partial fulfillment of the degree in M.Sc. (Computer Technology) has been successfully carried out and completed under the guidance of Ms. M. Kavitha (MCA), PGDCA, (Programmer) Profluent Technologies India Pvt. Limited, Coimbatore.

The duration of the project work was from Jan – 2002 to April 2002. The matter presented in this dissertation has not been submitted for any other degree. In order to maintain confidentiality, the source code has been withheld.

For PROFLUENT TECHNOLOGIES INDIA PVT LTD.,

[illegible]

R.Balakrishnan
Manager - Projects

ACKNOWLEDGEMENT

I express my deep sense of gratitude to our principal **Dr. K. K. Padmanabhan B.Sc (Engg,) M.Tech, Ph.D.,** for having provided necessary facilities for the successful completion of my project.

I also extend my sincere thanks to **Dr. S. Thangasamy Ph.D., Head of the Department**, for the help rendered by him to complete our project successfully.

I give my immense pleasure to express heartfelt thanks to my guide **Mr. S. Ganesh Babu M.C.A, Lecturer** encouragement and valuable suggestions to make this project a successful one.

I would like to thank my external guide **Ms. M. Kavitha M.C.A, Programmer** for giving her valuable suggestions to make this project.

Last but not least I extend my heartfelt thanks to all the faculty, friends and well wishers who helped me in completing this project work.

SYNOPSIS

The project work titled as "***DISTRIBUTED SYSTEM - CLUSTER COMPUTING ENVIRONMENT***" is to developed for Profluent Technologies, Coimbatore. The project aims in creating a parallel-computing environment for the users to develop parallel programs.

The project extends *Parallel Processing* for *Cluster of Workstations*. Parallel processing is the technology behind current high performance computers. The world's fastest supercomputer as of now is based on parallel processing. This project attempts to highlight parallel computing system based on *Commodity Parallel Processing*. That is, it's a parallel and distributed processing system that is designed and built, with readily available components and without resorting to specialized hardware circuitry. With the continuous fall in prices and increase in computing power of desktop machines, one can at present build a very good parallel computing system with the power of multiple stock PCs. Several such systems have been built and have met with good success. Most of them need some form of specialized hardware to work. "Beowulf" is one such well-known cluster computing system, which met with splendid success. We present complete software only approach that can be used on a wide range of problems. This system would be a viable cost effective alternative to supercomputers with better price-performance ratio where computing power is of great necessity but is seldom available due to high costs (like in Engineering colleges).

The objective of the CCE system is to provide a parallel programming environment on Cluster of Workstations running Windows and connected via an interconnection network. The CCE enables the application programmer to conceive the network of workstations as a single large virtual machine rather than as a set of independent systems. Extensive use of modern programming methodologies and careful optimization has been done to ameliorate the performance of the system.

CONTENTS

PAGE NO

1. INTRODUCTION

1.1 Project Overview 1

1.2 About the Organization 3

2. REQUIREMENT ANALYSIS

2.1 Problem Definition 7

2.2 Objective 10

2.3 Existing System 11

2.4 Proposed System 12

2.5 User Characteristics 13

3. SYSTEM ANALYSIS

3.1 System Requirements 14

3.1.1 Hardware Requirements 14

3.1.2 Software Requirements 14

3.1.3 Functional Requirements 14

3.1.4 Usability Requirements 15

3.2 Scope of the system 16

3.4 System Modeling 16

4. PROGRAMMING ENVIRONMENT

4.1 Development Environment 20

4.2 Implementation Environment 20

PAGE NO.

5. SYSTEM DESIGN AND DEVELOPMENT

5.1 Introduction 21

5.2 System Flow Diagram 22

5.3 System Development Languages &Tools 25

6. **CONCLUSION** 29

7. **SCOPE FOR FUTURE DEVELOPMENT** 30

8. APPENDIX

8.1 Basic Programming Techniques 31

8.2 API Manual 35

8.3 Message Description 43

8.4 Sample flow 50

9. BIBLIOGRAPHY 57

Introduction

1.1 Project Overview

The project entitled "*Distributed System – A Cluster Computing Environment*" is developed to create a parallel programming environment for the users to develop parallel programs. It consists of the following programs.

Manager:

The manager is program is one, which is started up by the master program during the initialization of CCE. The manager program runs in the local system where the master program is present. The manager program control overall activities of the system. Manager can be described as the "heart of CCE". The function calls made by the master program send signals to the manager. The manager interprets the signals and executes an appropriate routine for that signal. The manager acts as a mediator between the master, agent and worker programs. All the requests from these programs are sent to the manager, where the manager will take the effective ways of action.

The manager maintains data structures for storing the node list, task port information. The node list consists of the IP address of the node no and a unique number identifying that node.

The port information list consists of the port number on which the task is listening, task number of that task and the IP address of the system in which the task is currently executed.

Agent:

The agent is a deamon process in every node. The agent program listens for the requests from the manager to that specific node. The agent program is responsible for carrying out a spawn request from the manager.

The agent program maintains a list for tasks that are currently executing in the system. The task list contains details such as the task id of that task and the process id of the process. Once a task is spawned to the agent the agent makes entries into the task list. Once the task exits the entry for the task is deleted from the task list.

Master:

The master program is usually written by the user. This is the program, which contains the function calls to CCE environment. The master program will contain the name of the worker program and the number of copies to be spawned. The master program is written in "C" language syntax. This program must include the "cce.h" header file. The program is compiled using a "C" compiler.

Worker:

The worker program is written by the user. This program will contain the routines of the operations to be performed by the task and function calls to the CCE environment. The worker program is also written in "c" language syntax. This program must include the "cce.h" header file. The program is compiled using a "C" compiler.

1.2 ORGANIZATION PROFILE

Profluent Technologies India Private Limited is the result of a visio cherished by a group of young entrepreneurs with the zeal to establish a successfu business in information technology.

Profluent Technologies provides authentic IT solutions to its clients with its plethora of skills in Cutting Edge Technologies. The strengths of the company are its people; 76 professionals from diverse technical backgrounds - all converged to promote a synergy hitherto unseen.

A dedicated R&D division ensures that the company toes the line with all innovations in the industry. Rigid quality control methods ensures timely delivery of quality software. An agile support team augments the services provided by the company.

Profluent Technologies strives to be a responsible market leader while also contributing immensely to the expectations of its shareholders, growth of its employees and well being of society.

The company operates primarily from its 8000 sq. ft. facility at 'The Uffizi' - a prestigious landmark in the city with excellent facilities such as standby generators, multiple elevators and aesthetic interiors. It is located within three kilometers from the airport.

The development unit has an advanced local area network with equipment from brands like IBM, Intel, HP etc. High-speed data transmission facilities are available to cater to the needs of overseas clients.

Services

Application Development

Custom solutions for all types of business, utilizing superior systems analysi and design techniques , ERP solutions for medium and large enterprises , redesig and conversion of existing systems.

Internet Solutions

Profluent Technologies is your one-stop solution provider to put your busines on the Internet. We provide Internet solutions that enable organizations an business systems to improve information exchange and communication b connecting together vendors, customers, suppliers and distributors over the web.

Web Site Design

Web sites designed by Profluent Technologies are fast, browser independent visually appealing, and easy to navigate features that are essential to capture an retain your target audience. The web design team has immense experience on th following key elements of website designing:

- HTML coding
- Web graphics design
- Template and macros design
- Interactive navigational devices
- Synchronization with web editing tools

Web Content Development & Integration

We will develop and apply the needed content in an appropriate manner to best serve the needs and interests of the diverse website community. Work in this area covers:

- Conversion of legacy documents to HTML
- HTML template integration
- Forms and database integration
- Security

Web Application Development

We provide web based enterprise solutions to bind disparate systems into a single environment that allows organizations to effectively integrate their internal and external Services. Transactional integrity, scalability and requisite security are some of the salient features of our solutions.

Portal Development

We specialize in the development of various kinds of portals and EDI solutions that will enable

- Companies to advertise their products
- Clients and dealers to meet virtually
- Online Business and financial transactions

Long term Website Support and Maintainance

- Web site and web server administration
- Content translation and updation
- Search engine maintenance
- Internal and external link management
- Web traffic statistics and analysis

Facilities Management

Facilities management is a service package offered on an annual basis fo managing single or multiple sites. The package includes:

- Network study, designing, analysis and services on implementatio methods
- All server, client, database, peripheral integration with a variety of LAN operating platforms
- Host / remote connectivity configuration services
- Seamless integration of all network platforms

Requirement Analysis

2. *Requirement Analysis*

2.1 Problem Definition

Parallel and Distributed computing:

Parallel processing, the method of having many small tasks solve one large problem, has emerged as a key enabling technology in modern computing. The past several years have witnessed an ever-increasing acceptance and adoption of parallel processing, both for high-performance scientific computing and for more ``general-purpose" applications, was a result of the demand for higher performance, lower cost, and sustained productivity. The acceptance has been facilitated by two major developments: massively parallel processors (MPPs) and the widespread use of distributed computing.

MPPs are now the most powerful computers in the world. These machines combine a few hundred to a few thousand CPUs in a single large cabinet connected to hundreds of gigabytes of memory. MPPs offer enormous computational power and are used to solve computational Grand Challenge problems such as global climate modeling and drug design. As simulations become more realistic, the computational power required to produce them grows rapidly. Thus, researchers on the cutting edge turn to MPPs and parallel processing in order to get the most computational power possible.

The second major development affecting scientific problem solving i *distributed computing*. Distributed computing is a process whereby a set o computers connected by a network is used collectively to solve a single larg problem. As more and more organizations have high-speed local area network interconnecting many general-purpose workstations, the combined computationa resources may exceed the power of a single high-performance computer. In some cases, several MPPs have been combined using distributed computing to produce unequaled computational power.

The most important factor in distributed computing is cost. Large MPPs typically cost more than $10 million. In contrast, users see very little cost in running their problems on a local set of existing computers. It is uncommon for distributed-computing users to realize the raw computational power of a large MPP, but they are able to solve problems several times larger than they could by using one of their local computers.

Common between distributed computing and MPP is the notion of message passing. In all parallel processing, data must be exchanged between cooperating tasks. Several paradigms have been tried including shared memory, parallelizing compilers, and message passing. The message-passing model has become the paradigm of choice, from the perspective of the number and variety of multiprocessors that support it, as well as in terms of applications, languages, and software systems that use it.

Cluster parallel processing:

Clusters are currently both the most popular and the most varied approach, ranging from a conventional network of workstations (NOW) to essentially custom parallel machines. It offers several important advantages:

- Each of the machines in a cluster can be a complete system, usable for a wid range of other computing applications. This leads many people to suggest tha cluster parallel computing can simply claim all the "wasted cycles" o workstations sitting idle on people's desks. It is not really so easy to salvage those cycles, and it will probably slow the co-worker's screen saver, but it car be done.

- The current explosion in networked systems means that most of the hardware for building a cluster is being sold in high volume, with correspondingly low "commodity" prices as the result. In comparison, SMP and attached processors are much smaller markets, tending toward somewhat higher price per unit performance.

- Cluster computing can *scale to very large systems*. While it is currently hard to find a Windows/Linux-compatible SMP with many more than four processors, most commonly available network hardware easily builds a cluster with up to 16 machines. With a little work, hundreds or even thousands of machines can be networked. In fact, the entire Internet can be viewed as one truly huge cluster.

- The fact that replacing a "bad machine" within a cluster is trivial compared to fixing a partly faulty SMP. This becomes important not only for particular applications that cannot tolerate significant service interruptions, but also for general use of systems containing enough processors so that single-machine failures are fairly common.

As in most cases, there are a few disadvantages in employing "clusters" for paralle processing. They are

- With a few exceptions, network hardware is not designed for paralle processing. Typically latency is very high and bandwidth relatively lov compared to SMP and attached processors. For example, SMP latency i: generally no more than a few microseconds, but is commonly hundreds o: thousands of microseconds for a cluster. SMP communication bandwidth i: often more than 1000 Mbytes/second; although the fastest network hardware (e.g., "Gigabit Ethernet") offers comparable speed, the most commonly usec networks are between 10 and 1000 times slower.

- There is very little software support for treating a cluster as a single system.

Thus, the basic story is that clusters offer great potential, but that potential may be very difficult to achieve for most applications. The good news is that there is a lot of research currently happening in this area. There are also networks designed specifically to widen the range of programs that can achieve good performance on "Clusters".

2.2 Objective:

The CCE system is intended to provide a parallel programming environment on Cluster of Workstations connected via an interconnection network. The APIs provided by the CCE system enables the application programmer to develop parallel applications that execute on the cluster as a whole rather than on an independent workstation in the cluster. It facilitates the application developer to concentrate on the application specific tasks in developing a parallel application, rather than on the

implementation of common parallel processing primitives on clusters (like messag
passing, task control, load balancing, etc.).

2.3 Existing System:

There are already a few existing systems which provides a similar kind o
functionality, each having its own advantages and limitations. The most commor
ones are PVM (Parallel Virtual Machine) and MPI (Message Passing Interface).
Though these are currently the most widely used software for parallel programming
on *Clusters*, the overheads caused by these are comparatively high. So many
developers prefer to implement customized parallel programming primitives through
socket programming. The disadvantage with socket programming is that, apart from
the need for developers to have sufficient knowledge in network programming, the
size of the project increases as all the primitives here becomes a part of the
application. This implies that the application may become specific to a type of
cluster, and may not be portable even between similar types of clusters. Apart from
that, the duration of the project increases along with the cost, and the maintenance
effort also accrues.

Although PVM rather quickly became a de facto standard, critics pointed out
that in actuality there was no formal, enforced standard. It was also clear that the
message passing was slower than optimized protocols native to an architecture. Very
soon after the spread of PVM, a standard was put forward called MPI, the Message-
Passing Interface. MPI has advantages (e.g., faster message passing, a standard) and
disadvantages (e.g., not interoperable among heterogeneous architectures, not
dynamically reconfigurable) with respect to PVM.

2.4 Proposed System:

The CCE software provides a unified framework within which paralle programs can be developed in an efficient and straightforward manner using existin hardware. CCE enables a collection of networked computer systems to be viewed a a single parallel virtual machine. CCE transparently handles all message routing task scheduling, etc. across the Cluster.

The CCE computing model is simple yet very general, and accommodates wide variety of application program structures. The programming interface i deliberately straightforward, thus permitting simple program structures to b implemented in an intuitive manner. The user writes his application as a collectio of cooperating *tasks*. Tasks access CCE resources through a library of standar interface routines. These routines allow the initiation and termination of tasks acros the network as well as communication and synchronization between tasks.

CCE tasks may possess arbitrary control and dependency structures. In othe words, at any point in the execution of a concurrent application, any task i existence may start or stop other tasks or add or delete computers from the virtua environment. Any process may communicate and/or synchronize with any other Any specific control and dependency structure may be implemented under the CC system by appropriate use of CCE constructs and host language control-flow statements.

Care has been taken to ensure that the overhead of using this CCE system is relatively low. It is also optimized for the prudent usage of various resources.

2.5 User Characteristics

The user needs to know how to write a parallel program for his application. The user can learn how to write a parallel program by studying the manual for the software. The efficiency of this project depends only on the user. The language syntax to be used in the programs is 'C/C++'.

System Analysis

3. System Analysis

3.1 System Requirements:

3.1.1 Hardware Requirements:

- Network enabled Workstations with Intel Pentium(or its clones) capable of running Windows
- Bus/Star based interconnection network

3.1.2 Software Requirements:

- Windows95 or higher
- 'VC++' compiler for Windows platform

3.1.3 Functionality Requirements

- Customized host pool: The application's computational tasks execute on a set of machines that are selected by the user for a given run of the CCE program. The host pool may be modified by adding and/or deleting machines dynamically during operation.

- Job Scheduling & Load balancing: The CCE system is capable of finding the most appropriate host from the host pool for scheduling a task based on various factors like CPU load, Memory Utilization, etc.

- Task Control Mechanisms: The CCE system provides mechanisms for spawning a task on a local/remote host, killing a task, signaling a task, etc. apart from other message passing routines.

- Message Passing Mechanisms: Various message-passing primitives are implemented in this CCE system. These routines follow a synchronous mode of communication.

3.1.4 Usability Requirements

The APIs provided by the CCE system adheres to the Visual C++ standard, and they are expected to be used by application developers in context of developing parallel applications using Visual C++ language. These APIs and their naming conventions are similar to the ones provided by PVM and MPI packages, and the developers familiar with these can readily use this system. Even programmers inexperienced with the above packages can use this system with very little effort. Sufficient help manuals and documentation are provided along with the system for all kinds of users to accommodate with ease.

Apart from the knowledge and experience in the host language (i.e. C/C++), the programmers using this system are expected to have sufficient foundation in parallel programming and distributed computing systems.

3.2 Scope of the System:

The scope of this system is to provide an environment for developing an executing parallel applications on Cluster of Workstations. As the interface provided by this CCE system are confined to the Visual C++ language, the hos language that can be used for the development of parallel applications could only be either C or C++. And they are also restricted to the workstations running Windows as its OS.

Though the CCE system aids in building parallel applications on clusters, it follows an explicit message-passing model, and is not an automated tool to detect inherent parallelism in an application program. It is up to the user to detect and tap the concurrency exhibited by an application.

3.4 System Modeling:

CCE is an integrated set of software tools and libraries that emulates a general-purpose, flexible, concurrent computing framework on interconnected workstations. The overall objective of the CCE system is to enable such a collection of computers to be used cooperatively for concurrent or parallel computation. Detailed descriptions and discussions of the concepts, logistics, and methodologies involved in this network-based computing process are contained in the appendices. Briefly, the principles upon which CCE is based include the following:

- User configured host pool: The application's computational tasks execute on a set of machines that are selected by the user for a given run of the CCE program or it can be generated by the software. The host pool may be altered by adding and deleting machines during dynamically operation

- Process-based computation: The unit of parallelism in CCE is *task*, a independent sequential thread of control, which partakes in bot communication and computation.

- Explicit message-passing model: This collection of tasks cooperates b explicitly sending and receiving message to one another. There is no languag dependent restriction on the size of any messages.

The CCE system is composed of two parts. The first part is an *agent*, that resides on all the computers making up the virtual environment. The second part of the system is the *Manager* that should reside in the node where the user initiates the *Master* task. It contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This CCE library, that are linked with the user applications contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine.

The CCE computing model is based on the notion that an application consists of several tasks. Each task is responsible for a part of the application's computational workload. Sometimes an application is parallelized along its functions; that is, each task performs a different function, for example, input, problem setup, solution, output, and display. This process is often called functional parallelism. A more common method of parallelizing an application is called data parallelism. In this method all the tasks are the same, but each one only knows and solves a small part of the data. This is also referred to as the SPMD (Single-Program Multiple-Data) model of computing. CCE supports either or a mixture of these methods. Depending on their functions, tasks may execute in parallel and may need to synchronize or exchange data, although this is not always the case. An architectural view of the CCE system is shown in Figure

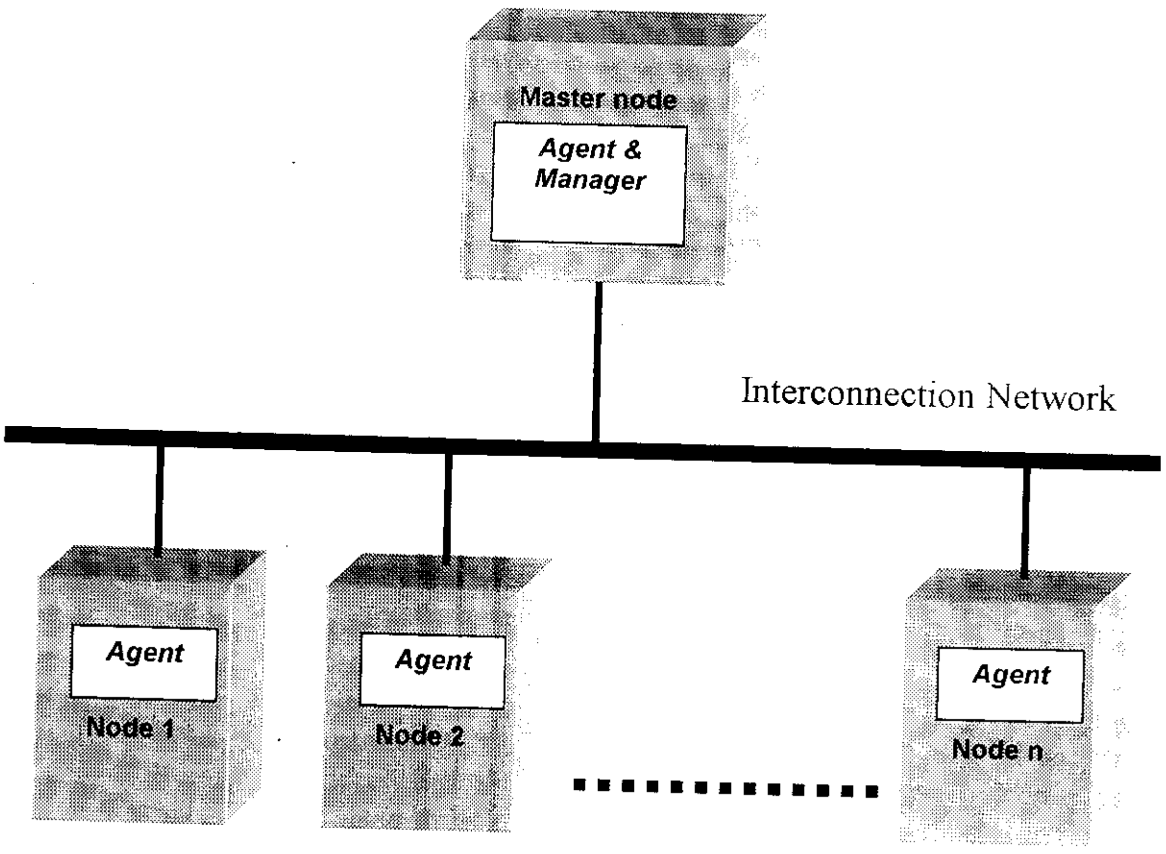


Figure CCE Architectural Overview

The CCE system currently supports C and C++ languages. This set of language interfaces have been included based on the observation that the predominant majority of target applications are written in C, with an emerging trend in experimenting with object-based languages and methodologies.

The C and C++ language bindings for the CCE user interface library ar
implemented as functions, following the general conventions used by most C
systems. To elaborate, function arguments are a combination of value parameter
and pointers as appropriate, and function result values indicate the outcome of th
call. All CCE tasks are identified by an integer *Task Identifier* (TID). Messages ar
sent to and received from tid's. Since tid's must be unique across the entire virtua
environment, they are supplied by the local *Agent* and are not user chosen. Althoug
CCE encodes information into each TID, the user is expected to treat the tid's a
opaque integer identifiers. CCE contains several routines that return TID values s
that the user application can identify other tasks in the system.

The general paradigm for application programming with CCE is as follows. A
user writes one or more sequential programs in C or C++ that contain embedded
calls to the CCE library. Each program corresponds to a task making up the
application. These programs are compiled, and the resulting object files are placed at
a location accessible from machines in the host pool. To execute an application, a
user typically starts one copy of one task (usually the ``master" or ``initiating" task)
by hand from a machine within the host pool. This process subsequently starts other
CCE tasks, eventually resulting in a collection of active tasks that then compute
locally and exchange messages with each other to solve the problem. As mentioned
earlier, tasks interact through explicit message passing, identifying each other with a
system-assigned, opaque TID.

Programming Environment

4. PROGRAMMING ENVIRONMENT

4.1 System Environment:

4.1.1 Development Environment:

The CCE system has been developed using Visual C++ language on Windows platform. The configuration of the systems used was 64MB RAM and Pentium II Processor.

4.2 Implementation Environment:

The APIs provided by the CCE system adheres to Visual C++ compilers can be used to compile the programs embedding CCE interfaces. Though appropriate CCE header files/libraries need to be included, no special parameters need to be passed for compilation.

System Design & Development

5. SYSTEM DESIGN

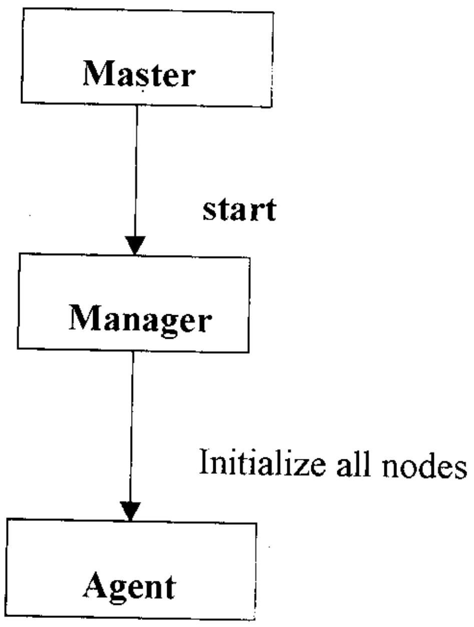
5.1 Introduction

The most creative and challenging phase of the system life cycle is system design. The term design describes a final system and the process by which it is developed. It refers to the technical specification that will be applied in implementing the candidate system. It also includes the construction of programs and program testing. The key question here is: how should the problem be solved?

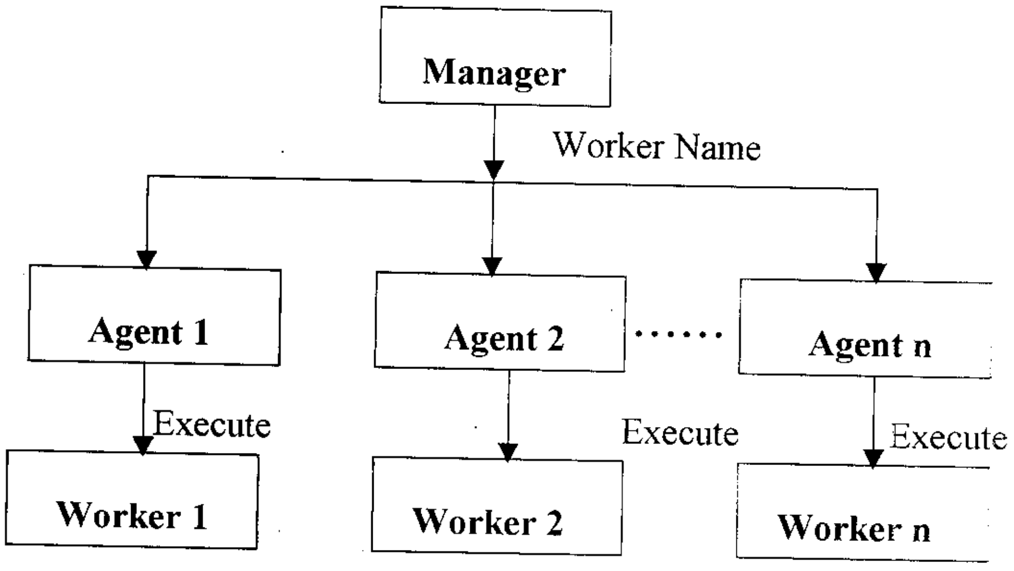
System Design is a solution, a "how to" approach to the creation of a new system. This important phase is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study. Emphasis is on translating the functional requirements into design specification. Design goes through logical and physical stages of development. Logical screen reviews the present physical system; prepares input and output specification; makes edit, security and control specification; details the implementation plan; and prepares a logical design walkthrough. The physical design maps the details of the physical system, plans the system implementation, devises a test and implementation plan and specifies any new hardware or software.

5.2 System Flow Diagram

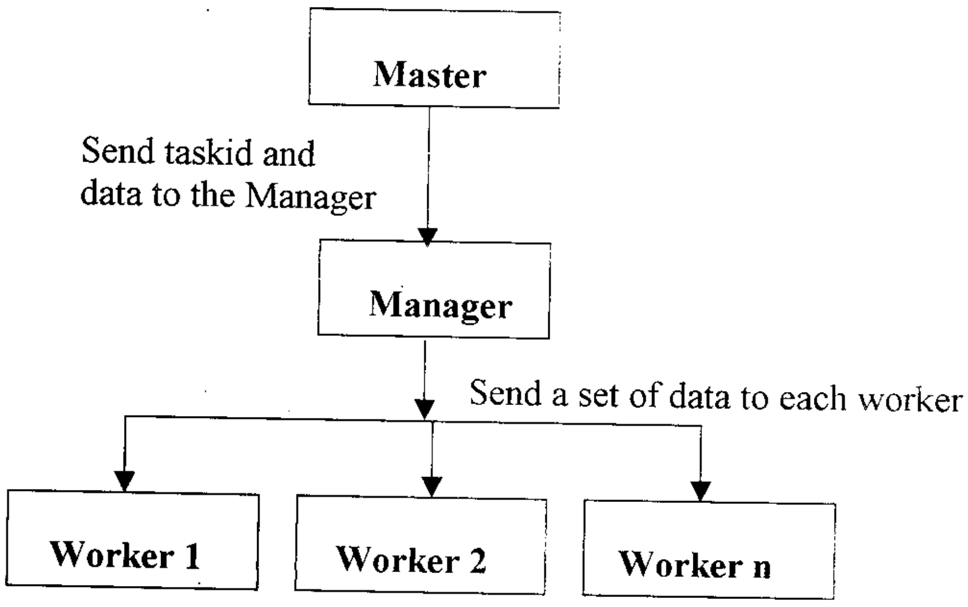
Initialization:



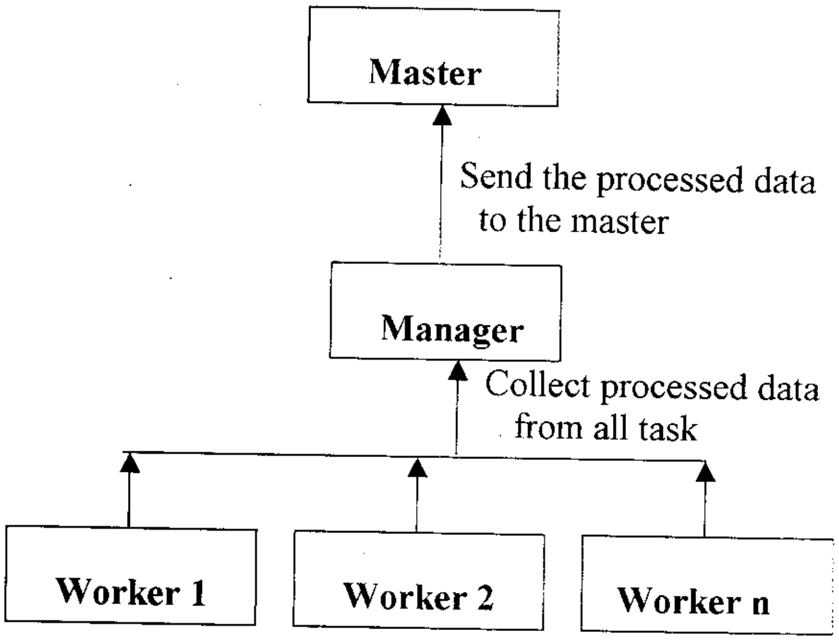
Spawn:



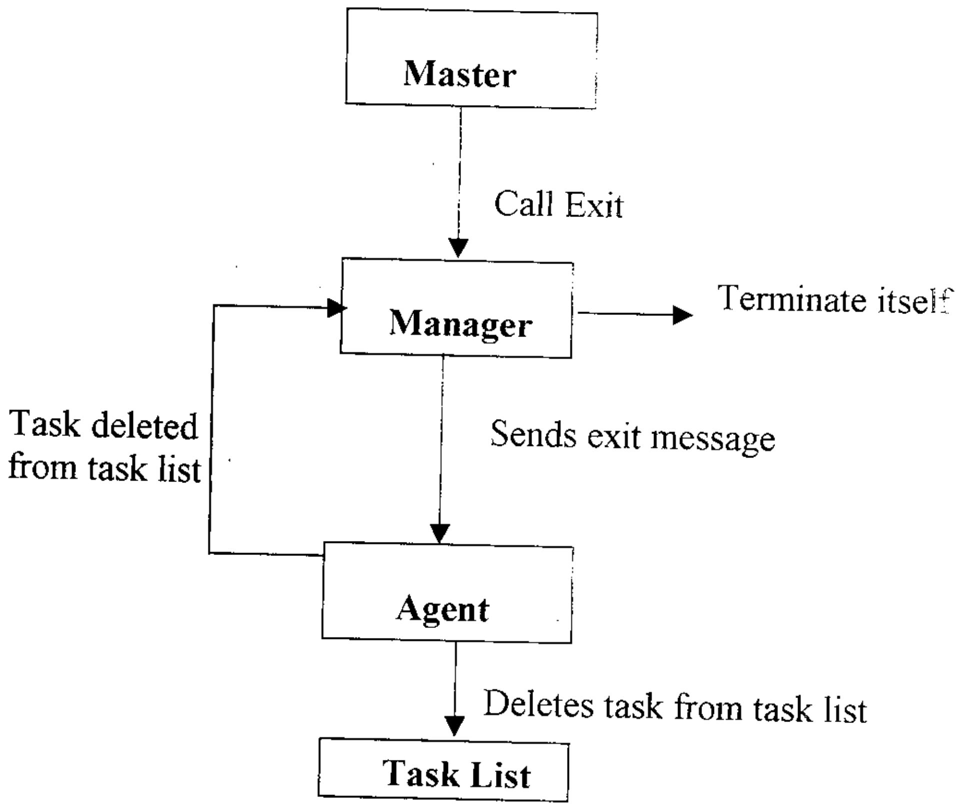
Send:


Master
Send taskid and data to the Manager
Manager
Send a set of data to each worker
Worker 1
Worker 2
Worker n

Receive:


Master
Send the processed data to the master
Manager
Collect processed data from all task
Worker 1
Worker 2
Worker n

Exit:


Master
Call Exit
Manager
Terminate itself
Task deleted from task list
Sends exit message
Agent
Deletes task from task list
Task List

5.3 System Development Tools

Visual C++

The Visual C++ programming language and environment is designed tc solve a number of problems in modern programming practice. Visual C++ is simple object oriented, and network savvy, interpreted, secure, architecture neutral portable, dynamic, multithreaded language.

Simple

Visual C++ is one of most essential tool for developing for more complicated. The Visual C++ is dividing in to two major parts. They are SDK (Software Development Kit) and DDK (Driver Development Kit). The Software Development kit consist of Dynamic Link Libraries, that are linked & called to a program is called Software Development Kit. The Driver Development Kit (DDK) this also set built in functions & Dynamic Link Libraries but used for Designing Driver for hardware or Devices.

WinSock

WinSock is a network application programming interface(API) for Microsoft Windows – a well defined set of data structures and functions calls implemented as a dynamic link library(DLL). WinSock is a preferred interface for accessing a variety of underlying network protocols and is available in varying forms on every win32 platform. The WinSock interface inherits a great deal from Berkeley(BSD) sockets implementation on UNIX platforms, which is capable of accessing multiple networking protocols. In Win32 environments, the interface has finally evolved int a truly protocol-independent interface, especially with the release of WinSock 2.

Most of the existing WinSock software was developed for the Internet. Eve though WinSock was originally developed primarily for TCP/IP, the API is abstrac enough to support other protocol families. With WinSock2 comes a substantial nev functionality. It formally supports IPX/SPX, DecNet, and OSI and expands the AP to allow other protocols to be hooked in at a later date. WinSock 2 applications ca even be entirely protocol independent. Using WinSock 2 services, an applicatio that is moved from one environment to another can adapt to differences in networ addressing and naming schemes.

WinSock has been extremely successful since it was released in 1992. as customer demands for interconnectivity grew, providers of network hardware and software began to realize the necessity of cooperation and open standards. Closed, proprietary standards started to lose ground in the marketplace. When heavyweights such as Microsoft, Novell, IBM, Sun, 3com, Hewlett-Packard, DEC, and others joined WinSock group, WinSock's acceptance was virtually assured.

Microsoft was a founding member of WinSock group and has since changed its network communications strategy from NetBIOS to WinSock. The Windows Open Services Architectures (WOSA) incorporates WinSock as a part of Microsoft suite of open APIs.

All WinSock programs follow then general client/server design. There are two fundamental types of client/server application pairs: connection-oriented and connectionless applications. Whether an application is connection-oriented or

connectionless is usually determined by the protocols used by the application t
communicate and amount of data to be exchanged.

For example, programs such as e-mail and file transfer that transmit larg
amounts of data are usually connection-oriented. Programs that transmit smal
amounts of data, such as daytime and echo applications are generally connectionless.

Connection-Oriented Applications

Applications that send and receive data as a stream of bytes are connection
oriented. A connection is established between the two process before any data i
transmitted or received. The identity of each endpoint(socket) is established onl
once, at the beginning of the conversation. From that point on, the origin an
destination of data sent and received are understood to come from and to the othe
end of data sent and received are understood to come from and go to the other end o
then established connection. The applications do not identify themselves again eac
time data is transmitted.

In the internet protocol suite, connection-oriented applications are usually tha
rely on Transmission Control Protocol(TCP) for transport.

Connectionless Applications:

Applications that send and receive data in datagrams are typicall
connectionless. The identity of each endpoint in the conversation is established eac
time data is is sent. Then scheme is only recommended for applications tha
transmit small amounts of information.

In the internet protocol suite, then usually corresponds to applications that us
the User Datagram Protocol (UDP).

Initializing WINSOCK

WinSock includes some functions that werent part of the original Berkeley API. Most were added to make applications more windows-friendly, and their uise is highly recommended. All then extension functions, are prefixed with WSA WHICH STANDS FOR Windows Sockets API. Every WinSock application must load the appropriate version of the WinSock. DLL. If you fail to load the WinSock library before calling a WinSock function, the function will return a SOCKET_ERROR and the error will be WSANOTINITIALISED. Loading the WinSock library is accomplished by calling the WSAStartup function. All the WinSock applications must call the WSACleanup function before it exists.

Conclusion

6. Conclusion

A number of significant trends can be observed in the computing environmen over the last decade:

- Desktop machines are getting smaller, faster and cheaper;
- Software is getting more sophisticated and easier to use;
- Networks are getting more robust, faster and wider, and are extending to th farthest reaches of the globe through various technologies;
- Problems deemed amenable to realistic investigation are getting much mor complex in nature, and larger in terms of both size and number of cycle needed to address them.

All of these and more contribute to the growth of interest and investment i distributed computing, specifically on *Clusters*.

This project attempts to revoke the complexities that exist in paralle programming for clusters. It provides a viable environment for developing an executing parallel applications with ease, yielding levels of abstraction to th developer and the user.

Scope for Future Development

7. Scope for future development

This session briefs about the possible enhancements that can be done to improve the scope of this system.

7.1 Process/Task Migration:

Migrating an executing job from one workstation to another during runtime may be an added advantage. It is often used when owner takes back control of his/her workstation. Here the job running on the workstation will be suspended first, and then migrated onto another workstation after a certain time interval. Another use for job migration is to move jobs around to load balance the cluster. Forthcoming versions of this software would try to support this feature.

7.2 Support for Heterogeneous Architectures:

Though Windows is the only operating system supported by CCE in its first version, other platforms like Linux and Mach can be accommodated in future versions.

Appendix

8.1 Basic Programming Techniques

This session contains some basic and general parallel programming techniques for clusters. This material provides background that may be helpful in designing and building parallel applications using software for cluster programming like CCE, MPI and PVM.

8.1.1 Introduction

Developing applications for the CCE system-in a general sense, at least-follows the traditional paradigm for programming distributed-memory multiprocessors such as the nCUBE or the Intel family of multiprocessors. The basic techniques are similar both for the logistical aspects of programming and for algorithm development. Significant differences exist, however, in terms of (a) task management, especially issues concerning dynamic process creation, naming, and addressing; (b) initialization phases prior to actual computation; and (c) granularity choices. In this session, we discuss the programming process for CCE and identify factors that may impact functionality and performance.

8.1.2 Parallel Programming Paradigms:

Parallel computing using a system such as CCE may be approached from three fundamental viewpoints, based on the organization of the computing tasks. Within each, different workload allocation strategies are possible and will be discussed later in this session. The first and most common model for CCE applications can be termed ***crowd*** computing: a collection of closely related processes, typically executing the same code, perform computations on different portions of the workload, usually involving the periodic exchange of intermediat results. This paradigm can be further subdivided into two categories:

- The master-slave (or host-node) model in which a separate *contro* program termed the master is responsible for process spawning initialization, collection and display of results, and perhaps timing o functions. The slave programs perform the actual computation involved they either are allocated their workloads by the master (statically o dynamically) or perform the allocations themselves.
- The node-only model where multiple instances of a single program execute, with one process (typically the one initiated manually) taking ove the non-computational responsibilities in addition to contributing to the computation itself.

We note that these three classifications are made on the basis of process relationships, though they frequently also correspond to communication topologies. Nevertheless, in all three, it is possible for any process to interact and synchronize with any other. Further, as may be expected, the choice of model is application dependent and should be selected to best match the natural structure of the parallelized program.

8.1.3 Workload Allocation:

In the preceding section, we discussed the common parallel programming paradigms with respect to process structure, and we outlined representative examples in the context of the CCE system. In this section we address the issue of workload allocation, subsequent to establishing process structure, and describe some common paradigms that are used in distributed-memory parallel computing. Two general methodologies are commonly used. The first, termed ***data decomposition*** o partitioning, assumes that the overall problem involves applying computationa operations or transformations on one or more data structures and, further, that thes data structures may be divided and operated upon.

8.1.3.1 Data Decomposition

As a simple example of data decomposition, consider the addition of tw vectors, A[1..N] and B[1..N], to produce the result vector, C[1..N]. If we assume tha P processes are working on this problem, data partitioning involves the allocation o N/P elements of each vector to each process, which computes the corresponding N/ elements of the resulting vector. This data partitioning may be done either *statically* where each process knows *a priori* (at least in terms of the variables N and P) its share of the workload, or *dynamically*, where a control process (e.g., the maste process) allocates subunits of the workload to processes as and when they becom free. The principal difference between these two approaches is *scheduling*. Wit static scheduling, individual process workloads are fixed; with dynamic scheduling they vary as the computation progresses. In most multiprocessor environments, stati scheduling is effective for problems such as the vector addition example; however, in the general CCE environment, static scheduling is not necessarily beneficial. The reason is that CCE environments based on networked clusters are susceptible to external influences; therefore, a statically scheduled, data-partitioned problem might encounter one or more processes that complete their portion of the workload much faster or much slower than the others. This situation could also arise when the machines in a CCE system possess varying CPU speeds and different memory and other system attributes.

In a real execution of even this trivial vector addition problem, an issue tha cannot be ignored is input and output. In other words, how do the processe described above receive their workloads, and what do they do with the resul vectors? The answer to these questions depends on the application and th circumstances of a particular run, but in general:

1. Individual processes generate their own data internally, for example, usin random numbers or statically known values. This is possible only in very specia situations or for program testing purposes.

2. Individual processes independently input their data subsets from externa devices. This method is meaningful in many cases, but possible only when paralle I/O facilities are supported.

3. A controlling process sends individual data subsets to each process. This i the most common scenario, especially when parallel I/O facilities do not exist Further, this method is also appropriate when input data subsets are derived from a previous computation within the same application.

The third method of allocating individual workloads is also consistent wit dynamic scheduling in applications where interprocess interactions durin computations are rare or nonexistent. However, nontrivial algorithms generall require intermediate exchanges of data values, and therefore only the initia assignment of data partitions can be accomplished by these schemes.

8.2 API Manual

8.2.1 Introduction

This session comprises the manual pages of some of the APIs provided by the CCE system. *Name, Syntax, Parameters* and *Description* are elaborated for each of the specified APIs.

Name:

CCE_AddHosts - Add hosts to the virtual environment.

Syntax:

int info = CCE_AddHosts(char *hosts)

Parameters:

hosts: An array of strings naming the hosts to be added.

Discussion:

The routine *CCE_AddHosts* adds the computers named in *hosts* to the configuration of computers making up the virtual environment.

If CCE_AddHosts is successful, *info* will be equal to *nhost*. Success is indicated by 0, and failure by info < 1.

Name:

CCE_Exit – Shuts down the entire CCE system

Syntax:

int info = CCE_Exit(int pgm)

Parameters:

info: Integer returning the error status.

Pgm: Identifier for master and worker tasks

Discussion:

The routine *CCE_Exit* shuts down the entire CCE system including remote tasks, , the local tasks and the *Manager* except the *master* task. This routine should be called only by the *master* task, and it should be last CCE routine executed by it.

Name:

CCE_DelHost - Deletes hosts from the virtual environment.

Syntax:

int info = CCE_DelHost(char *hosts)

Parameters:

hosts: An array of pointers to character strings containing the names of the machines to be deleted.

info: Integer status code returned by the routine. Value 0 means success and 01 means failure.

Discussion:

The routine *CCE_DelHosts* deletes the node pointed to in by *hosts* from th existing configuration of node making up the virtual environment. All CCE task running on these computers are killed as the node name is deleted.

If a host fails, the CCE system will continue to function and will automaticall delete this host from the virtual environment. An application can be notified of host failure by calling CCE_Notify. It is still the responsibility of the applicatio developer to make his application tolerant of host failure.

Name:

CCE_Final – Tells the local *Agents* and *Manager* that this task is leavin CCE.

Syntax:

int info = CCE_Final (void)

Parameters:

info: Integer status code returned by the routine. Values of -1 indicates a error.

Discussion:

The routine *CCE_Final* tells the local *agent (CCEA)* and the *manage* (CCEM) that this process is leaving CCE. This routine does not kill the process which can continue to perform tasks just like any other serial process.

All CCE processes (except the *master* task) should call this routine befor they stop or exit. It checks out the calling task from the CCE system.

Name:

CCE_MyTid – Returns the *tid* of the calling process.

Syntax:

int tid = CCE_MyTid(void)

Parameters:

tid: Integer returning the task identifier of the calling CCE process. Values less than zero indicate an error.

Discussion:

The routine *CCE_MyTid* returns the task ID of the calling process generated by the local *agent*. Value -1 indicates an error.

Name:

CCE_Parent – Returns the *tid* of the task.

Syntax:

int tid = CCE_Parent(void)

Parameters:

tid: Integer returning the task identifier of the parent the calling CCE task. Value -1 indicates an error.

Discussion:

The routine *CCE_Parent* returns the task ID of the task from the *agent*. Value -1 indicates an error.

Name:

CCE_Recv – Receives data from another CCE task.

Syntax:

int info = CCE_Recv(int *tid, void *buff, int *len, int mesgid)

Parameters:

tid: Pointer to task identifier of CCE task to that sent the data.

Buff: Address of the buffer that receives the data

len: Pointer to the length of the received data in buffer

mesgid: Determines which application has called the function.

info: Integer status code returned by the routine.

Discussion:

The routine *CCE_Recv* receives the data in the location pointed by *buff* sent by another CCE task identified by **tid*. If CCE_Recv is successful, *info* will be 0. If some error occurs then *info* will be -1. The pointer len should be initialized to the size of the buff allocated.

Name:

CCE_Send - Sends data to another CCE task.

Syntax:

int info = CCE_Send(int tid, void * buff, int len, int mesgid)

Parameters:

tid: Integer task identifier of CCE task to receive the data.

buff: Address of the buffer that contains the data to be sent

len: Length of the data in buffer to be sent

mesgid: Determines which application has called the function.

info: Integer status code returned by the routine.

Discussion:

The routine *CCE_Send* sends the data pointed by *buff* to the CCE task identified by *tid*. If CCE_Send is successful, *info* will be 0. If some error occurs then *info* will be -1. The receiving task should use CCE_Recv() routine to accept the data sent using CCE_Send().

Name:

CCE_Spawn - Starts new CCE task.

Syntax:

int numt = CCE_Spawn(char *task, int ntask, int *tids)

Parameters:

task: Character string which is the executable file name of the CCE task to be started. The executable must already reside on the same directory as the master program.

where: Character string specifying where to start the CCE process Depending on the value of *flag*, *where* can be a host name such a ``busybee.hcl.net " or a CCE architecture class such as ``SUN4". If *flag* is 0 then *where* is ignored and CCE will select the most appropriate host.

ntask: Integer specifying the number of copies of the executable to start.

tids: Integer array of length *ntask* returning the tids of the CCE processes started by this CCE_Spawn call.

numt: Integer returning the success of the function. Value –1 indicates failure and value of 0 indicates success.

Discussion:

If CCE_Spawn starts one or more tasks given in the function across all agents. If a system error occurs then *numt* will be -1.

Name:

CCE_InitSys – Starts the CCE system.

Syntax:

int sts = CCE_*InitSys* (void)

Parameters:

sts: Integer returning whether an error occurred. A value of 0 indicates success and –1 indicates failure.

Discussion:

The routine *CCE_InitSys* routine starts the CCE system by starting the *manager* process. This is the first CCE routine executed by a CCE system.

Name:

CCE_Wait – Waits for termination of a specified CCE process (task).

Syntax:

int info = CCE_Wait(int tid)

Parameters:

tid: Integer task identifier of the CCE process to be waited on.

info: Integer status code returned by the routine. Value -1 indicates an erro and 0 indicates a success.

Discussion:

The routine *CCE_Wait* waits until the termination of the CCE process identified by *tid*. If CCE_Wait is successful, *info* will be 0. If some error occurs then *info* will be-1.

CCE_Wait is not designed to wait on the calling process and may cause indefinite wait if called on the same task.

8.3 Message Description

Name:

MN2AG_INITSYS

Format:

Message	

Sender:

Manager

Processed By:

Agent

Description:

This is the message sent by a manager to the *agents*. The response to this message is the IP address of the agent system.

Name:

MN2AG_PGMNAME

Format:

Message	**Program name**

Sender:

Manager

Processed By:

Agent

Description:

This is the message sent by a manager to the *agents* in the node list. This message contains the name of the worker program.

Name:

TS2AG_REQTID

Format:

Message	

Sender:

Task

Processed By:

Agent

Description:

This is the message sent by a task to the local *agent*. The local agent replies by sending the task id of that task to it.

Name:

MN2AG_TIDNO

Format:

Message	**Higher order value of taskid**

Sender:

Manager

Processed By:

Agent

Description:

This is the message sent by a manager to the *agent*. This message contains the higher order bit of the taskid.

Name:

TS2MN_TIDINFO

Format:

Message	**Task id**

Sender:

Task

Processed By:

Manager

Description:

This is the message sent by a task to the manager.. This message contains the full task id of that task.

Name:

MS2MN_SYSINIT

Format:

Message	

Sender:

Master

Processed By:

Manager

Description:

This is the message sent by master to the manager. On receiving this message the manager calls the cce_initsys().

Name:

MS2MN_SPNREQ

Format:

Message	Program name	Number of copies

Sender:

Master

Processed By:

Manager

Description:

This is the message sent by a master to the manager. This message contains the program name that is to be spawned and the number of copies to be spawned.

Name:

TS2MN_PRTINFO

Format:

Message	tidno	PortNumber

Sender:

Task

Processed By:

Manager

Description:

This is the message sent by a task to the manager.. This message contains the port number of the task to which it is binded and the task number.

Name:

MS2MN_DATA

Format:

Message	**Task id**	**Array of datas**

Sender:

Master

Processed By:

Manager

Description:

This is the message sent by master to the manager.. This message contains the data and the task id to which it is to be sent.

Name:

MN2TS_DATA

Format:

Message	**Array of datas**

Sender:

Manager

Processed By:

Task

Description:

This is the message sent by manager to the task.. This message contains the data which it is to be ment for that task.

8.4 SAMPLE FLOW

The flow of the system can be explained with the following example.

Master.c

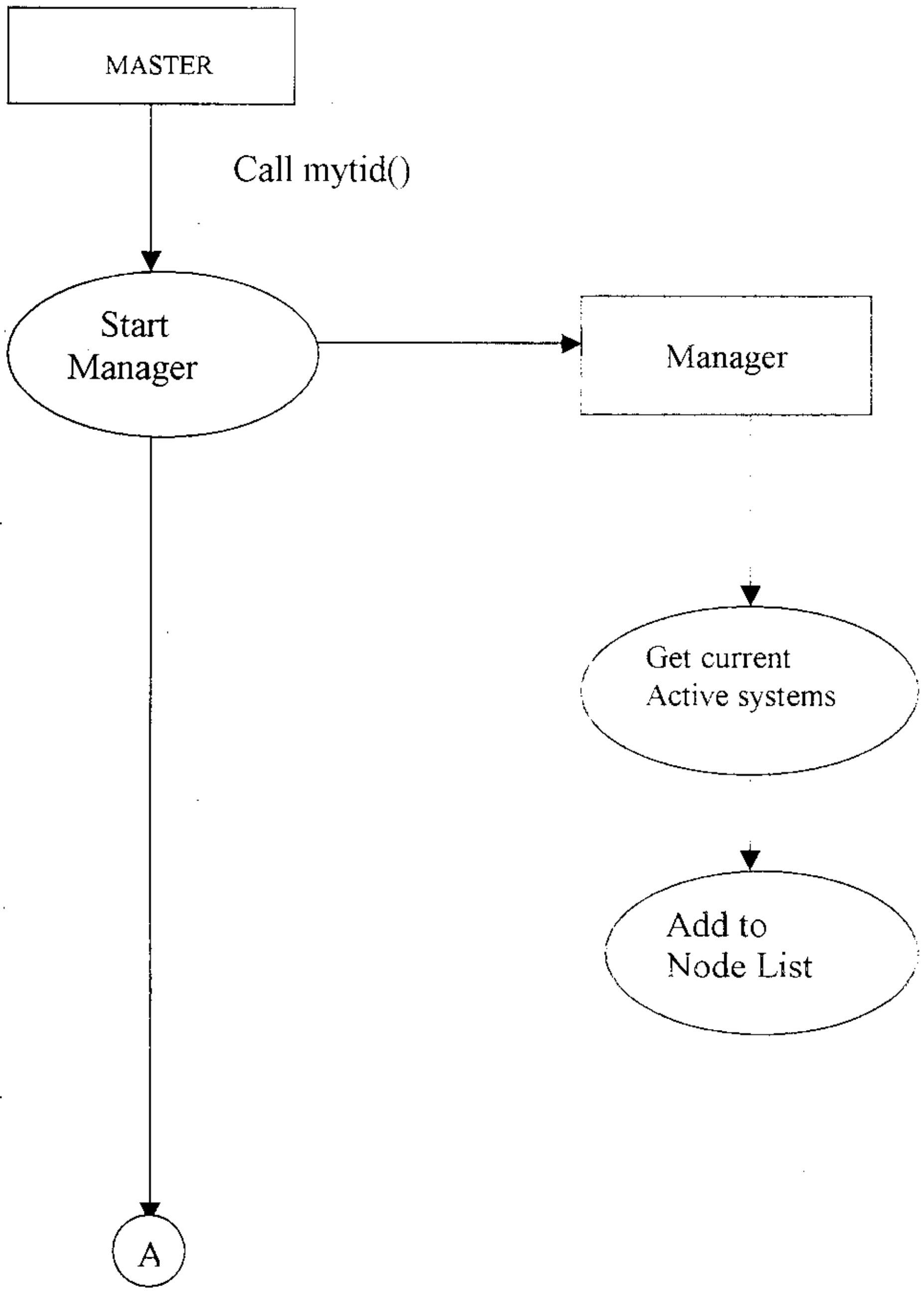
```
#include "cce.h"
#include <stdio.h>
#define SZ 1000
#define NPROCS 5

main()
{
 int mytid, task_ids[NPROCS];
 int a[SZ], results[NPROCS], sum = 0,rlen;
 int i, msgtype, num_data = SZ/NPROCS;
 int s,r,sp,e;
  mytid = CCE_MyTid();

  for (i = 0; i < SZ; i++)
  a[i] = i % 25;

  sp=CCE_Spawn("worker", NPROCS, &task_ids);

  for (i = 0; i < NPROCS; i++)
s=CCE_Send(task_ids[i],&a,num_data,1);

  for (i = 0; i < NPROCS; i++)
  {
    r=CCE_Recv(task_ids[i],&results[i],rlen,1);
    sum += results[i];
  }
```

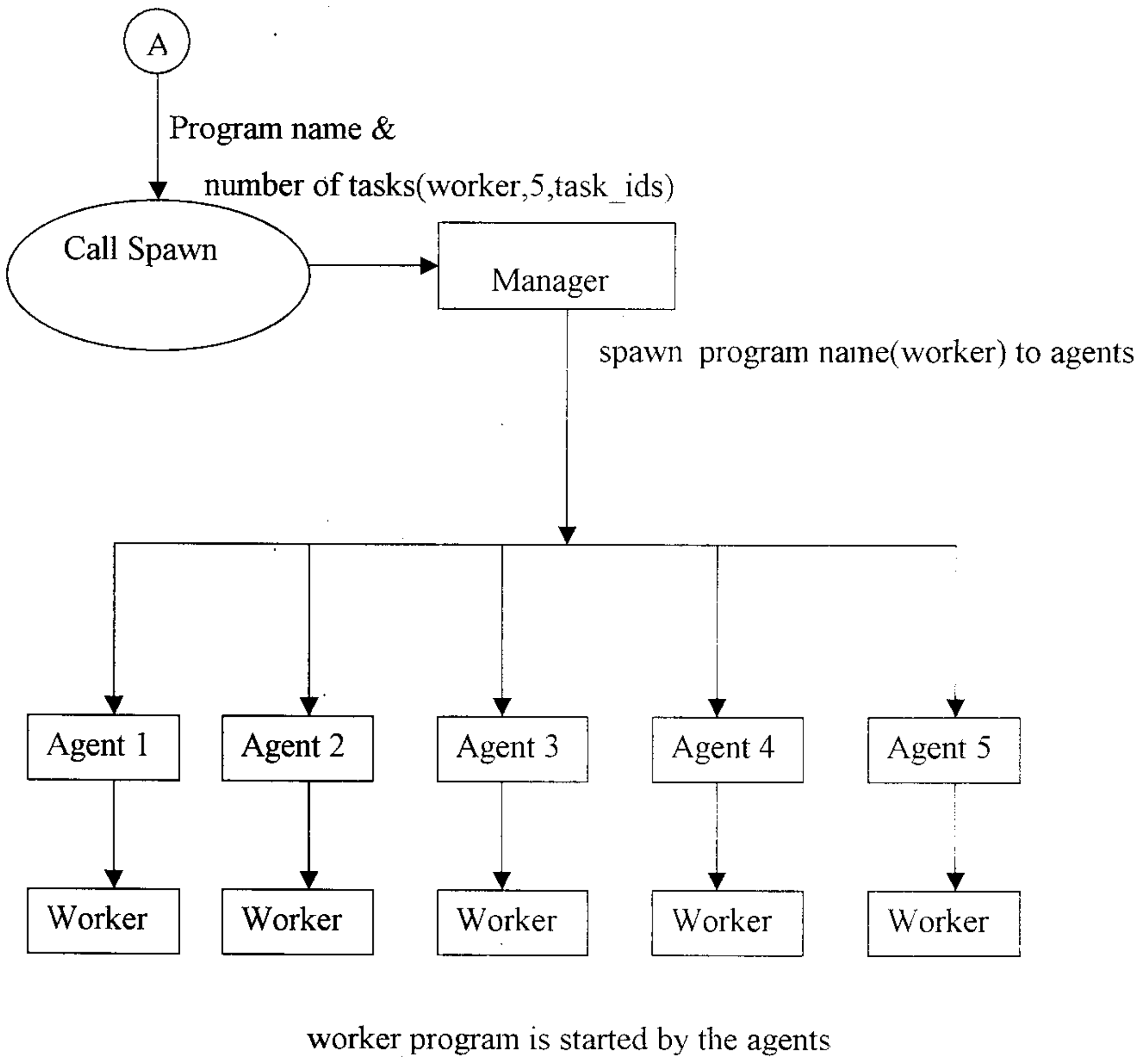
```
    printf("The sum is %d \n",sum);

    e=CCE_exit(1);
 }
```

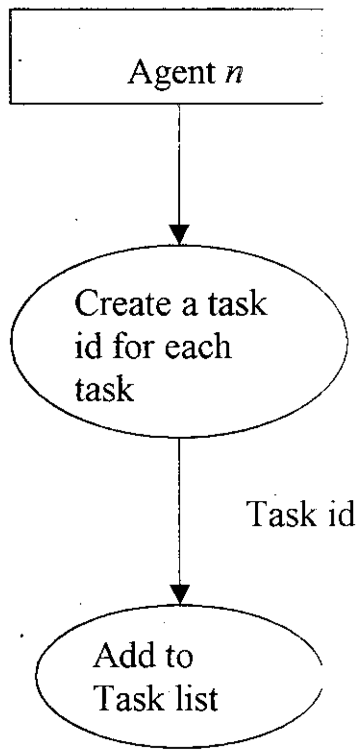
Worker.c

```
#include "cce.h"
#include <stdio.h>

main()
{
  int mytid;
  int sw,rw,spw,ew;
  int i, sum, a[1000];
  int num_data, master,td;

  mytid = CCE_Parent().

  rw=CCE_Recv(td,&a,num_data,rlen,2);

  sum = 0;
  for(i = 0; i < num_data; i++)
  sum += a[i];

  sw=CCE_Send(mytid,&sum,sizeof(sum),2);

  ew=CCE_Exit(2);
}
```

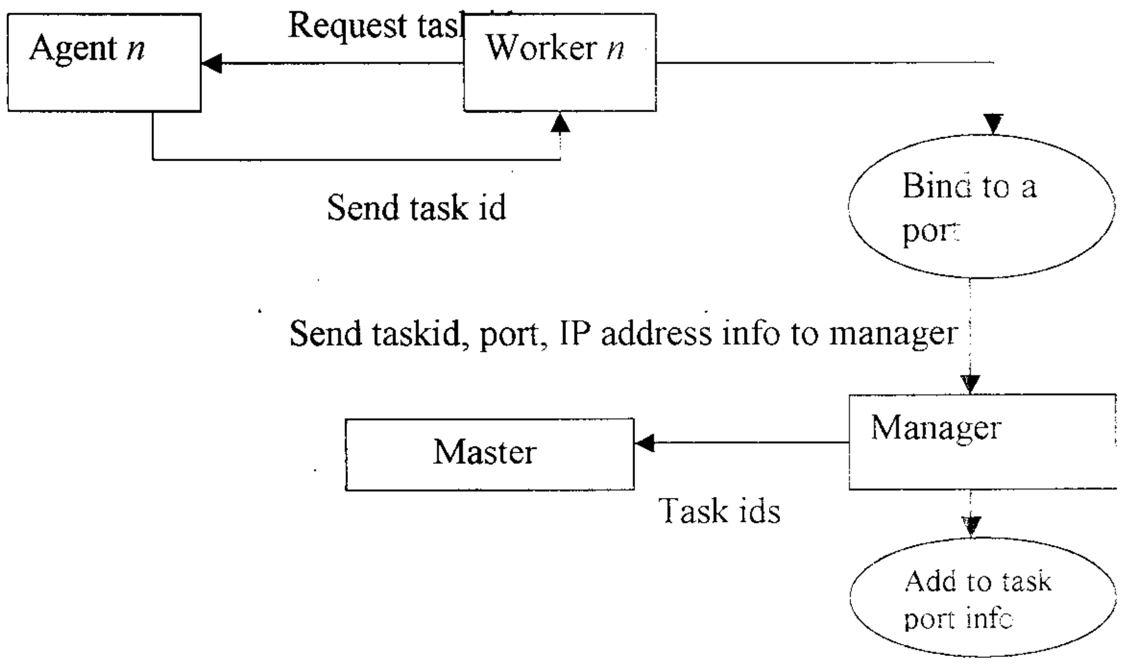
System flow for the example



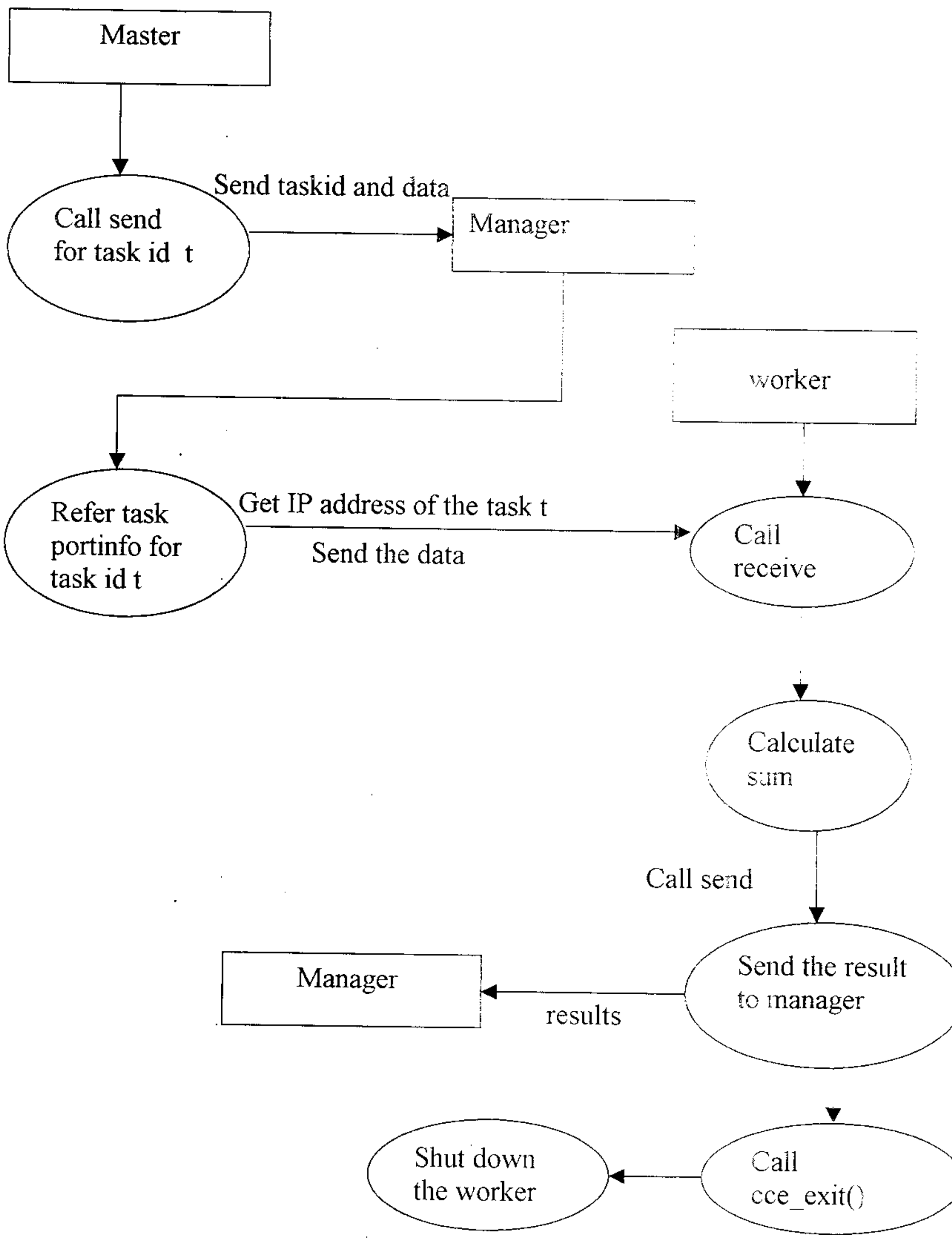

A
Program name &
number of tasks(worker,5,task_ids)
Call Spawn
Manager
spawn program name(worker) to agents
Agent 1
Agent 2
Agent 3
Agent 4
Agent 5
Worker
Worker
Worker
Worker
Worker
worker program is started by the agents

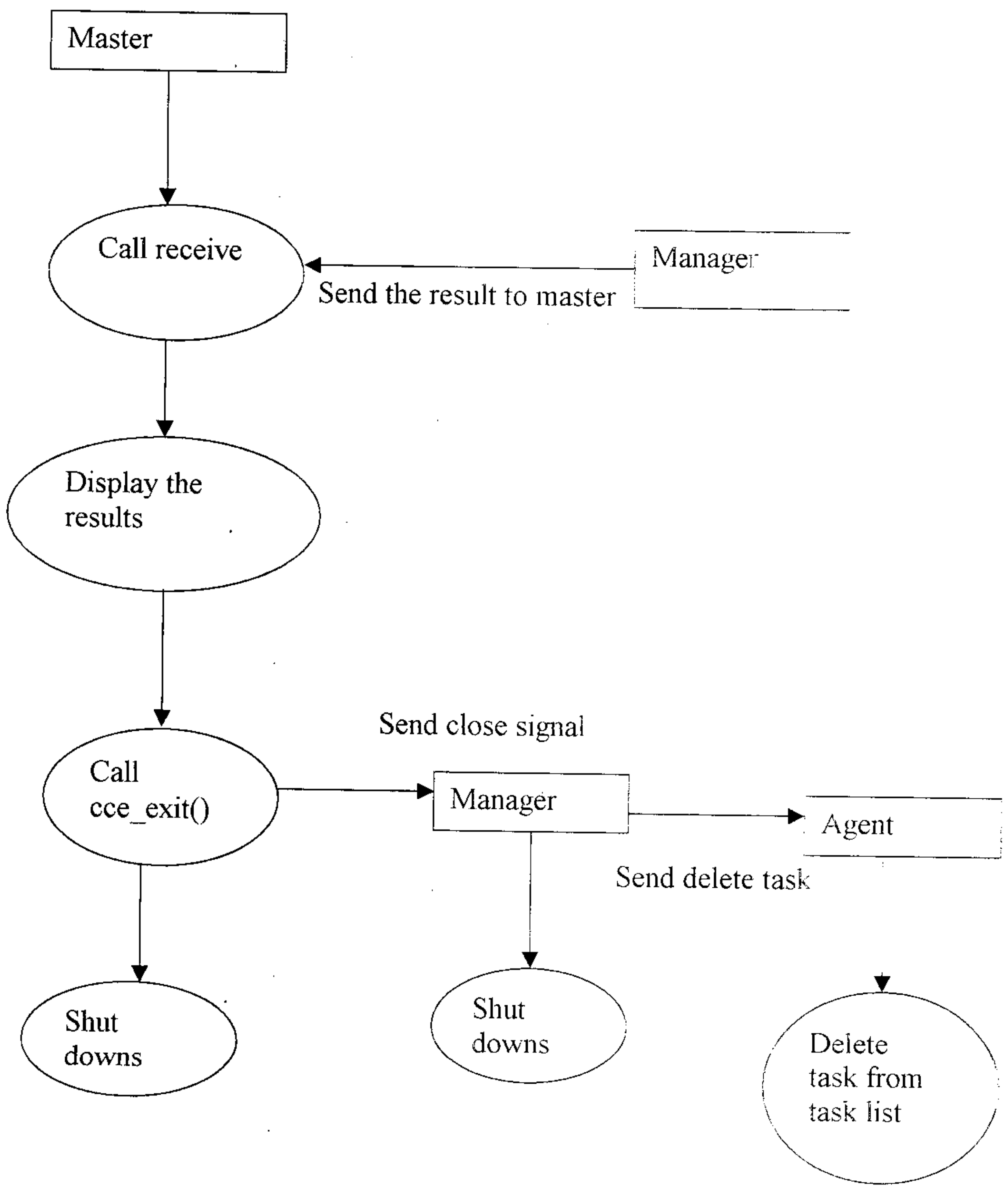


Agent *n*
Create a task id for each task
Task id
Add to Task list



Agent *n*
Request tas
Worker *n*
Send task id
Bind to a por
Send taskid, port, IP address info to manager
Master
Manager
Task ids
Add to task port info


Master
Call send for task id t
Send taskid and data
Manager
worker
Refer task portinfo for task id t
Get IP address of the task t
Send the data
Call receive
Calculate sum
Call send
Send the result to manager
Manager
results
Shut down the worker
Call cce_exit()


Master
Call receive
Manager
Send the result to master
Display the results
Send close signal
Call cce_exit()
Manager
Agent
Send delete task
Shut downs
Shut downs
Delete task from task list

8. BIBLIOGRAPHY

1."**WinSock 2.0**" by Lewis Napper, Comdex Computer Publishing

2. "**Network Programming for Microsoft Windows**" by Anthony Jones and Jim Ohuld, Microsoft Press

3. "**Windows 2000 Advanced API & System Programming Black Book** ", AI Williams, Dreamtech Press

4. "**Windows NT Clustering**" by Mark. A. Sportack, SAMS Publishing